Hybrid GPU / CPU Navier-Stokes lattice Boltzmann method for urban wind flow

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Science and Engineering

2021

By
Marta Camps Santasmasas
School of Mechanical, Aerospace and Civil Engineering

Contents

List of Tables

List of Figures

Nomenclature

Recurring abbreviations and symbols are summarised here.

Abbreviations

BEM	Boundary element method
BGK	Bhatnagar-Gross-Krook
CFD	Computational fluid dynamics
CPU	Central processing unit
CWE	Computational wind engineering
DNS	Direct numerical simulation
FLIP	Fluid implicit particles
FSI	Fluid-structure interaction
GASCANS	GPU-Accelerated solver for coupled approaches to Navier-Stokes
GPU	Graphics processing unit
LB	Lattice Boltzmann
LB-LES	Lattice Boltzmann large eddy simulation

LES	Large eddy simulation
LHS	Left hand side
LUMA	Lattice Boltzmann at the University of Manchester
MMSF	Multiscale modelling and simulation framework
MRT	Multi relaxation time
NS	Navier-Stokes
NSLB	Navier-Stokes lattice Boltzmann
PIC	Particle in cell
PISO	Pressure implicit splitting operator
RANS	Reynolds-averaged Navier-Stokes
RHS	Right hand side
SEL	Scale separtaion loop
SEM	Synthetic eddy method
SPH	Smoothed particle hydrodynamics
SSM	Scale separation map

Dimensionless Quantities

Re	Reynolds number

Greek Symbols

$\bar{\rho}$	LES filtered density
Δt	Time step
δt	Time step in physical or dimensionless units
δx	Cell size in physical or dimensionless units
Δ	LES filter lenght
ϵ	Turbulent kinetic energy dissipation rate
ν	Kinematic viscosity
ν_t	Smagorinsky eddy viscosity/subgrid scale viscosity
Ω	Boltzmann colision operator
ρ	Density
σ_i	Turbulent length scale in the direction i
σ_{ij}	Cauchy stress tensor
τ	Relaxation time
τ_{ij}^d	Deviatoric stress tensor
τ_{ij}	Subgrid scale Reynolds stress
ξ_i	Fluid particle velocity

Roman Symbols

$\mathbf{c}_\alpha$	Discretised particle velocity in the direction α

$\mathbf{U}$	Time averaged macorscopic velocity
$\mathbf{u}$	Instantaneous macroscopic velocity
$\mathbf{u}'$	Fluctuating macroscopic velocity
$\overline{\mathbf{u}}$	LES filtered velocity
$\overline{f}_\alpha$	LES filtered particle distribution function
$\overline{p}$	LES filtered pressure
C_s	Smagorinsky model constant
c_s	Lattice speed of sound
f	Particle distribution function
f_α^*	Post-streaming discretised particle distribution function in the direction α
f^{eq}	Equilibrium particle distribution function
f_α^{eq}	Discretised equilibrium particle distribution function in the direction α
f_α	Discretised particle distribution function in the direction α
F_i	Body force component in the i direction
p	Instantaneous pressure
R_{ij}	Reynolds stress tensor

Abstract

Hybrid GPU / CPU Navier-Stokes lattice Boltzmann method for urban wind flow

Marta Camps Santasmasas

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy, 2021

The wind flow through an urban built environment has a significant impact on the safety and comfort of pedestrians and inhabitants. Simulation of urban wind flow presents a formidable challenge to computational engineering field due to the complex geometry of the built environment and the multiscale nature of the flow. Numerical analysis via computational fluid dynamics (CFD) based on the Navier-Stokes equations is now commonplace in the industry, usually run on high performance computer clusters of CPU nodes. However, resolution of all the turbulent scales of motion in the entire domain is likely to remain beyond the reach of such hardware for the foreseeable future; thus, so called Direct Numerical Simulation is not practical for industrial applications. Moreover, the region of interest usually represents a small percentage of the total volume of the domain.

The objective of this thesis is to achieve a time-dependent simulation that resolves large to medium turbulence scales in the region of interest at a significantly reduced computational cost compared to turbulent scale resolving Navier-Stokes methods. To do so, we couple a lattice Boltzmann (LB) solver running on graphics processing units (GPUs) with a Navier-Stokes (NS) solver running on CPUs. The LB solver incorporates the mean turbulent flow information from the Navier-Stokes model into its resolved velocity via a synthetic eddy method (SEM) implemented at the inlet of the LB domain. The resulting coupled Navier-Stokes lattice Boltzmann (NSLB) solver combines the accuracy and computing speed of the GPU implementation of the LB model with the stability, low memory consumption and mesh flexibility of the NS solver. Moreover, the NSLB model exploits the widespread availability of CPU and GPU hardware on desktop, and

workstation computers.

Validation results of the two-way coupled NSLB solver for laminar flow demonstrate that the coupled solver is able to reproduce the results of the full domain single solver (either LB or NS) independently of the position of the interface between the solvers. For the application to urban wind flow, we embedded our lattice Boltzmann large eddy simulation (LB-LES) solver within a pre-calculated Reynolds averaged Navier-Stokes (RANS) simulation of flow around a single building at $Re_H = 47893$. The LB-LES model accurately predicts the mean and fluctuating velocity around the building, increasing the flow data available and its accuracy with respect to the underlying RANS simulation. The RANS LB-LES coupling allows fully resolved LES results to be achieved in practical time scales with a single desktop based GPU, which shows potential to run industry applications on consumer devices.

The original contributions of this work include the development of the coupling framework, adaptation of the SEM to LB and refinement of the embedded boundary conditions. The resulting solver fully realises the objective of a coupled Navier-Stokes method with a GPU accelerated lattice Boltzmann method in order to accurately simulate high Reynolds number flow at affordable computational cost.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like first to thank Alistair Revell for his guidance and support through my PhD. His approachable, sincere nature and his ingenuity made our meetings productive and enjoyable. Also I would like to thank Ben Parslew for all his help in developing and structuring my PhD even when the topics of my PhD were far from his research topics. I would also like to thank Ben for his valuable advice on presentations, especially how to make them appealing for the audience.

To my colleagues at the University of Manchester, thank you for being there and keeping me company during the long days at the office. I would especially like to thank Sam for all our mutual help and joyful sharing of our little findings and new colourful plots; and Alex and J, for all their support and encouragement. Finally, thank you Sijiang for all your help and hard work.

And I'm not forgetting the MACE PGR society, I'm grateful for all the entertaining coffee mornings and hiking trips; George Begg would be a much sadder place without you.

To Zach, thank you for your support and encouragement and for living with me during all the lockdown trying to make the minimum amount of noise possible while I spend hours and hours in front of the computer writing this thesis. Also to Robert "Bobby" Camps, the cat, for keeping me company with his soft fur and purrs during all these years.

Finally, I would like to thank my family and friends in Spain for their support and good conversations, even when we are so far apart.

Close your eyes and turn your
face into the wind.
Feel it sweep along your skin in an
invisible ocean of exultation.
Suddenly, you *know* you are *alive*.

Vera Nazarian, The Perpetual
Calendar of Inspiration

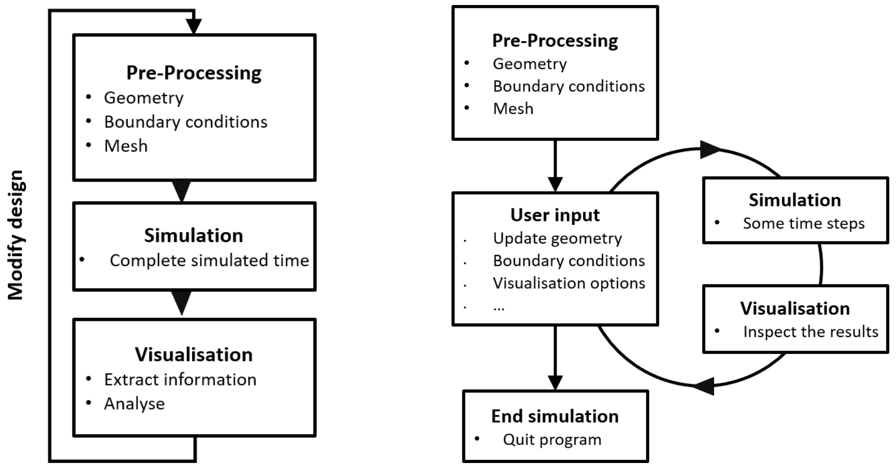
Chapter 1

Introduction

We live immersed in air. Its behaviour affects every aspect of our lives, from walking along a street, to flying a drone, driving a car or the dispersal of airborne contaminants and the evolution of forest fires. One of the environments in which we feel most the effects of wind and in which we can affect more the local wind climate is in urban environments. Buildings and new urban constructions are affected by the local wind climate and, in return modify the wind climate of the region they are in.

Wind Engineering studies the effects of wind in the local environment and assesses the potential benefits, dangers and discomforts that can be caused by the wind. A subgroup inside wind engineering is computational wind engineering (CWE) [11], which uses computational fluid dynamics (CFD) to approximate the equations that describe the behaviour of the wind, namely the Navier-Stokes equations. The first ingredient needed for a CFD simulation in urban wind is a 3D digital model of the interest buildings. Then, the engineer decides the volume of air around the city they want to model which they generally proceed to divide up into cells. The CFD solver is then employed to approximate the wind velocity and pressure in each cell using a discretised form of the Navier-Stokes equations. Finally, the last step is to post-process the results from the model and extract useful information (Fig. 1.1).

The CFD procedure itself is reasonably well established, but its accuracy is heavily dependent on the quality and quantity of information available on the simulated region: wind profiles at the boundaries, quality of the geometry data and wind climate amongst others. Note that the wind characteristics at the boundaries of the computational domain depend on the wind climate of the region and all the surrounding geometry not included in the CFD domain. Thus, obtaining an accurate representation of the wind at the boundaries of the domain is only feasible in simple cases, and generally, they need to be estimated. The misrepresentation of the flow introduced by this estimation is often high, and as such a much larger region must generally be modelled than is needed, in order to overcome these limitations [12]. A better representation of the wind flow at the boundaries would allow to move the boundaries closer to the region of interest, thus reducing the size of the domain and its computational cost. Moreover, wind is a turbulent physics process, which is characterised by a range of temporal and spatial scales. The majority of these scales must be represented in the computational grid to accurately model the wind behaviour. Indeed, these characteristics of the wind plus the size of the domain which greatly impacts on the cost of CFD simulations for the urban environment.



(a) Iterative design. (b) Interactive design.

Figure 1.1: (a) Design approach used by traditional CWE; (b) Design approach for interactive CWE.

The traditional computational cycle used in CWE can be considered to be an iterative design process (Fig. 1.1a). Iterative design is a linear process in which the simulation results are evaluated once the simulation step is finished and results are available for the complete amount of time simulated. Once the simulation is completed, the results can be post-processed and the design or simulation parameters adjusted if needed before preparing and running the new simulation with the new parameters. An alternative to iterative design is interactive design (Fig. 1.1b). Interactive simulation allows the user to modify the inputs of the model and to examine the obtained results while the simulation is running. Interactive

CWE would enable an engineer to iteratively change the design of a building while the simulation continues to run until the desired surrounding wind field is obtained. This approach increases the knowledge and control the engineer has over the effect of the building on the local wind and reduces the time needed to obtain a satisfactory design.

Moreover, CWE could benefit from computational times that allow for faster than real-time simulations (Fig. 1.2). Real time or faster than real time urban wind modelling could be used by emergency response teams to predict the spread of an airborne contaminant. It could also allow a drone to evaluate and avoid regions in which high levels of turbulence could damage the aircraft. Real-time and faster than real time multi-scale resolving CWE is still out of reach for our current modelling tools and computational resources. However, the CWE industry will benefit from any advances in reducing the computational time and resources spent by multi-scale resolving wind CFD.

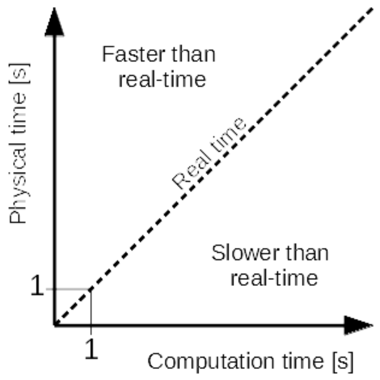


Figure 1.2: Physical time modelled vs. the computation time required to model it.

The existing computational models that can be applied to real-time wind modelling are widely used in the computer animation industry due to their low resource consumption and stability. They are varied in their approaches to discretise the domain, in the set of equations solved and how they are discretised. Some examples include Smoothed Particle Hydrodynamics (SPH), which discretises the Lagrangian Navier-Stokes equations and divides the domain in moving particles, semi-Lagrangian methods based on Stable Fluids [5], which discretises the Eulerian Navier-Stokes equations and divides the domain into cells, and fluid implicit particles (FLIP) [13] which discretises the Navier-Stokes equations and combines particles to model convection with a grid to model viscosity and time derivatives. The CFD models traditionally used in CWE discretise the Eulerian Navier-Stokes equations and divide the domain in cells [12]. The Eulerian Navier-Stokes solvers are more accurate and able to model a wider range of flows and scales than the semi-Lagrangian Navier-Stokes models; however, their algorithm is more complex and consumes more computational resources. In recent years a relatively new approach is emerging as an alternative for this sector, the lattice Boltzmann (LB) method. The LB method views the fluids from a mesoscale perspective instead of the macroscale perspective of the Navier-Stokes equations but is able to recover the behaviour of the flow as the Navier-Stokes equations for low Mach number weakly compressible flows [1]. The main advantages of the LB method lie in the simplicity and locality of its algorithm, which render it significantly more efficient when implemented in massive parallel architectures as graphics processing units (GPUs), and the low numerical dissipation of its convection scheme, which maintains turbulence at relatively coarse cell sizes. However, it is conditionally stable and requires a constant cell size to maintain computational efficiency.

Each of the presented CFD methods can be implemented on its own or two or more models can be coupled together. The objective of the coupling is usually to model a multi-physics process, modelling each process with a different model (e. g. Keyes et al. [14]). Another reason is to reduce the computational cost of the simulation by combining the strengths of each of the component models and compensating for their weaknesses. A family of coupled models used in CWE is the RANS-LES models [15]; where the regions of the domain that need flow detail are solved using a turbulence resolving model, usually large eddy simulation (LES), while the rest of the domain is solved by a less computationally expensive mean flow resolving method, usually Reynolds averaged Navier-Stokes equations (RANS). The most common RANS-LES solvers implement the Eulerian Navier-Stokes equations with different turbulence models on CPU architectures. On the other hand, Navier-Stokes based solvers can be coupled to lattice Boltzmann solvers to also save computational resources (e. g. Neumann [8] and Tong and He [9]).

The lattice Boltzmann method is able to resolve turbulent flow to the same accuracy as the Eulerian Navier-Stokes models used in CWE [16]; moreover it is efficient in GPU, which reduces its computational cost and allows it to run in consumer grade computers. However, the size of the domain is limited due to the limited memory resources contained in GPU cards. On the other hand RANS solvers are less computationally expensive than Navier-Stokes LES solvers and can run large domains with relatively low CPU requirements. The motivation for this work results from the combination of these with the general observation that domains generally modelled in CWE are much larger than the area of interest, combining RANS grid flexibility and low computational CPU cost with LB-LES speed and accuracy around the region of interest might be a promising way to reduce the cost of urban wind simulations, while still resolving the details of the flow in the region of interest. Thus, the objectives of this thesis are twofold:

- Expand the use of the GPU-accelerated lattice Boltzmann method in CWE.
- Develop a coupled Navier-Stokes lattice Boltzmann (NSLB) solver to reduce the computational resources needed by its two components, while achieving the desired accuracy in the interest region.

The work in this thesis presents a NSLB solver that can be coupled one way, the information travels from the Navier-Stokes domain to the lattice Boltzmann domain, and two-way, the information travels from the Navier-Stokes domain to the lattice Boltzmann domain and from the lattice Boltzmann domain to the Navier-Stokes domain. The two-way coupling is tested for laminar flow around a wall mounted cube and the one-way coupling is tested for turbulent flow around a rectangular building. The results for both test cases are promising, demonstrating that the NSLB is able to accurately predict the wind flow in the area of interest. Moreover, the NSLB simulations presented in this thesis run in a single node workstation, which shows potential to run CWE cases on consumer devices.

1.1 Thesis outline

This thesis is structured in journal format according to the University of Manchester regulations. The outputs of this work are presented as journal papers, which are appended at the end of the thesis. The main body of the thesis presents background information and a detailed explanation of the NSLB model. The chapters of this thesis are:

Chapter 1 contains the background, motivation and objectives of the research presented with an overview of the main findings.

Chapter 2 reviews the families of CFD methods that can be applied to wind engineering placing more emphasis on the lattice Boltzmann and Eulerian Navier-Stokes methods. It also provides a review of the different methods proposed until now to couple two or more solvers with particlar focus on coupling lattice Boltzmann with Navier-Stokes methods.

Chapter 3 describes in more detail the Eulerian Navier-Stokes method and the lattice Boltzmann method that form the NSLB solver developed in this thesis. It also describes the approach taken to simulate turbulent flow.

Chapter 4 describes the NSLB solver in more detail including the coupling algorithm, the communication between its components and how it is implemented.

Chapter 5 discusses the contributions to the field from the research in this thesis and how they are expressed in each of the appended papers.

Chapter 6 summarises the main findings of the present research, including the findings in the appended papers, and points at future work and future developments for the NSLB solver.

Appendices presents the main results and software derived from the research presented in this thesis in the form of 4 papers. Each paper and the contributions from each author are:

1. **M. Camps Santasmasas, A. J. Revell, B. Parslew. Two-way coupled Navier-Stokes / lattice Boltzmann solver to reduce the resources used by CFD simulations of flow around bluff objects. *In preparation*.**

   This work presents and tests the two-way coupled NSLB solver developed in this thesis for a low Reynolds number flow around a wall mounted cube.

   M. Camps Santasmasas developed the NSLB algorithm, wrote the paper and performed the simulations under the supervision and guidance from A. J. Revell and B. Parslew.

2. **M. Camps Santasmasas, X. Zhang, A. J. Revell. Synthetic eddy method applied to lattice Boltzmann model for wind around a rectangular prism building. In preparation**

This work applies the large eddy simulation lattice Boltzmann solver used as the LB component of NSLB model described in this thesis to turbulent flow around a rectangular building. The LES-LB solver is one-way coupled to an underlying pre-calculated RANS simulation. This paper illustrates a possible configuration of the one-way coupled NSLB model described in this thesis applied to a CWE setting.

M. Camps Santasmasas wrote the paper and performed the LES-LB simulations; X. Zhang conducted the Navier-Stokes based RANS and DDES simulations; A. Revell provided extensive guidance on simulation setup and writing and structure of the paper.

3. **A. R. G. Harwood, J. O'Connor, J. Sanchez Muñoz, M. Camps Santasmasas, A. J. Revell. LUMA: A many-core, Fluid-Structure Interaction solver based on the Lattice-Boltzmann method. *Software X 7 (2018) 88-94*.**

   This work describes and publishes the LUMA lattice Boltzmann code. LUMA is the original CPU based lattice Boltzmann code predecessor of the LB-LES code developed during this thesis. It was the first experience with a general purpose lattice Boltzmann code and it promted the development of its GPU version, GASCANS. GASCANS is the LB-LES code developed during this thesis.

   The main contributions of M. Camps Santasmasas to both the LUMA code and the corresponding paper are developing and testing the LES turbulence model and converting the input and output data between dimensionless and lattice units.

4. **M. Camps Santasmasas, A. R. G. Harwood, I. Hinder, S. Fan, B. Owen, J. O'Connor, A. J. Revell. GPU-Accelerated Solver for Coupled Approaches to Navier-Stokes (GASCANS). *In preparation***

   This work describes and publishes the GASCANS lattice Boltzmann code. GASCANS is the lattice Boltzmann solver used as the LB component of the NSLB solver described in this thesis. It is then the LB-LES code used to perform the LB simulations in paper 1 and 2.

M. Camps Santasmasas led the creation of the paper and its structure and drafted the abstract, introduction, conclusions, boundary conditions, turbulence sections and contributed to the architecture and BGK Lattice Boltzmann Method sections. She also performed the simulations and drafted the validation test cases for turbulent channel flow, SEM channel flow and NSLB method. Regarding the development of the GASCANS code, M. Camps Santasmasas is responsible for the turbulence modelling and coupling with other solvers. She is also involved in testing, bug fixing and other code development.

The world's an exciting place
when you know CFD.

John Shadid

Chapter 2

Computational fluid dynamics applied to urban wind flow.

Researchers, engineers and computer graphics programmers use Computational fluid dynamics (CFD) to model a wide range of flows and applications. From rapid turnaround low accuracy applications such as video games, to high precision computations of flow around vehicles and buildings. However, the size of the simulated region and accuracy of the CFD results are constrained by the available computational resources and time. Figure 2.1 shows some examples of applications and the time the user needs to get results. On one hand, high accuracy CFD simulations require the use of supercomputers and/or having to wait a long time to get results. On the other hand, interactive applications require instant results, which limits the amount of interactive cases that can be modelled accurately using currently available computational resources.

This chapter starts by reviewing current applications of urban wind CFD and the new cases that could be studied with interactive / real time wind modelling of acceptable accuracy. It then reviews the currently available CFD models applicable to model wind around bluff bodies and urban wind flow and the different ways to combine two or more models in a hybrid scheme to further increase computational efficiency. Finally, we present a summary and conclusions on what can be improved.

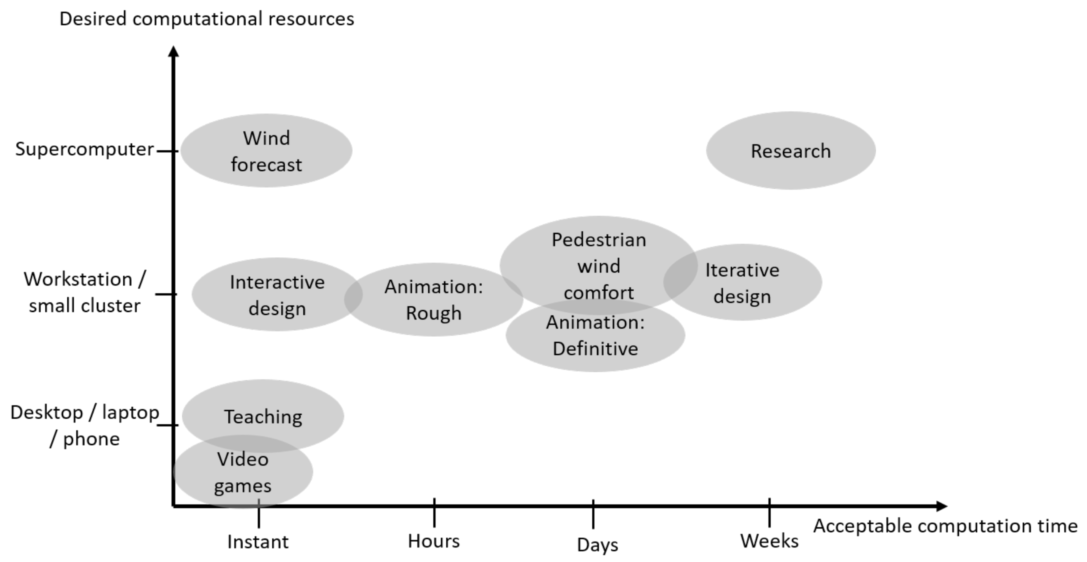


Figure 2.1: Applications of computational wind modelling classified by the time needed to obtain meaningful results depending on the available computational resources.

2.1 Applications of wind flow modelling

Buildings and new urban developments affect the wind climate of the region they are in, and in turn, are affected by the local wind climate. Strong winds can damage buildings [17]. On the other hand, the buildings affect the microclimate in their vicinity [11] and can create areas of locally strong winds that can result in a discomfort or danger to pedestrians. One example of such a case is the Bridgewater Place building in Leeds; the adverse wind conditions created by its presence causes discomfort to pedestrians and are the cause of various injuries and one death [18]. Pedestrian wind comfort studies like the one by Murphy [19] are usually a requirement prior to the construction of a building. The suitability of the building is assessed by comparing the probability of the wind speed to be above certain values with an established comfort criteria [20]. These studies are usually conducted using CFD and/or wind tunnel analysis after the building's design is finished. If the building is assessed as *unsafe*, its design has to be changed and/or wind protection structures have to be installed. This is a slow process that could be sped up if the engineers possessed a tool that allowed them to analyse the effects of the building on its surroundings at the same time as they design it. The computational model needed for this application needs to be stable, since it has to be robust against the unpredictable user input; it also needs to run at interactive time so the user sees the effects on the wind of the changes they made.

Regarding its accuracy, the great majority of the wind comfort criteria [20] are based on the Beaufort scale [20], which has a precision of order 1 m/s.

However, there are still windy locations in the already built areas that can pose a threat to pedestrians. For example, Blocken et al. [21] propose an automatic system to paliate too windy conditions in the passages under high rise buildings. The system consists of installing a set of automatic doors that open and close depending on the measured wind speed at both sides of the passage. The idea of real time pedestrian wind comfort could be extended to many more urban areas with the aid of real time CFD. Combining the data from precomputed CFD simulations, a real time CFD model and local wind measurements the users of the application could know in real time what are the potential dangerous or uncomfortable areas as they walk around. Moreover, this application could be extended to an even more local scale and be used to predict uncomfortable wind conditions inside a building or in the garden. The characteristics of the real time CFD model are very similar to those for the building design application. The main difference is that the current application requires real time or faster than real time computation speed, since the user needs a prediction of the wind characteristics in order to avoid unfavourable areas.

Another possible application at the local urban area scale is to build a wind awareness system for drones. There is a great variety of applications for drones, from filming and aerial photography to shipping goods [22]. Drones are greatly affected by the local wind conditions they are flying into, thus being able to use real-time CFD to model the wind conditions around a drone will improve the survival rate of the drones and lower the risk of the flights. The requirements for the real time CFD model for this application differ from the previous one in that drones are really sensitive to turbulence, so the application needs to predict and display volumes of unsafe flight depending on wind speed and turbulence intensity.

Chemical, biological, radiological or nuclear contaminants released by accidents or intentional attacks in a urban area can have disastrous consequences. Predicting the path of the contaminants and knowing their source is a crucial task that allows to minimize the damage caused. However, the interaction between the wind and the structures in urban areas complicates modelling the dispersion and behaviour of the contaminants. Computational Fluid Dynamics is one of the most accurate methods to model contaminant dispersion in urban areas [23] [24] but it is not usually used by the emergency response teams in the field due to its difficult configuration and consumption of time and resources. A real time CFD model for this application has to be able to model contaminant transport and it also should include algorithms to find probable sources of the contaminant by measuring contaminant concentration at different points in the affected region [25].

There are other applications of real time CFD that are not strictly modelling the wind around urban areas. Real time CFD could be used in sport applications to, for example, predict the wake of the other vehicles in a race, to predict the wind in sailing regattas, or to find favourable currents and avoid dangerous areas while doing paragliding or hang gliding among others. It could also be applied to video games as a part of a more realistic weather simulation and it could even lead to new types of games that use the wind simulation as their main mechanic.

Finally, interactive time computational fluid dynamics can be used for educational purposes, for example to demonstrate fluid dynamics concepts and flow behaviour in an interactive and easy to understand way.

2.2 Computational fluid dynamics models applied to urban wind flow

Fig. 2.2 analyses the reviewed CFD methods by the following criteria:

- Application: the method is more commonly used for engineering, computer graphics applications or both.
- Equations modelled: which fluid motion equations do the methods discretise and solve.
- Discretisation: discretisation of the domain in either a grid of cells, a group of particles or both cells and particles.
- Convection: how the convection and acceleration of the flow are modelled.
- Viscosity: how diffusion due to viscosity is represented.
- Incompressibility: how is the incompresibility enforced, if it is enforced.

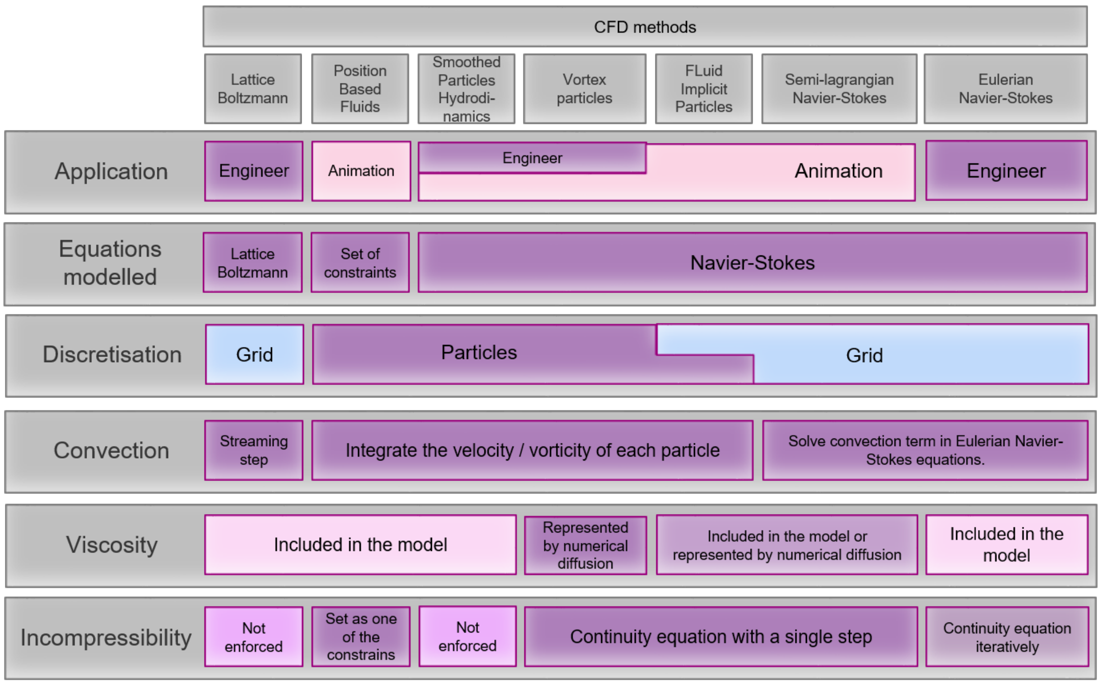


Figure 2.2: Main characteristics of CFD models that can be applied to model wind around objects either in engineering, computer animation or both. Sources: Lattice Boltzmann [1], Position Based Fluids [2], Smoothed Particles Hydrodynamics [3], Vortex particles [4], Semi-lagrangian Navier-Stokes [5], Eulerian Navier-Stokes [6].

The remaining of this section explains with more detail the different models with more attention put in the once more commonly applicable to urban wind flow modelling. We chose to classify the CFD models using the discretisation type, so then the CFD methods are classified as: Grid-based methods, particle methods and particle-grid methods.

Grid-based methods divide the simulation domain in a grid of cells and compute the fluid's discretised equations in each cell before advancing in time. Their main advantages are that all the mesh points have a fixed position so finding neighbours and enforcing incompressibility is relatively easy; the disadvantages are that you have to store the data in each cell for the whole fixed grid so they require large memory allocation, you have to mesh around obstacles which means that you have to either wrap your mesh around the object (mesh becomes complicated), or submerge the object in the mesh (the cells inside the object might still use a significant amount of memory and the flow close to the object (boundary layer) might become inaccurate). Also, discretising the fluid in cells causes the movement of the fluid to be described by a diffusion term, a temporal derivative at each cell plus a convection term that describes the fluid coming from the neighbouring cells. The convection term is non-linear and thus complicated to model. Grid methods can be divided in 3 main groups: Eulerian Navier-Stokes methods like the ones used in the engineering CFD software StarCCM+ [26] and OpenFOAM [27], semi-Lagrangian Navier-Stokes (see Stable Fluids [5]), which are widely used to model smoke in computer graphics, and lattice-Boltzmann methods (e g. LUMA [28] and Palabos [29]).

Particle methods discretise the fluid as a set of particles that move through the domain following a set of equations in Lagrangian form. The particles carry all the fluid's information and their position, velocity and acceleration is updated at each time step. One of the advantages of particle methods is that they reduce memory usage, since there is no need to store a fixed grid. Another one is that there is no convection term, which is an important source of numerical dissipation in grid methods. However, particle methods are intrinsically compressible and need extra work to enforce incompressiblity. Another disadvantage is that locating the neighbouring particles needed for some calculations, for example gradients, is not straighforward. Some examples of particle methods are Smoothed particle hydrodynamics (SPH) (e g. Müller et al. [3]), vortex particles (e g. Seller et al. [4]) and position based fluids [2].

Particle-grid methods combine some of the best features of grid and particle methods. They use particles to carry the fluid's velocity across the grid to model convection. The particles' velocity is then interpolated to an Eulerian grid in which the interaction between particles is computed and their velocity and pressure updated. In the same way as particle methods, particle-grid methods do not model convection in the Eulerian grid, thus reducing the numerical dissipation. Moreover, the lack of interaction between the particles makes them easier to manage than the particles of particle methods. However, the interpolation steps betwen the grid and the particles are not uniquely defined. This alias issue is called ringing instability and might be important depending on the spatial and temporal discretisation of the studied case. Moreover, particle-grid methods needs to store a set of particles and a fixed grid, which leads to particle-grid methods being computationally expensive. Amongst the most relevant particle-grid methods are the fluid-implicit-particle (FLIP) [13] [30] and the particle-in-cell (PIC) [31].

The next section reviews different types of grid-based CFD methods, focusing on the ones with more potential for accurate and computationally fast urban wind flow, namely Navier-Stokes and lattice Boltzmann methods. This literature review does not include an in-depth review of the grid-based methods used for animation, because, even if they are computationally efficient, they are generally not accurate enough for our application. A useful overview of grid-based CFD methods for computer graphics can be found here [32].

2.3 Grid-based methods

Grid-based methods are widely used in computational wind engineering. Some examples of their applications are: assess possible wind damage to buildings [17], calculate pedestrian wind comfort indices [20] and model contaminant dispersion [24]. This thesis focuses on the following types of grid-based methods:

- *Eulerian Navier-Stokes*: widely used in computational wind engineering.
- *Semi-Lagrangian Navier-Stokes*: widely used in computer graphics to model smoke. They can also be applied to computational wind engineering. They offer stability and, in general, a lower computational cost than Eulerian Navier-Stokes models in exchange for a loss in accuracy.
- *Lattice Boltzmann methods*: used for urban wind modelling in recent years (e g. [33] [34]). Their accuracy is comparable to the accuracy of the Eulerian Navier-Stokes methods and their algorithm is efficient when implemented in Graphic processing units (GPUs). The main appeal of GPU is their computational speed and low energy consumption.

2.3.1 Eulerian Navier-Stokes models

Eulerian Navier-Stokes grid models are commonly used in engineering [35] and are implemented in many engineering CFD software like Star CCM+ [26] and OpenFOAM [27]. *Eulerian Navier-Stokes* methods for CWE discretise and numerically solve the Eulerian form of the incompressible Navier-Stokes (eq. 2.1) and continuity equations (eq. 2.2).

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} \tag{2.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \qquad i,j = 1-3 \tag{2.2}$$

where u_i is the instantaneous velocity of the fluid, p is its pressure and ν its kinematic viscosity. The spatial terms of eq. 2.1 and eq. 2.2 can be discretised using different methods. The most commonly used discretisation technique is finite volumes (for example in Van Hooff et al. [36], Liu et al. [37] and Lien et al. [24]), but there are studies using other techniques such as finite differences [38] and finite elements [25]. However, the Eulerian Navier-Stokes and continuity equations present two numerical challenges:

- The convection term $u_j \frac{\partial u_i}{\partial x_j}$ in eq. 2.1 is non-linear.
- The pressure p is present in the momentum equation 2.1 but not in the continuity equation 2.2. If both equations are discretised and build into a linear system of equations, part of the diagonal of the coefficient matrix will be 0, which results in a singular (unresolvable) system. Thus, a separate equation for pressure is needed.

A range of methods have been developed to handle these challenges in a variety of ways. For example, the non-linearity of the convection term, the pressure equation and the values of velocity and pressure are solved iteratively until convergence is reached [1]. The first developed iterative algorithm is the Semi-Implicit Method for Pressure Linked Equations (SIMPLE). Other algorithms based on SIMPLE are the Pressure-Implicit with Splitting of Operators (PISO) and PIMPLE, a hybrid of PISO and SIMPLE. We refer the reader to Versteeg and Malalasekera [39] for detailed information on each of the algorithms.

The **main strength** of *Eulerian Navier-Stokes* methods is that they solve all the terms of the Eulerian Navier-Stokes equations until convergence is reached and the Navier-Stokes equations provide a clear physical interpretation for each macro scale process. Moreover, the most common time integration schemes are

[1]the difference between the values of pressure and velocity obtained in consecutive iterations is below a set tolerance

unconditionally stable, however this depends on the discretisation schemes used on each term of the Navier Stokes equations.

The **main weakness** of *Eulerian Navier-Stokes* methods is the complexity and iterative nature of their algorithm. In general, this iterative nature leads to significant computational cost and although Eulerian Navier-Stokes equations can be implemented in GPUs, their implementation is more complex than the one for lattice Boltzmann methods and, to some extend, semi-Lagrangian methods.

2.3.2 Semi-Lagrangian Navier-Stokes models

Semi-Lagrangian Navier-Stokes methods are widely used to model smoke in fluid animation and computer graphics [32]. In the wind engineering field, Zuo and Chen [40] use a semi-Lagrangian Navier-Stokes method to model airflow in buildings in real-time. Semi-Lagrangian Navier-Stokes methods are based on the Stable Fluids solver developed by Stam [5]. Semi-Lagrangian Navier-Stokes methods also model the Eulerian form of the Navier-Stokes equations (eq. 2.1) and the continuity equation (eq. 2.2). However, they differ from the *Eulerian Navier-Stokes* models in their treatment of the numerical challenges.

The convection term in eq. 2.1 is discretised and solved using the *method of characteristics* [5]. This approach calculates the velocity convected by the flow by back tracing where this velocity was at the previous time step along the fluid's streamlines. The streamlines are usually calculated using an implicit Euler integration scheme, which is unconditionally stable but only accurate to first order and still implicit, so it needs to be solved iteratively. Huang et al. [32] presents a review of different convection schemes which aim to increase the accuracy of the convection term discretisation while maintaining the computational efficiency and stability of Stam's original scheme.

The pressure equation is derived using the Helmholtz-Hodge decomposition [5]. Then the terms of the momentum equation with the projection operator are discretised and solved sequentially. The obtained velocity field is then used in the pressure equation, and the obtained pressure field is used to correct the velocity field so that it is divergency free. This process is carried out once each time step, the values of the velocity and pressure fields are not checked for convergence. To further increase computational efficiency, many *semi-Lagrangian* methods (e g.

example Cohen et al. [41], Molemaker et al. [42] and Selle et al. [43]) disregarded the viscosity term $\nu \frac{\partial^2 u_i}{\partial x_j^2}$ of eq. 2.1. The dissipative effect of the viscosity is accounted for indirectly via artificial numerical diffusion, the majority of which comes from the low order discretisation of the convection term. However, without a physical viscosity, the model is unable to accurately impose no slip boundary conditions. Another way to increase computational efficiency is to lower the number of iterations used to solve the pressure equation [42], which controls the computational resources assigned to each of the terms of the NS equation.

The **main strength** of *semi-Lagrangian Navier-Stokes* models is that the most common methods are unconditionally stable, however this depends on the discretisation schemes used on each term of the Navier Stokes equations. This is a sought after characteristic in interactive simulations since they can not be allowed to diverge while the user is running the program. Moreover, the *semi-Lagrangian* algorithm is simpler and less computationally expensive than the *Eulerian Navier-Stokes* algorithm because the pressure and momentum equations are not solved iteratively, the convection term is solved using the method of characteristics and the viscosity term is often omitted.

The **main weakness** of *semi-Lagrangian* methods is their loss of accuracy due to numerical diffusion. The inviscid assumption commonly employed by these models is restrictive, and in general leads to significant approximations for all but very high Reynolds number flows without boundary layers, shear layers of mixing layers. Modern semi-Lagrangian schemes reduce numerical diffusion by improving of the advection scheme and the pressure projection (e. g. Selle et al. [43], Molemaker et al. [42] and [44]).

2.3.3 Lattice Boltzmann methods

Original applications of the lattice Boltzmann methods were generally restricted to laminar flow with complex geometries (for example Chu and Tai [45], Fattahi et al. [46]). Nowadays, especially due to the advent of new computing hardware and GPU, LB methods are regularly applied to turbulent flows around bluff bodies (see Koda and Lien [16], Feng et al. [33], Lenz et al. [34]). There are also commercially available CFD software that use LB for turbulent flow: Dassault Systemes XFlow [47] which is aimed at turbulent external aerodynamics, Altair UltraFluidX [48] which works on GPU and is aimed at turbulent external aerodynamics, Simscale [49] which runs urban wind and other external aerodynamic cases using cloud resources and GPU lattice Boltzmann.

The lattice Boltzmann methods are based on the Boltzmann equations (eq. 2.3), which describe the behaviour of a fluid at a mesoscopic scale. Instead of modelling the macroscopic quantities of the gas (density, velocity, pressure...), they model the statistical distribution of the particles that form it. The particle distribution function f represents the probability of a particle to be moving with a velocity ξ_i in a location $\mathbf{x}$ at a time t. The density ρ and macroscopic velocity u_i of the fluid are obtained from the 0th and 1st moments of the Boltzmann equations as shown in eq. 2.4 and eq. 2.5. It can be shown that the moments of the lattice Boltzmann equations reproduce the Navier-Stokes and continuity equation for low Mach number flows (see section 3.2.6 for more details).

$$\frac{\partial f}{\partial t} + \xi_i \frac{\partial f}{\partial x_i} = \Omega(f) \tag{2.3}$$

$$\rho(\mathbf{x}, t) = \int f(\mathbf{x}, \xi, t) d^3\xi \tag{2.4}$$

$$\mathbf{u}(\mathbf{x}, t)\rho(\mathbf{x}, t) = \int \xi f(\mathbf{x}, \xi, t) d^3\xi \tag{2.5}$$

The Boltzmann equation is an advection equation in Eulerian form. The first term in the LHS of eq. 2.3 models the time evolution of the distribution function f, the second models convection and the RHS term $\Omega(f)$ is the collision operator and models the collisions between the particles of the fluid. The collision operator $\Omega(f)$ must ensure that collision conserves mass, momentum and energy and that it evolves the distribution functions f towards the equilibrium distribution functions f^{eq}.

The Boltzmann equations (eq. 2.3) have no analytical solution, thus the domain has to be discretised and solved numerically. The particle distribution function $f(\mathbf{x}; \xi; t)$ depends on 7 variables: x_i, ξ_i and t. Discretising and solving eq. 2.3 for all the variables is computationally expensive and would require large scale supercomputers. However, we only need the moments of the Boltzmann equation to be correct to reproduce the Navier-Stokes equations, thus much of the underlying physics is not relevant to the values of the macroscopic variables u_i and ρ. The particle velocity ξ_i can then be discretised into a small number of velocities without affecting the validity of the macroscopic quantities.

The velocity discretisation of the Boltzmann equation (eq. 2.3) is done using Hermite polynomials and the Hermite-Gauss quadrature, see Krüger [1]. The resulting velocity-discretised Boltzmann equation and corresponding macroscopic density and velocity are:

$$\frac{\partial f_\alpha}{\partial t} + c_{\alpha i}\frac{\partial f_\alpha}{\partial x_i} = \Omega(f_\alpha) \tag{2.6}$$

$$\rho = \sum_\alpha f_\alpha \tag{2.7}$$

$$u_i \rho = \sum_\alpha f_\alpha c_{\alpha i} \tag{2.8}$$

where α takes values from 0 to the number of segments used to discretise the particle velocity ξ_i and $c_{\alpha i}$ is the discretised particle velocity. The resulting number of discretised particle velocities to exactly conserve the macroscopic quantities (i e. the first 3 moments) of eq. 2.3 is 27. However, some of them can be simplified and the number of discretised velocities reduced to 19, 15 or 13, which is the minimum number of velocities needed to reproduce the Navier-Stokes equations [1]. The higher velocity spaces are more stable but also more memory consuming. The most widely used velocity set for 3D fluid simulations is the D3Q19 (Table 2.1), since it strikes a good balance between stability and computational resources. However, some lattice Boltzmann models use other velocity sets to work: for example the cummulant [50] needs D3Q27 .

Name	**Velocities c_α**	**Number**	**Weights**
D3Q27	$(0,0,0)$	1	8/27
	$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	6	2/27
	$(\pm1,\pm1,0),(\pm1,0,\pm1),(0,\pm1,\pm1)$	12	1/54
	$(\pm1,\pm1,\pm1)$	8	1/216
D3Q19	$(0,0,0)$	1	1/3
	$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	6	1/18
	$(\pm1,\pm1,0),(\pm1,0,\pm1),(0,\pm1,\pm1)$	12	1/36

Table 2.1: Most common discrete velocity sets for three dimensional problems.

Eq. 2.6 remains continuous in time and space and as for the Navier-Stokes equations, there is no analytical solution. Thus, it needs to be discretised and numerically solved. The velocity discretised Boltzmann equation can be discretised in space using finite differences [51], finite volumes [52] or even spectral difference [53] methods following similar procedures as in the Eulerian and semi-Lagrangian Navier-Stokes models. For example [51] uses second order central-difference and upwind schemes to discretise the convection term and a second-order Runge-Kutta scheme for time discretisation; their collision operator is local, so it does not need to be discretised. This discretisation yields as many equations and unknowns f_α as discrete particle velocity in the set; unlike with the Navier-Stokes methods, this equations can be directly solved with linear equation solvers without having to deal with the continuity equation and the pressure coupling. Finite volume/differences/elements discretisation allows the use of non-uniform mesh and mesh refinement, thus potentially lowering the computational cost of the simulation; however, solving linear systems of equations is computationally expensive and the approximation of the spatial derivatives may include more than only nearest neighbours, which complicates GPU implementation.

An alternative to finite volume/differences/elements discretisation and the most commonly used time and space discretisation of the velocity discrete Boltzmann equations is to use the method of characteristics [1]; which converts the LHS of 2.6 into a total derivative with respect to the trajectory of the flow in space ϕ

$$\frac{df_\alpha}{d\phi} = \Omega_\alpha(\mathbf{x}(\phi), t(\phi)) \tag{2.9}$$

integrating both sides along the trajectory ϕ, from $\phi = 0$ to $\phi = \Delta t$, and using $t(\phi = 0) = t_0$ and $x_i(\phi = 0) = x_{i0}$ as initial conditions yields

$$f_\alpha(\mathbf{x_0} + \mathbf{c}_\alpha \Delta t, t_0 + \Delta t) - f_\alpha(\mathbf{x_0}, t_0) = \int_0^{\Delta t} \Omega_\alpha(\mathbf{x}_0 + \mathbf{c}_\alpha \phi, t_0 + \phi) d\phi \tag{2.10}$$

and numerically solving the RHS of eq. 2.10 yields the lattice Boltzmann equations (eq. 2.11):

$$f_\alpha(\mathbf{x_0} + \Delta x, t_0 + \Delta t) - f_\alpha(\mathbf{x_0}, t_0) = \Delta t \Omega_\alpha \tag{2.11}$$

$$\rho = \sum_\alpha f_\alpha \tag{2.12}$$

$$\rho u_i = \sum_\alpha f_\alpha c_{\alpha i} \tag{2.13}$$

where Δt is the size of the time step and $\Delta x = \boldsymbol{c}_\alpha \Delta t$ is the cell size. The magnitude of the discrete velocities $\boldsymbol{c}_\alpha$ is chosen so that the particles in a cell are displaced to the neighbouring cell (i.e. $c_{\alpha i} = \Delta x / \Delta t$). Note that unlike semi-Lagrangian Navier-Stokes models, the particle distribution functions f_α in one cell move exactly to the neighbouring cell in the direction of the discretised particle velocity $c_{i\alpha}$ in one time step. Thus, the method of characteristics in LB is exact, requiring no interpolation and introducing no artificial numerical diffusion.

Eq. 2.11 is solved in two steps: the *streaming step* solves the convection term by copying the distribution functions from one cell to its nearest neighbour in the direction of each discretised particle velocity $_\alpha$; then the *collision* step applies the collision operator to the post-streaming f^*_α to obtain the f_α for the next time step. The method of characteristics plus explicit Euler discretisation yields a simple equation that does not require to solve a system of equations or to be solved iteratively in any way. It is also local (i e. a function of neighbouring cells only), which makes it suitable to be implemented in massive parallel architectures (like GPU) without major modifications. However, the assumption that the discrete velocities will move the particles one cell in one time step makes it difficult to implement mesh refinement and unstructured meshes, even prism meshes. Moreover, the explicit Euler integration scheme is conditionaly stable, valid only for certain combinations of Δt, Δx. Unlike semi-Lagrangian Navier-Stokes models, eq. 2.11 solves one variable for each discretised particle velocity in cell, thus an implicit Euler scheme for lattice Boltzmann is around 6 times more computationally expensive than the same scheme for semi-Lagrangian Navier-Stokes models. Horstmann et al. [54] presents an hybrid segregated finite volume - streaming collision discretisation with the objective to use mesh refinement in the finite volume region and take advantage of the computational efficiency of the stream-collide algorithm elsewhere.

The **collision operator** Ω_α must ensure that particles conserves mass, momentum and energy during particle-particle interactions and that it evolves the particle distribution functions f_a towards the equilibrium distribution functions f_α^{eq} (see Krüger [1] for more details). The simplest collision operator that complies with both conditions is the Bhatnagaar Gross and Krook (BGK) [55] (eq. 2.14)

$$\Omega_\alpha = -\frac{f_\alpha - f_\alpha^{eq}}{\tau} \tag{2.14}$$

where the relaxation time τ controls the rate at which the particle distribution functions relax towards equilibrium and is related to the viscosity of the fluid ν via

$$\nu = \frac{1}{3}\left(\tau - \frac{1}{2}\right)\frac{\delta x^2}{\delta t} \tag{2.15}$$

where δx is the cell size in dimensionless/physical units and δt is the time step in dimensionless/physical units (see section 3.2.3). As shown in eq. 2.15, lattice Boltzmann with BGK collision becomes unstable for values of τ below $\frac{1}{2}$, which would make the flow viscosity negative. Moreover, $u_i\delta t/\delta x$ needs to be maintained below the lattice speed of sound (section 3.2.6). High Reynolds number flows are characterised by low fluid viscosity and high velocity, thus an accurate and stable BGK lattice Boltzmann simulation for high Reynolds flow would require a usually prohibitively small cell size δx to comply with the quadratic dependence between the fluid viscosity ν and the cell size δx combined with the restriction on $u_i\delta t/\delta x$. τ is present only in the collision term, so modifying the collision term is a way to search for more stable and accurate lattice Boltzmann formulation. The advanced collision operators have more degrees of freedom than BGK, so the extra freedom can be used to improve stability and accuracy. Some examples of advanced collision operators are the multi relaxation time (MRT) [1], the cumulant [50] and the central moments based [56], also referred to as cascaded. See [57] for a more detailed explanation of the different collision models.

The **main strength** of the stream - collide BGK lattice Boltzmann model lies on its simplicity. The interactions between nodes during streaming are linear (one cell only affects its nearest neighbours), and all the heavy computations included in the collision step are local. This simplicity and locality makes the lattice Boltzmann method very efficient when implemented in massively parallel architectures like GPUs. Moreover, the streaming step models advection without numerical diffusion. Other factors that reduce the computational cost of the LB method is that it does not involve solving a Poisson equation for pressure and that it incorporates the effects of viscosity without solving a linear system of equations.

The **main weakness** of the stream - collide BGK lattice Boltzmann is that is only conditionally stable. The explicit time integration and spurious terms in the discretisation restrict the stable combinations of temporal and spatial discretisation steps. Moreover, the assumption that the discrete particle velocities reach from one cell to exactly the boundary of its neighbours combined with that the viscosity depends also on the time and space discretisation makes mesh refinement cumbersome to apply. Another disadvantage is that the LB method is memory intensive, since it has to store the values of all the discretised particle distribution functions plus velocity and density at each cell.

2.4 Summary of findings

The CFD methods reviewed in this thesis are classified in 3 groups: grid-based methods, particle methods and particle-grid methods. From them, grid-based methods are the most used in wind engineering and in general for modelling single phase single component fluids. We sub-divide grid-based methods in three groups: *Eulerian Navier-Stokes, semi-Lagrangian Navier-Stokes* and *lattice Boltzmann* methods.

Eulerian Navier-Stokes methods are widely used in wind engineering. They numerically solve the Eulerian form of the Navier-Stokes and continuity equations using iterative algorithms and system of linear equations solvers. They are the most accurate of the grid methods; however, their algorithms are complex and computationally expensive. In order to increase stability and reduce computer time the computer animation industry developed the *semi-Lagrangian Navier-Stokes* methods [5]. *semi-Lagrangian Navier-Stokes* methods also model the Eulerian form of the Navier-Stokes and continuity equations but their algorithm is aimed at reducing its complexity and number of iterations. They are the most stable of the grid methods and their algorithms are simpler than the *Eulerian Navier-Stokes* methods algorithms; however, they are the less accurate. Finally, *Lattice Boltzmann* methods are being used more and more in wind engineering. They numerically solve the velocity discretised Boltzmann equations. Lattice Boltzmann results can match the accuracy of Eulerian Navier-Stokes [16] while the most commonly employed version fo the lattice Boltzmann algorithm is simple, local and efficient for GPU acceleration. However, the relationship between the fluid viscosity and the spatial and temporal discretisation and their explicit time integration scheme contributes to their instability; implementing mesh refinement is complicated and they are memory intensive.

Reducing the computational resources used by CFD methods will potentially provide more accurate and useful information and expand the use of CFD in applications like emergency response teams, teaching and computer graphics. The low memory consumption of Eulerian Navier-Stokes methods could be combined with the computational speed provided by lattice Boltzmann methods to further this goal. The next chapter provides more details about the NS, LB methods and turbulence modelling used in the current work; followed by a study on coupling methods.

It's still magic, even if you know how it's done.

Terry Pratchett, A hat full of sky

Chapter 3

Theory

3.1 Eulerian Navier Stokes method

The most commonly employed equations to model fluid flow are the incompressible Eulerian Navier-Stokes equations. The main body forces that are considered when modelling wind flow are the pressure of the wind, the gravity, Coriolis force and buoyancy due to the differences in temperature. In this thesis, we decided to neglect the effect of Coriolis and buoyancy forces for simplicity. Finally, the effect of gravity is irrelevant in the absence of thermal effects, since one of the main effect of gravity is to counterbalance the buoyancy force.

This section describes the Eulerian Navier-Stokes solver which when coupled with the lattice Boltzmann solver described in section 3.2, form the hybrid segregated Navier-Stokes lattice Boltzmann solver (see chapter 4).

3.1.1 Equations solved and discretisation

The Eulerian Navier-Stokes model used in the present work solves the incompressible Navier-Stokes equations with no body forces together with the continuity equation:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \sigma_{ij}}{\partial x_j} \tag{3.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \qquad i,j = 1-3 \tag{3.2}$$

where u_i is the velocity of the fluid, where σ_{ij} is the Cauchy stress tensor. The Cauchy stress tensor can be divided in the normal stresses, which are the isotropic part of σ_{ij} and the deviatoric stress tensor, which is the anisotropic part of σ_{ij}. Then the second term of the RHS of eq. 3.1 can be expressed as:

$$\frac{\partial \sigma_{ij}}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau^d_{ij}}{\partial x_j} \tag{3.3}$$

where the normal stresses are the flow pressure p. For an incompressible flow the deviatoric stress tensor only contains the shear stress and can be expressed as:

$$\tau^d_{ij} = \nu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.4}$$

where the constant ν is the viscosity of the fluid.

The spatial terms in Eq. 3.1 and Eq. 3.2 are discretised with a finite volume method, which divides the domain in 3-dimensional cells or control volumes and solves the integral form Eq. 3.1 and eq. 3.2 over the control volume. The simulation is advanced in time using an implicit Euler scheme. A detailed description of the finite volume discretisation and time stepping can be found in Ferziger and Milovan [6]. This method can be applied to unstructured meshes, which poses an advantage over the constant cell size structured meshes of the common form of the lattice Boltzmann method (section 3.2).

The non-linearility in the convection term $u_j \frac{\partial u_i}{\partial x_j}$ is approximated by freezing u_j to the value from the previous time step and solving only for u_i. The pressure equation is obtained using the Helmholtz-Hodge decomposition and the pressure-velocity coupling is solved using the pressure implicit splitting (PISO) algorithm [39]. PISO algorithm needs a priory unknown number of iterations at each time step to obtain accurate pressure and velocity values, which makes it more computationally expensive and computer time consuming than the stream-collide lattice Boltzmann algorithm described in section 3.2.

3.1.2 Initial and boundary conditions

This work implements Dirichlet boundary conditions for velocity and zero gradient Neumann boundary conditions for pressure to impose a 0 velocity on walls (no slip boundary condition), or a set velocity at inlet faces.

The intended outlet boundary conditions are zero gradient Neumann boundary conditions for both pressure and velocity, since they are the most similar to the extrapolated lattice Boltzmann boundary condition (see section 3.2.5). However, the Navier-Stokes solver used (i.e `pisoFoam` [27]) corrects mass flow imbalances by adjusting the pressure at a Dirichlet pressure boundary. If no boundary has fixed pressure, the mass flow cannot be corrected and the simulation stops. This mass correction can destabilize the Navier-Stokes lattice Boltzmann coupled solver and has to be taken into account (see chapter 4 for more details).

Finally, periodic boundary conditions are used in boundaries where the flow presents a repeating pattern. See [6] for a detailed explanation of boundary conditions in a finite volume discretisation.

3.2 Lattice Boltzmann method

The lattice Boltzmann method is one of the two solvers coupled in this work which form the hybrid segregated Navier-Stokes lattice Boltzmann solver. This section describes in more detail the numerical features implemented for the lattice Boltzmann solver used and its relationship to the Navier-Stokes equations.

3.2.1 Lattice Boltzmann model

This thesis implements a D3Q19 BGK collision standard lattice Boltzmann:

$$f_\alpha(\mathbf{x_0}+\Delta x, t_0+\Delta t) - f_\alpha(\mathbf{x_0}, t_0) = \Delta t \frac{f_\alpha^{eq}(\mathbf{x_0}, t_0) - f_\alpha(\mathbf{x_0}, t_0)}{\tau} \qquad \alpha = 0:18 \tag{3.5}$$

$$\rho(\mathbf{x_0}, t_0) = \sum_\alpha f_\alpha(\mathbf{x_0}, t_0) \tag{3.6}$$

$$p(\mathbf{x_0}, t_0) = c_s^2 \rho(\mathbf{x_0}, t_0) \tag{3.7}$$

$$\rho(\mathbf{x_0}, t_0) u_i(\mathbf{x_0}, t_0) = \sum_\alpha f_\alpha(\mathbf{x_0}, t_0) c_{\alpha i} \qquad i = 1:3 \tag{3.8}$$

$$\nu = \frac{1}{3}\left(\tau - \frac{1}{2}\right)\frac{\Delta x^2}{\Delta t} \tag{3.9}$$

where Δt is the time step, Δx is the spacial step in x, y and z, τ is the relaxation time at which the discretised particle distribution functions f_α relax to their equilibrium values f_α^{eq}, ν is the viscosity of the fluid, ρ is the density field, u_i is the macroscopic velocity field, p is the pressure field, $c_s = \frac{\Delta x}{\sqrt{3}\Delta t}$ is the lattice speed of sound and $c_{\alpha i}$ are the discretised particle velocities:

$$c = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 0 \end{bmatrix} \frac{\Delta x}{\Delta t} \tag{3.10}$$

The order of the discretised particle velocities is not universal and must be defined specifically for each solver. Eq. 3.10 and Fig. 3.1 show the order used in this work, including its software implementation (annex B).

3.2.2 Equilibrium distribution function

The collisions between fluid particles tend to homogenise the particle velocities ξ_i around the macroscopic velocity u_i. So, in the absence of external forcing the particle distribution functions will reach an equilibrium state f^{eq}. The derivation of the equilibrium distribution assumes that the particle velocities ξ_i in the equilibrium state are isotropic around the macroscopic velocity u_i, the components

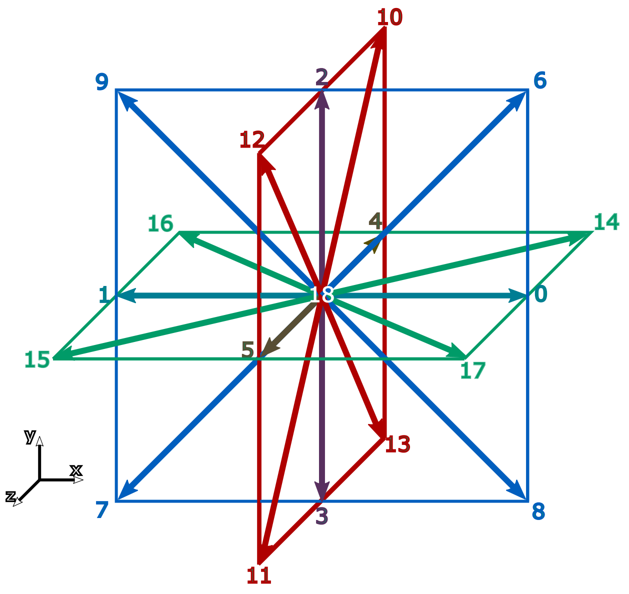


Figure 3.1: Discretised particle velocities in a 3DQ19 cell. The numbers correspond to the columns in eq. 3.10 and the vectors are coloured by the plane they rest on.

of the fluctuating velocity $v_i = \xi_i - u_i$ are independent variables and the collisions between particles conserve mass, momentum and energy. Equation 3.11 presents the resulting form of the equilibrium distribution, also referred to as Maxwell-Boltzmann distribution.

$$f^{eq}(\rho, |\mathbf{v}|, T) = \rho \left(\frac{1}{2\pi RT} \right)^{3/2} e^{-|\mathbf{v}|^2/(2RT)} \tag{3.11}$$

where R is the specific gas constant and T is the gas temperature. The fluctuating velocity $v_i = \xi_i - u_i$ is the particle velocity relative to the macroscopic velocity u_i so that the mean of the particle velocity ξ_i is u_i and the mean of the relative or fluctuating velocity v_i is 0. Discretising the particle velocity from v_i to $c_{\alpha i}$, using 3DQ19 velocity model and assuming isothermal fluid [1] yields the discretised equilibrium particle distribution function used in this work:

$$f^{eq}_{\alpha} = w_{\alpha}\rho \left(1 + \frac{1}{c_s^2}(c_{\alpha i}u_i) + \frac{1}{2c_s^4}(c_{\alpha i}u_i)(c_{\alpha i}u_i) - \frac{1}{2c_s^2}(u_i u_i) \right) \tag{3.12}$$

where $c_s = \frac{1}{\sqrt{3}}\frac{\Delta x}{\Delta t}$ is the speed of sound and the weights w_α are:

$$w_\alpha = \begin{cases} 1/18 & \alpha = 0:5 \\ 1/36 & \alpha = 6:17 \\ 1/3 & \alpha = 18 \end{cases} \tag{3.13}$$

3.2.3 Lattice units

In lattice Boltzmann it is useful to consider that the macroscopic quantities ρ, u_i and ν are in lattice units. The reference length in lattice units is 1 cell length and the reference time is 1 time step. For example, a velocity value in lattice units indicates how many cells are covered in a time step. The time step Δt and spatial step Δx in eq. 3.5 and eq. 3.10 are always 1 in lattice units. This assumption simplifies the lattice Boltzmann equations by avoiding to carry the relationship between the cell size and the time step size during streaming and collision. Results in lattice units can be converted to dimensionless units by applying the definition of lattice units:

$$t_d = t\delta t \tag{3.14}$$

$$x_d = x\delta x \tag{3.15}$$

where t_d and x_d are a certain temporal and length value in dimensionless units, t and x are the same value in lattice units, δt is a time step in dimensionless units and δx is the cell size in dimensionless units. The macroscopic variables in dimensionless units can then be obtained by dimensional analysis:

$$u_{di} = u_i \frac{\delta x}{\delta t} \tag{3.16}$$

$$\nu_d = \nu \frac{\delta x^2}{\delta t} \tag{3.17}$$

Finally, physical units can be obtained by using the reference scales, so that:

$$t_p = t_d T \qquad u_{pi} = u_{di}\frac{L}{T} \tag{3.18}$$

$$x_p = x_d L \qquad \nu_p = \nu_d \frac{L^2}{T} \tag{3.19}$$

where the index p denotes physical units and T and L are a length in metres and a time in seconds representative of the modelled case. Since T and L are representative of the modelled case they are also used to define the Reynolds number:

$$Re = \frac{L^2}{T\nu_p} = \frac{1}{\nu_d} \tag{3.20}$$

where ν_p is the kinematic viscosity of the fluid in $m^2 s^{-1}$.

For the remaining of this thesis all the lattice Boltzmann equations are presented in lattice units, i e. $\Delta x = 1$ cell length, $\Delta t = 1$ time step.

3.2.4 Algorithm

The form of the lattice Boltzmann equation in eq. 3.5 is solved in two steps: stream and collide. The **stream step** (LHS of eq. 3.5) models the advection of the fluid particles from the previous cell to the current one in the direction of the discretized particle velocities $c_{\alpha i}$. There are two variations of streaming: pull and push, both presenting the same accuracy. Push streaming advects the particle distribution functions from the current cell to the neighbouring cells; while pull streaming advects the particle distribution functions from the neighbouring cells to the current cell. The code used in this thesis uses pull streaming due to its increased efficiency for GPU implementation [58].

Fig. 3.2 and eq. 3.21 illustrate pull streaming for a 2DQ9 velocity discretisation, which is equivalent to a two-dimensional plane of the 3DQ19 velocity discretisation in Fig. 3.1. All the simulations and case studies in this thesis are three-dimensional and use a D3Q19 velocity discretisation, however, for the remainder of this chapter all the explanatory figures only show one plane of the 3DQ19 cells to aid the clarity of the representation.

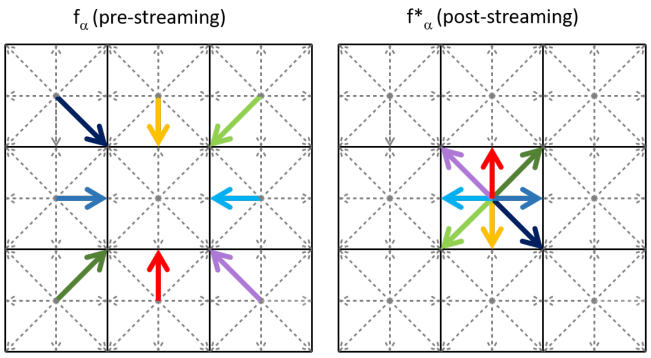


Figure 3.2: Scheme of the pull streaming algorithm for one plane of a 3DQ19 cell.

$$f^*_\alpha(\mathbf{x}, t + \Delta t) = f_\alpha(\boldsymbol{x} + \Delta t \boldsymbol{c}_{\tilde{\alpha}}, t) \tag{3.21}$$

where f^*_α is the post-streaming particle distribution function in the α direction, Δt is the size of the time step (equal to 1 in lattice units), and $\tilde{\alpha}$ denotes the velocity vector on the opposite direction of α.

Once the pull step is completed, the **collide step** models the collisions between the particles in the cell by solving the right hand side of equation 3.5.

Algorithm 1 shows a LBM time step using a pull streaming algorithm. This algorithm only loops through the grid cells once every time step: it executes streaming then collision for one cell before moving to the next cell. It is important to note that $f_\alpha(\boldsymbol{x}, t + \Delta t)$ and $f_\alpha(\boldsymbol{x}, t)$ need to be stored in different arrays, so that $f^*_\alpha(\boldsymbol{x}, t + \Delta t)$ does not overwrite the values in $f_\alpha(\boldsymbol{x}, t)$ during streaming.

3.2.5 Initial and Boundary conditions

Initial and boundary conditions are usually imposed on the macroscopic quantities, velocity and pressure. Unlike Navier-Stokes based CFD methods, Lattice Boltzmann solves the particle distribution functions f_α, which are related to the

Algorithm 1 Algorithm for a time step a pull streaming BGK collision lattice Boltzmann method.

```
1:  for all cells, l do
2:
3:      Step 1: Streaming
4:      for all particle velocities, $\alpha$ do
5:          $f^*_\alpha(\boldsymbol{x}_l, t+\Delta t) = f_\alpha(\boldsymbol{x}_l + \Delta t \boldsymbol{c}_{\tilde{\alpha}}, t)$;
6:      end for
7:
8:      Step 2: Macroscopic quantities
9:      Initialise macroscopic density and velocity to 0
10:     for all particle velocities,$\alpha$ do
11:         density: $\rho(\boldsymbol{x}_l, t+\Delta t) \mathrel{+}= f^*_\alpha(\boldsymbol{x}_l, t+\Delta t)$;
12:         for $i = 0; i < 3; i++$ do
13:             $u_i(\boldsymbol{x}_l, t+\Delta t) \mathrel{+}= c_{\alpha i} f^*_\alpha(\boldsymbol{x}_l, t+\Delta t)$
14:         end for
15:         velocity: $u_i(\boldsymbol{x}_l, t+\Delta t) \mathrel{/}= \rho(\boldsymbol{x}_l, t+\Delta t)$
16:     end for
17:
18:     Step 3: Collision
19:     for all particle velocities, $\alpha$ do
20:         $f_\alpha(\boldsymbol{x}_l, t+\Delta t) = f^*_\alpha(\boldsymbol{x}_l, t+\Delta t) - \frac{1}{\tau}(f^*_\alpha(\boldsymbol{x}_l, t+\Delta t) - f^{eq}_\alpha(\boldsymbol{u}(\boldsymbol{x}_l, t+\Delta t), \rho(\boldsymbol{x}_l, t+\Delta t))$
21:     end for
22:
23: end for
```

macroscopic quantities via eq. 3.5. Velocity and pressure can be calculated given all the f_α. However, if only the macroscopic quantities are known, the system is underdetermined and additional closure equations are needed. Initialisation procedures for lattice Boltzmann particle distributions from a set velocity and pressure are an open topic of research (see for example Mei et al. [59] and Chikatamar et al. [60])).

For a boundary cell, the particle distribution functions leaving the domain are known because they have been streamed from its neighbouring cells and the particle distribution functions entering the domain are unknown because their particle distribution functions should be streamed from outside the domain (Fig. 3.3). Boundary conditions in lattice Boltzmann use a combination of the known particle distribution functions and the macroscopic quantities to deduce the values of the unknown particle distribution functions.

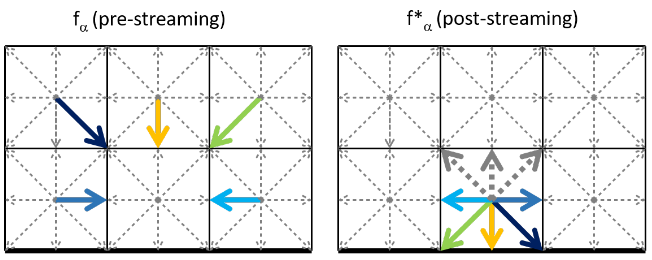


Figure 3.3: Streaming in a boundary cell without applying boundary conditions. After streaming the cell is missing the particle distribution functions from outside the domain (thick dashed arrows).The black thick grid line indicates the boundary.

Periodic

Periodic boundary conditions are used to model flows with repeating patterns. In the LBM this is achieved by streaming the particle distribution functions that would exit the domain into the incoming particle distribution functions for the opposite boundary (Fig. 3.4).

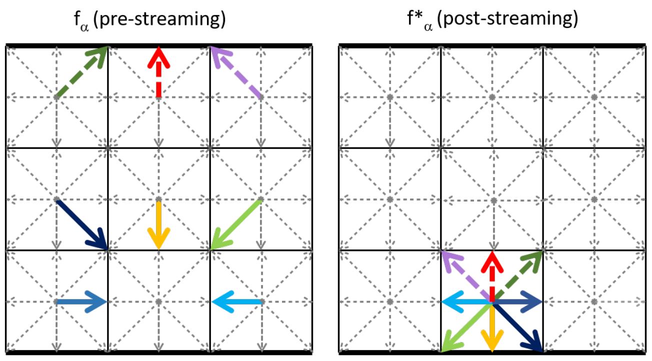


Figure 3.4: Streaming in a boundary cell with periodic boundary conditions. The thick dashed arrows indicate the particle distributions where the boundary condition is applied and the thick solid lines indicate the boundaries.

The periodic boundary condition can be implemented into the streaming step by wrapping the cell indices around the number of cells as shown in algorithm 2.

Algorithm 2 Lattice Boltzmann periodic boundary condition implemented as part of the pull streaming step. d is the coordinate (x, y or z), $id_s[d]$ is the cell that provides f_α, $id[d]$ is the current cell, $c[d, \alpha]$ are the discretised particle velocities (eq. 3.10) and $N[d]$ is the number of cells.

```
1: for all cells, l do
2:     Step 1: Streaming
3:     for all particle velocities, $\alpha$ do
4:         Step 1.1: Get the indices of the cells to pull from
5:         for all dimensions, $d$ do
6:             $id_s[d] = id[d] - c[d, \alpha]$;
7:             if $id_s[d] == -1$ then
8:                 $id_s[d] = 0$
9:             else if $id_s[d] == N[d]$ then
10:                $id_s[d] = N[d] - 1$
11:            end if
12:        end for
13:        Step 1.2: Pull stream as normal
14:        $f^*_\alpha(id, t + \Delta t) = f_\alpha(id_s, t)$
15:    end for
16:    Step 2: Macroscopic quantities
17:    Step 3: Collision
18: end for
```

Extrapolated

Extrapolated boundary conditions allow the flow to go through the boundary, and their effect on the macroscopic quantities is close to a zero gradient boundary condition.

Extrapolated boundary conditions set the missing particle distribution functions to the values of the particle distribution functions on the neighbour cell in the stream direction (Fig. 3.5).

Forced equilibrium

The forced equilibrium condition is used at boundaries to impose a macroscopic velocity u_{i0} and density ρ_0. Forced equilibrium boundary condition sets the population distribution functions of each boundary cell $f_\alpha(\mathbf{x}, t+\Delta t)$ to the equilibrium values that yield the imposed u_{i0} and density ρ_0 (eq. 3.22). If density information is not available, it is common to set $\rho_0 = 1$.

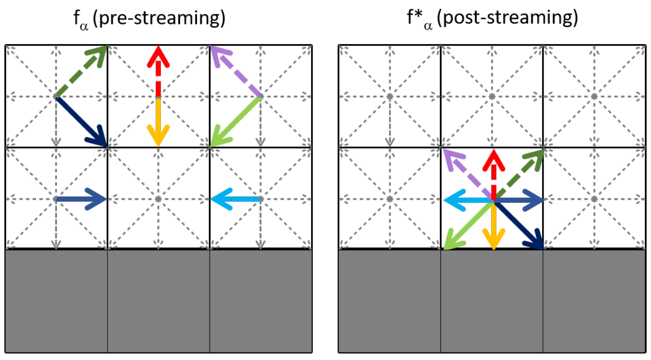


Figure 3.5: Streaming in a boundary cell with extrapolated boundary conditions. The thick dashed arrows indicate the particle distributions where the boundary condition is applied (right) and where they come from (left). The thick solid lines indicate the boundary and the grey areas are dummy cells marked as extrapolated boundary.

$$f_\alpha(\mathbf{x}, t + \Delta t) = f_\alpha^{eq}(\rho_0(\mathbf{x}, t), \mathbf{u_0}(\mathbf{x}, t)) \tag{3.22}$$

Forced equilibrium is stable and the simplest way of imposing a Dirichlet boundary condition in lattice Boltzmann. However, it is the less accurate since it only takes into account the equilibrium components of $\mathbf{u_0}$ and ρ_0, thus the non-equilibrium information is lost. The effect of viscosity is also lost (see section 3.2.6) so it is not able to impose no slip boundary conditions. Besides, the boundary cells do not receive information from its neighbouring cells and thus the mass of the system is not conserved.

Regularised

Regularised boundary conditions by Latt et al. [61] combine macroscopic information with the particle distribution functions streamed from inside the domain to imposed a macroscopic velocity u_{i0} and density ρ_0. Unlike forced equilibrium, regularised boundaries allow for either the velocity normal to the boundary u_n or the density ρ_0 to be determined from the known velocity components and /or known density and the known particle distribution functions. Once the missing component (either u_n or ρ_0) is calculated, all the particle distributions in the regularised boundary are reconstructed using the non-equilibrium distribution functions. Since regularised boundary conditions use non-equilibrium information they are more accurate than forced equilibrium and able to model no-slip boundaries. However, regularised boundary conditions are less stable than forced equilibrium boundaries.

The first step to implement the regularised boundary condition is to calculate the missing macroscopic component on the boundary cells from the known distribution functions and remaining macroscopic variables. Density ρ can be divided in the following three components: ρ_0, which is in the same plane as the boundary, ρ_+, which is aligned with the boundary's normal vector pointing inside the domain and ρ_- which is aligned with the boundary's normal vector pointing outside the domain:

$$\rho = \rho_0 + \rho_+ + \rho_- \tag{3.23}$$

where ρ_0 is the sum of the particle distribution functions in directions planar to the boundary

$$\rho_0 = \sum_{\alpha | c_{\alpha n}=0} f_\alpha, \tag{3.24}$$

ρ_+ is the sum of the particle distribution functions streaming from outside the domain and n is the direction normal to the boundary

$$\rho_+ = \sum_{\alpha | c_{\alpha n}=1} f_\alpha, \tag{3.25}$$

and ρ_- is the sum of the particle distribution functions streaming from inside the domain

$$\rho_- = \sum_{\alpha | c_{\alpha n} = -1} f_\alpha, \tag{3.26}$$

Substituting eq. 3.23 into the macroscopic velocity equation (eq. 3.8) yields:

$$\rho u_n = \rho_+ - \rho_- \tag{3.27}$$

The particle distributions streaming from outside the domain are unknown, but can be eliminated by substituting eq. 3.23 into eq. 3.27. Then, the unknown macroscopic variable (density ρ or normal velocity u_n) can be obtained by:

$$\rho = \frac{\rho_0 + 2\rho_-}{1 - u_n} \tag{3.28}$$

Fig 3.6 shows the streaming and regularised boundary condition applied to a corner of a 2D domain. In 3D, corners and edges present special cases; some of the particle distribution functions that form ρ_0 and ρ_- (dashed coloured arrows in Fig 3.6) are unknown, so 3.28 can not be directly used to obtain the macroscopic density ρ. To solve this issue, the code used in this work first decides to which boundary the cell is part of (bottom boundary in Fig. 3.6), then streams all particle distribution functions implementing periodic boundaries (see algorithm 2), and then applies regularised boundary conditions.

The next step is to use the previously calculated macroscopic quantities u_i and ρ to obtain intermediate values for the distribution functions f_α streaming from outside of the domain. The particle distribution functions f_α can be approximated using the Chapman Enskog expansion

$$f_\alpha = f_\alpha^{(0)} + \epsilon f_\alpha^{(1)} + O(\epsilon^2) \tag{3.29}$$

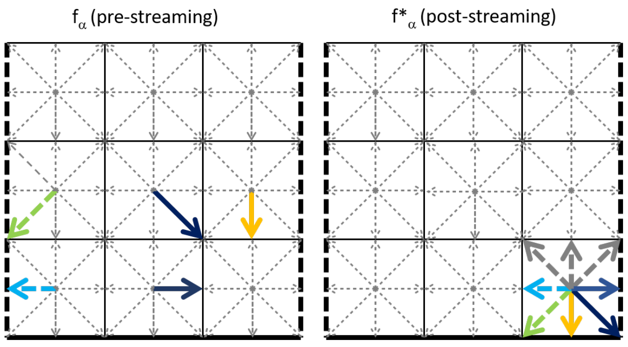


Figure 3.6: Streaming in a corner/edge boundary cell with regularised boundary conditions. In this example the cell pertains to the lower boundary (thick continuous line), the other boundaries are thick dashed lines. The coloured thick dashed arrows indicate the particle distributions part of ρ_0 and ρ_- streamed using periodic boundary conditions. The thick dashed grey arrows are the particle distributions part of ρ_+.

where $\epsilon << 1$ is the smallness parameter (Knudsen number), $f_\alpha^{(0)} = f_\alpha^{eq}$ and $f_\alpha^{(1)} \approx f_\alpha^{neq}$. The non-equilibrium distribution function f_α^{neq} is:

$$f_\alpha^{neq} = f_\alpha - f_\alpha^{eq} \tag{3.30}$$

f_α for the particle distribution functions streamed from outside the domain is unknown, but it can be calculated via the non-equilibrium bounceback assumption [62]:

$$f_\alpha^{neq} = f_{\tilde{\alpha}}^{neq} \tag{3.31}$$

where $\tilde{\alpha}$ denotes the particle velocity direction opposite α. Thus the intermediate values of the unknown f_α can be calculated by substituting eq. 3.31 into eq. 3.30 and then into eq. 3.29 as

$$f_\alpha = f_\alpha^{eq} + f_{\tilde{\alpha}} - f_{\tilde{\alpha}}^{eq} \tag{3.32}$$

Finally, all the particle distribution functions are recalculated as

$$f_\alpha = f_\alpha^{eq} + \frac{w_\alpha}{2c_s^4} Q_{\alpha ij} \Pi_{ij}^{(1)} \tag{3.33}$$

where

$$Q_{\alpha ij} = c_{\alpha i} c_{\alpha j} - c_s^2 \delta_{ij} \tag{3.34}$$

and the first order components of the stress tensor $\Pi_{ij}^{(1)}$ are:

$$\Pi_{ij}^{(1)} = \sum_\alpha c_{\alpha i} c_{\alpha j} f_\alpha^{(1)} \qquad i, j = 0 : 2 \tag{3.35}$$

Half-way bounce back

Half-way bounce back boundary conditions are used at solid walls to impose no slip condition. Half-way bounce back modifies the streaming step so that the particle distributions bouncing away from the solid wall are set to the pre-streaming particle distribution functions directed towards the wall; each particle distribution is assigned to the one in the opposite direction (Fig. 3.7).

The bounce back boundary condition sets the solid wall (i.e the point where the macroscopic velocity is zero) at the mid plane between the boundary cells and the dummy bounce back cells (thick line in Fig. 3.7). This position halves the distance between the centre of the boundary cell and the wall, thus also halving $y+$ in turbulent simulations and potentially increasing their accuracy. Moreover, the half-way bounce back boundary condition conserves mass and it is easy to implement in solid surfaces that conform with the mesh. Curved surfaces and flat surfaces not aligned with the mesh are represented by a "staircase" pattern of step size equal to the cell size, degrading the accuracy of the boundary condition. Bounce-back can be modified to treat curved walls [63], though the modifications

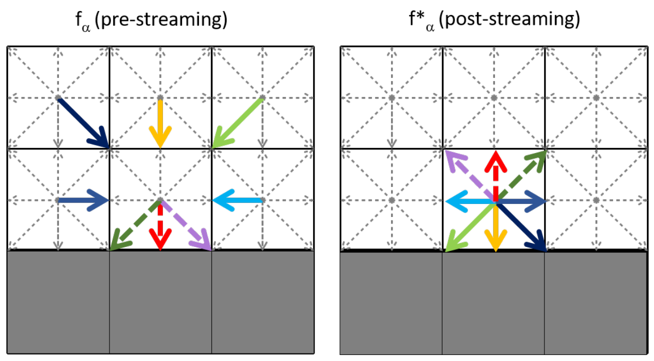


Figure 3.7: Streaming in a boundary cell with half-way bounce back boundary condition. The thick dashed arrows indicate the particle distributions where the boundary condition is applied (right) and where they come from (left). The thick solid lines indicate the boundary and the greyed areas are dummy cells marked as bounce-back boundary.

increase the complexity of the algorithm. Another disadvantage of bounce back is that, when used with BGK collision, the location of the no-slip boundary depends on the lattice Boltzmann viscosity, which depends on the discretisation (eq. 3.17). This can lead to different flow behaviour for flows with the same Reynolds number, which is not physical.

Initial conditions

All initial conditions used in the present work are forced equilibrium; in which the macroscopic velocity and density are set and the particle distributions are initialised to the equilibrium particle distributions.

3.2.6 Macroscopic conservation equations

This section shows how the Boltzmann and lattice Boltzmann equations relate to the continuity and Navier-Stokes equations (see section 3.1); with emphasis on the conditions for the stability restrictions in the values of lattice velocity, and thus in the values of the time and spatial discretisation.

The Boltzmann equation is:

$$\frac{\partial f}{\partial t} + \xi_i \frac{\partial f}{\partial x_i} = \Omega(f) \tag{3.36}$$

where ξ_i is the particle velocity. The first three moments of the particle distribution function f are:

$$\int f d^3\xi = \rho \tag{3.37}$$
$$\int \xi_i f d^3\xi = \rho u_i \tag{3.38}$$
$$\int \xi_i \xi_j f d^3\xi = \Pi_{ij} = u_i u_j \rho - \sigma_{ij} \tag{3.39}$$

where Π_{ij} is the momentum flux density tensor and σ_{ij} is the viscous stress tensor.

The first step to obtain the **continuity equation** is to integrate the Boltzmann equation (eq. 3.36) over velocity space:

$$\frac{\partial}{\partial t} \int f d^3_\xi + \frac{\partial}{\partial x_i} \int \xi_i f d^3\xi = \int \Omega(f) d^3\xi \tag{3.40}$$

The first and second terms of eq. 3.40 can be resolved using the moments in eq. 3.37 and eq. 3.38; the integral of the collision operator is zero, since the collision operator conserves mass. The resulting equation is the continuity equation for compressible fluids

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0 \tag{3.41}$$

We obtain the **momentum conservation equations** by calculating the first moment of the Boltzmann equation:

$$\frac{\partial}{\partial t}\int \xi_i f d^3\xi + \frac{\partial}{\partial x_j}\int \xi_i \xi_j f d^3\xi = \int \xi_i \Omega(f) d^3\xi \tag{3.42}$$

Following the same procedure used for the continuity equation, the first and second terms of equation 3.42 can be solved using the first order and the second order moments of the particle distribution function f (eq. 3.38 and 3.39). The RHS term is 0, since the collision operator conserves momentum. Thus, the resulting equation is the Cauchy momentum equation:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(u_i u_j)}{\partial x_j} - \frac{\partial \sigma_{ij}}{\partial x_j} = 0 \tag{3.43}$$

where

$$\begin{aligned}\sigma_{ij} &= -\int v_i v_j f d^3\xi && (3.44)\\ &= p\delta_{ij} - (1-\delta_{ij})\int v_i v_j f d^3\xi && (3.45)\end{aligned}$$

with the fluctuating particle velocity $v_i = \xi_i - u_i$; where u_i is the mean velocity of the particles (i e. the macroscopic velocity of the fluid).

Unlike the continuity equation (eq. 3.41), the form we obtained of the Cauchy momentum equation is not closed, since it lacks an explicit equation for f. The simplest method to approximate f is to assume that the particles in the fluid are always at equilibrium and thus $f = f^{eq}$. The equilibrium distribution function does not depend on the fluctuating particle velocity v_i, so it can be taken out of the integral:

$$\sigma_{ij} = p\delta_{ij} - f^{eq}(1-\delta_{ij})\int v_i v_j d^3\xi \tag{3.46}$$

The integral in the last term of equation 3.46 is $\overline{v_i v_j}$ (i.e. the time-averaged value of $v_i v_j$). One of the assumptions used to build the equilibrium distribution function is that, in equilibrium, the components of the fluctuating velocity v_i are independent variables [1]. Then it follows that: $\overline{v_i v_j} = \bar{v}_i \bar{v}_j$. Remember also that the time-averaged value of any of the components of v_i is zero by definition. So if we approximate $f \approx f^{eq}$ we recover the Euler equations:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(\rho u_i u_j)}{\partial x_j} = -\frac{\partial p}{\partial x_j} \tag{3.47}$$

Note that the Euler equations are inviscid, which suggests that the effect of viscosity is related to the non-equilibrium part of the distribution function $f^{neq} = f - f^{eq}$.

The non-equilibrium particle distribution functions can be approximated using the the Chapman-Enskog expansion. Its main idea is to express f as a perturbation expansion around f^{eq} as:

$$f_\alpha = f^{(0)} + \epsilon f^{(1)} + \epsilon^2 f^{(2)} + ... \tag{3.48}$$

were $f^{(0)} = f^{eq}$ and the smallness parameter ϵ^n indicates terms of order Kn^n, where Kn is the Knudsen number [1]. Each order in Kn forms a semi-independent equation by itself, so that the higher order terms might be seen as corrections to the lower order terms. Applying a Taylor expansion to the lattice Boltzmann equation and expanding its derivatives using the Chapman-Enskog expansion [1] we obtain:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(u_i u_j)}{\partial x_j} = -\frac{\partial(\rho c_s^2)}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\rho c_s^2 \left(\tau - \frac{\Delta t}{2}\right)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \mathcal{O}(u^3)\right) \tag{3.49}$$

[1]The Knudsen number is defined as the ratio of the molecular mean free path length of the fluid and a physical length scale.

Eq. 3.49 is the Navier-Stokes equations with the fluid viscosity $\eta = \rho c_s^2 \left(\tau - \frac{\Delta t}{2}\right)$, pressure $p = \rho c_s^2$ and an extra $\mathcal{O}(u^3)$ term, which is the error due to the lack of a $\mathcal{O}(u^3)$ term in the discretised equilibrium function f_α^{eq}. This error is only negligible if $u^2 \ll c_s^2$, which means that the lattice Boltzmann equation only reproduces the Navier-Stokes equations for velocities much lower than the speed of sound c_s. It can also be interpreted that the macroscopic momentum equation from the lattice Boltzmann equation represents the compressible Navier-Stokes equations for weakly compressible flows. Another observation from the previous equation is that $\frac{\tau}{\Delta t} \leq \frac{1}{2}$ yields a negative viscosity η, and thus the solved flow becomes unstable. This condition comes from the explicit Euler time integration scheme.

3.3 Turbulence modelling

Most flows occurring in nature are turbulent and local urban wind is no exception. To have an idea of what a turbulent flow looks like we only need to look at the clouds, at a smoke plume or at a river surface (fig. 3.8). Tennekes and Lumley [64] describe turbulence as [2] :

- irregular: we need statistical tools to describe turbulent flows;
- diffusive: turbulent velocity fluctuation will spread with time to cover more space;
- characteristic of high Reynodls numbers: where the lack of mathematical tools to give general solutions to the Navier-Stokes equations hinders the study of turbulence;
- rotational: characterised by high levels of vorticity fluctuations;
- three-dimensional: since the vorticity fluctuations can only be maintained by three dimensional flows [65];
- dissipative: viscosity dissipates turbulence into heat, thus turbulence decays without a source of energy.

[2]See [64] for a detailed description of turbulence

- multi scale in time and space: Large scale turbulent structures, also referred to as eddies, dissipate kinetic energy by forming smaller eddies, until the smallest eddies are dissipated into heat by viscosity.

.

Figure 3.8: Turbulent flow at the surface of the river Mersey, Manchester.

The statistical tools used to describe turbulent flows in the present work are the mean velocity U_i, the fluctuating velocity u'_i, the Reynolds stress tensor $< u'_i u'_j >$ and the turbulent length scale σ_i.

The mean and fluctuating component of the velocity are obtained from the instantaneous velocity using the Reynolds decomposition. It states that the instantaneous velocity of a turbulent flow u_i can be decomposed in a time averaged mean componentU_i and a fluctuating component u'_i

$$U_i =< u_i(\mathbf{x}) >= \sum_t u_i(\mathbf{x}, t) \tag{3.50}$$

$$u'_i(\mathbf{x}, t) = u_i(\mathbf{x}, t) - U_i(\mathbf{x}) \tag{3.51}$$

with $< u'_i(\mathbf{x}) >= \sum_t u'_i(\mathbf{x}, t) = 0$.

The Reynolds stress tensor R_{ij} calculates the correlation between the three components of the fluctuating velocity

$$R_{ij}(\mathbf{x}) =< u'_i u'_j >= \sum_t u'_i(\mathbf{x},t) u'_j(\mathbf{x},t) \tag{3.52}$$

The turbulent length scale σ_i represents the size of the turbulent eddies. It can be calculated using the Reynolds stress tensor as

$$\sigma_i = \frac{R_{ii}^{3/2}}{\epsilon} \tag{3.53}$$

where ϵ is the turbulent kinetic energy dissipation rate.

3.3.1 Direct numerical simulation

The most direct approach to compute turbulent flow is using the Navier-Stokes equations (section 3.1) or the lattice Boltzmann equations (section 3.2) to solve for all the spatial and temporal scales of the instantaneous velocity of the fluid u_i. The resulting flow field is equivalent to the flow field obtained in a wind tunnel experiment or observed in nature. Thus the name of this approach is direct numerical simulation (DNS).

Turbulence acts in a range of lengths and time scales and, to obtain an accurate representation of the unsteady flow, all of them need to be captured in the temporal and spacial discretisation. A model that represents all time and length scales is computationally expensive. For example, in the case of the spatial discretisation, the size of the domain must be a few times the size of the largest turbulent structure, with cell sizes of the scale related to viscous dissipation, the Kolmogorov scale. The description of the flow obtained from DNS is very detailed and thus, post-processing and extracting the information needed for engineering applications is resource intensive. Another issue of DNS is that the initial and boundary conditions are difficult to generate, since they have to contain the same amount of information as the modelled flow.

The arguments in the paragraph above motivated the development of methods that resolve only the large scales of motion and approximate the rest with a turbulence model. This chapter describes the large eddy simulation (LES) turbulence model, which resolves the turbulent velocity field above a certain scale and models the effect of the velocities at lower scales. Another family of turbulence models are the Reynolds averaged Navier-Stokes equations (RANS), which only solve for the time averaged velocity and model the effect of the turbulence on the main flow. For a more extensive review of the turbulence models available in the literature, the reader is referred to Pope [65].

3.3.2 Large eddy simulation (LES)

The largest length scales present in a turbulent flow, i e. those which scale with the size of the geometry or shear flow that generates them, provide the majority of the transport of the conserved quantities. The effect of the small length scales on the transport of the conserved quantities is less significant [6], thus it is adequate to resolve the large length scales and incorporate the small scales only through their effect on the large scales.

The first step towards the LES form of the fluid flow equations is to generate a velocity field $\bar{u}_i$, for Eulerian Navier-Stokes, or a particle distribution function $\bar{f}_\alpha$, for lattice Boltzmann, that only contains the large scales of turbulence. The LES solved quantity is obtained by spatially filtering the instantaneous velocity or particle distribution function for lengths scales larger than a filter width Δ. In one dimension:

$$\bar{w}_i(x) = \int G(x, x')w_i(x')dx' \tag{3.54}$$

where the over bar indicates spatially filtering operation, w_i is the instantaneous velocity u_i for the Eulerian Navier-Stokes equations and the instantaneous particle distribution function f_α for the lattice Boltzmann equations. The filter kernel $G(x, x')$ can be for example a box filter (i.e. a local average) or a cut-off filter (i.e. use the Fourier transform to eliminate all wavelengths below the filter width Δ).

Navier-Stokes implementation

The incompressible Eulerian Navier-Stokes equations and the continuity equation for the filtered velocity $\bar{u}_i$ and filtered pressure $\bar{p}$ are:

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j}(\overline{u_i u_j}) = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_i}\left[\nu\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right)\right] \tag{3.55}$$

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0 \tag{3.56}$$

The correlation $\overline{u_i u_j}$ is not easily computed. We define the *subgrid scale Reynolds stress tensor* τ_{ij} as the missing term needed to compute $\overline{u_i u_j}$:

$$\overline{u_i u_j} = \bar{u}_i \bar{u}_j + \tau_{ij} \tag{3.57}$$

The subgrid scale Reynolds stress τ_{ij} accounts for the contribution of the unresolved small scales to the resolved large scales. For an extensive explanation and review of LES turbulence models the reader is referred to Sagaut [66]. Sagaut [66] describes two modelling approaches to approximate τ_{ij}: *structural modelling* and *functional modelling*. Structural models approximate the form of the tensor τ_{ij}. For functional models, τ_{ij} represents the effect of the subgrid scales on the filtered velocity $\bar{u}_i$ and thus provide only an energy balance, not the form of τ_{ij} itself.

The simplest sub grid scale model is the Smagorinsky model [67], which is a functional model. It assumes that the effects of the subgrid scale term on the flow are increased transport and dissipation of momentum. The viscosity of the fluid ν causes these same effects, so it is reasonable to assume that the subgrid scale term could take the same form as the viscosity term. For an incompressible flow:

$$\tau_{ij} - \frac{\delta_{ij}}{3}\tau_{kk} = \nu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \tag{3.58}$$

where ν_t is the sub grid scale eddy viscosity and $\frac{\delta_{ij}}{3}\tau_{kk}$ is the isotropic part of the Reynolds stress, which can be modelled as part of the solved pressure $\bar{p}$ [66]. The original Smagorinsky model defines the sub grid eddy viscosity as

$$\nu_t = C_s \Delta^2 |\bar{S}| \tag{3.59}$$

where C_s is a parameter of the model, Δ is the length of the filter and $|\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$. The strain rate of the filtered velocity $\bar{S}_{ij}$ is:

$$\bar{S}_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \tag{3.60}$$

It is worth to note that the parameter C_s might depend on different factors, both physical and numerical, and might take different values in different flows. Standard values of C_s are between 0.01 and 0.0144 and the standard value for plane channel flows is $C_s = 0.01$ [66]. To reduce the variability of the C_s value, Germano et al. [68] developed a dynamic Smagorinsky model in which the value of C_s changed to adapt to the flow's characteristics. However, this thesis implements the original Smagorinsky model due to its simplicity.

Lattice Boltzmann implementation

The lattice Boltzmann implementation of the LES Smagorinsky model follows the steps described by Hou et al. [69] and Koda and Lien [16]. The lattice Boltzmann equations for the filtered particle distribution function $\bar{f}_\alpha$ are:

$$\bar{f}_\alpha(\mathbf{x_0} + \Delta x, t_0 + \Delta t) - \bar{f}_\alpha(\mathbf{x_0}, t_0) = \Delta t \bar{\Omega}_\alpha(\mathbf{x_0}, t_0) \qquad \alpha = 0:18 \tag{3.61}$$

where $\bar{\Omega}_\alpha$ is the filtered collision operator. Hou et al. [69] assumes that the filtered collision operator $\bar{\Omega}_\alpha$ relaxes the filtered particle distribution functions $\bar{f}_\alpha$ to a filtered equilibrium distribution $\bar{f}^{eq}_\alpha$ of the same form as the original unfiltered equilibrium distribution f^{eq}_α but function of the filtered macoscopic density $\bar{\rho}$ and velocity $\bar{u}_i$. It also assumes that the influence of the filtering process on the collision operator only introduces an eddy viscosity ν_t, which modifies the relaxation time. Thus $\bar{\Omega}_\alpha$ for a BGK collision operator becomes:

$$\bar{\Omega}_\alpha = \frac{f^{eq}_\alpha(\bar{\rho}, \bar{\mathbf{u}}) - \bar{f}_\alpha}{\bar{\tau}} \tag{3.62}$$

$$\bar{\rho} = \sum_\alpha \bar{f}_\alpha \tag{3.63}$$

$$\overline{\rho u_i} = \sum_\alpha \bar{f}_\alpha c_{\alpha i} \qquad i = 1:3 \tag{3.64}$$

$$\bar{\tau} = 3(\nu + \nu_t) + \frac{1}{2} \tag{3.65}$$

where $\bar{\tau}$ is the modified relaxation time, $c_{\alpha i}$ is the discretised particle velocity, ν is the viscosity of the fluid and

$$\nu_t = C_s \Delta^2 |\bar{S}| \tag{3.66}$$

is the Smagorinsky eddy viscosity; Δ^2 is the filter width, which is usually equal to the cell size. The value of the Smagorinsky constant C_s is typically between 0.01 and 0.04 (see for example Hou et al. [69] Wang et al. [70] and Yu et al. [71]). $|\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$ is the intensity of the strain rate of the filtered velocity

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \tag{3.67}$$

The velocity gradients in $\bar{S}_{ij}$ can be discretised using finite differences [70] [72]. However, the discretisation incorporates non-local operations to the lattice Boltzmann collision algorithm, thus reducing its computational efficiency on GPU implementations [58]. How et al. [69] proposes the following method to calculate $\bar{S}_{ij}$ involving only local operations. The first order components of the filtered velocity stress tensor $\bar{\Pi}_{ij}^{(1)}$ can be approximated as

$$\bar{\Pi}_{ij}^{(1)} = \sum_{\alpha} c_{\alpha_i} c_{\alpha j} \bar{f}_{\alpha}^{neq} \tag{3.68}$$

where $\bar{f}_{\alpha}^{neq} = \bar{f}_{\alpha} - \bar{f}_{\alpha}^{eq}$ is the non-equilibrium distribution function. Then the second variance Q of the tensor $\bar{\Pi}_{ij}^{(1)}$ is

$$Q = \bar{\Pi}_{ij}^{(1)} \bar{\Pi}_{ij}^{(1)} \tag{3.69}$$

The first order components of the filtered velocity stress tensor are related to the filtered velocity strain rate tensor via [69] [1].

$$\bar{\Pi}_{ij}^{(1)} = -\frac{2\bar{\rho}\bar{\tau}}{3} \bar{S}_{ij} \tag{3.70}$$

Combining eq. 3.70, eq. 3.69, eq. 3.66 and eq. 3.65 we obtain

$$Q^{1/2} = (\tau + 3C_s\Delta^2|\bar{S}|)\frac{2\bar{\rho}}{3\sqrt{2}}|\bar{S}| \tag{3.71}$$

where $\tau = 3\nu + \frac{1}{2}$ is the original relaxation time. Eq. 3.71 is a second order equation for $|\bar{S}|$. Solving it and substituting in eq. 3.65 we obtain:

$$\bar{\tau} = \frac{1}{2\bar{\rho}}\left(\sqrt{\bar{\rho}^2\tau^2 + 18\sqrt{2}\bar{\rho}C_s\Delta^2 Q^{1/2}} - \tau\bar{\rho}\right) + \tau \tag{3.72}$$

All the operations in 3.72 are local, so the implementation of the LES Smagorinsky using eq. 3.72 does not significantly increase the computational cost of the lattice Boltzmann method in GPU.

3.3.3 Inlet boundary condition: Synthetic eddy method

The fluid velocity at the inlet of a turbulent flow simulation needs to contain all the scales of motion for the simulation to accurately represent turbulent flow. Unresolved scales decrease the accuracy of the simulation and increase the distance needed by the flow to develop from the inlet conditions to realistic turbulent flow. Tabor and Baba-Ahmadi [73] review different inlet boundary conditions for LES simulations. They divide the inlet boundary conditions in two groups:

- *Synthesised turbulence methods*: add artificially generated fluctuating velocity to the mean velocity using mean flow data.

- *Precursor simulation*: the inlet velocity is generated from a precursor periodic DNS or LES.

One of the reviewed synthesised turbulence methods is the synthetic eddy method (SEM). SEM uses the mean flow statistics to generate synthetic eddies that are convected across the inlet. The inlet velocity field is obtained by superposing the eddies velocity to the mean velocity. SEM presents reduced computational cost compared to methods that require a precursor LES or DNS simulation and reduced development lengths compared to simpler synthesised turbulent methods [73]. The present work uses the SEM by Jarrin et al. [74] improved by Skillen et al. [7]. The improved SEM model presents the following advantages:

- Shorter development length than previous SEM models [7].

- Decreased computational cost, since it generates less eddies.

Additionally, our research group has access to a C++ implementation of the code; which made it easy to adapt to lattice Boltzmann and introduce in the lattice Boltzmann code used in this thesis.

Navier-Stokes implementation

The synthetic eddy method proposed by Jarrin et al. [74] and improved by Skillen et al. [7] generates an artificial fluctuating velocity field u_i' across the inlet face using as input the time averaged inlet velocity U_i, the Reynolds stresses R_{ij}, and turbulent length scale σ_i.

The first step is to generate a fixed number of synthetic eddies centred at random positions on the inlet plane. The eddies are then moved by a random distance perpendicular to the inlet face in the range of the eddy size (Fig. 3.9). This way, all the eddies start at random positions both across and along the inlet while all of them intersect the inlet and thus contribute to the fluctuating velocity.

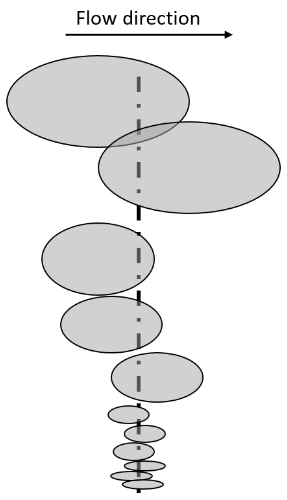


Figure 3.9: Sketch of the side view of the inlet plane (dash-dot line) showing the initial positioning of the SEM eddies. Figure reproduced from Skillen et al. [7].

The next step is to calculate the preliminary fluctuating velocity field $u_i^*(\mathbf{x}, t)$ by adding the contribution of each eddy normalised by the running average of the eddy concentration (eq. 3.73).

$$u_i^*(\mathbf{x},t) = \frac{\sum_{e=1}^{N} \epsilon_i f(\mathbf{x}-\mathbf{x_e},\sigma_\mathbf{e})}{\sqrt{\left\langle \sum_{e=1}^{N} g^2(\mathbf{x}-\mathbf{x_e},\sigma_\mathbf{e}) \right\rangle^{AVG}}} \tag{3.73}$$

where ϵ_i is an integer representing the sign of the eddy and is assigned a ± 1 value at random, $\mathbf{x}$ is the current inlet cell, $\mathbf{x_e}$ is the position of the centre of the current eddy, $\sigma_\mathbf{e}$ is the length scale of the current eddy, N is the number of eddies and f is a weight function based on the distance from the centre of the current eddy. Skillen et al. [7] use a truncated Gaussian function with extents defined by the length scale of the current eddy $\sigma_\mathbf{e}$. Skillen et al. [7] contains more information on the weight function g and the implementation of the running average of the eddy concentration ($\langle\rangle^{AVG}$).

The next step is to correct the preliminary fluctuating velocity u_i^* so that it has zero mean and the second order statistics dictated by the prescribed Reynolds stresses R_{ij}. To do so, the fluctuating velocity u_i^* is multiplied by the Cholesky decomposition of the Reynolds stress tensor L_{ij}

$$L = \begin{bmatrix} \sqrt{R_{11}} & 0 & 0 \\ R_{21}/L_{11} & \sqrt{R_{22}-L_{21}^2} & 0 \\ R_{31}/L_{11} & (R_{32}-L_{21}L_{31})/L_{22} & \sqrt{R_{33}-L_{31}^2-L_{32}^2} \end{bmatrix} \tag{3.74}$$

and the instantaneous velocity u_i is obtained via:

$$u_i(\mathbf{x},t) = L_{ij}u_j^*(\mathbf{x},t) + U_i \tag{3.75}$$

where U_i is the prescribed mean velocity.

At each time step, the eddies are advected through the inlet by the bulk velocity of the prescribed mean flow. When an eddy no longer intersects the inlet, it is regenerated to a new random starting position, which is recomputed until the eddy lands in a region with low eddy density.

Lattice Boltzmann implementation

The instantaneous inlet velocity for lattice Boltzmann simulations is generated using Skillen et al. [7] SEM following the same formulation as the Navier-Stokes implementation. The obtained instantaneous velocity is then converted to lattice Boltzmann units and applied to the inlet via a regularised boundary condition (see. section 3.2.5). However, this direct implementation had two main shortcomings:

- One time step of the lattice Boltzmann simulation run in the GPU takes less time than 1 time step of the SEM simulation in the CPU; which makes the SEM a bottleneck.

- The lattice Boltzmann method implemented in this work allows only for meshes with constant size cubic cells; thus the calculation of the contribution of each eddy to the preliminary fluctuating velocity (eq. 3.73) can be optimised to work only with constant regular grids.

The lattice Boltzmann code used in this work, GASCANS (see apendix B), addresses the short-commings above using the SEM Courant number and an optimised eddy overlap function.

The *SEM courant number* Co_{SEM} is an input set by the user used to calculate the time step for the SEM δt_{SEM}.

$$\delta t_{SEM} = Co_{SEM} * \frac{\delta x}{U_b} \tag{3.76}$$

where δx is the inlet cell size and U_b is the bulk velocity perpendicular to the inlet. δt_{SEM} is then corrected so that $\delta t_{SEM} = r * \delta t_{LBM}$, where r is an integer value. If $\delta t_{SEM} < \delta t_{LB}$ the SEM time step is set to the lattice Boltzmann time step. The SEM instantaneous velocity is only updated at δt_{SEM}, thus saving computational cost and reducing the time spend in calculating SEM.

The algorithm to calculate the preliminary velocity field (eq. 3.73) is the most computationally expensive part of the SEM implementation, since it involves a nested search loop. For each cell in the inlet, the code goes through all the eddies to add up the contribution of the eddies whose centre is less than σ_e away from the current cell. The order of this algorithm is of $N_{cells} * N_{eddies}$. For example, if the inlet has 200×360 cells and SEM generates 1800 eddies, one time step of this algorithm involves 129.6 million steps. This algorithm does not need any a priori knowledge of the inlet grid structure and it will work with any kind of structured or unstructured mesh. However, our lattice Boltzmann implementation works only with constant size cubic cells, which allows for a more optimised algorithm. The position and size of the eddies are now trivial to convert to cell indices $\mathbf{index}_e$, since $\mathbf{index}_e = \mathbf{x_e}/\delta x$, where δx is the cell size. The outer loop now goes through all the eddies and adds the contribution of each eddy to the cells it occupies. The order of the new algorithm is approximately $N_{eddies} * 4\sigma_e^2$. Using the same mesh as in the previous example and assuming a mean $\sigma_e = 20$ cells, one time step of the new algorithm involves 5.76 million steps, which is two orders of magnitude lower than the original algorithm.

Algorithm 3 shows the Skillen et al. [7] SEM method implemented in lattice Boltzmann, including the SEM courant number and mesh optimisations.

The SEM Courant number and the optimisation of the preliminary velocity field algorithm reduce the computational cost of SEM to a lower value than the lattice Boltzmann loop. Thus SEM is no longer a bottle neck in the lattice Boltzmann simulation.

Algorithm 3 Algorithm for a time step of the SEM boundary condition implemented in the lattice Boltzmann method used in this thesis.

```
1:  Step 0: Update SEM mean flow if needed
2:
3:  State 1: Check if SEM velocity needs updating
4:  if $\delta t_{SEM} \% \delta t_{LBM} != 0$ then
5:      Exit.
6:  end if
7:
8:  Step 2: Advect the eddies
9:  for all eddies, i do
10:     $\mathbf{x}_e[i] += U_b \delta t_{SEM}$
11:     if time > residence time[i] then
12:         Regenerate eddy i
13:     end if
14: end for
15:
16: Step 3: Update instantaneous velocity
17: for all eddies,e do
18:     Get the mesh indices of the eddy bounding box i0, in, j0, jn, k0, kn
19:     for i=i0; i < in; i++ do
20:         for j=j0; j < jn; j++ do
21:             for k=k0; k < kn; k++ do
22:                 if eddy e is in cell, because eddies are not prisms then
23:                     Add eddy to running average (eq. 3.73 denominator)
24:                     Add eddy contribution to $\mathbf{u}^*[cell]$ (eq. 3.73 numerator)
25:                 end if
26:             end for
27:         end for
28:     end for
29: end for
30: for all cells, c do
31:     Update $\mathbf{u}^*[c]$
32:     Calculate instantaneous velocity $\mathbf{u}[c]$ (eq. 3.76)
33: end for
34:
35: Step 4: Correct inlet mass flow
36:
37: Step 5: Advance time
38: time += $\delta t_{SEM}$
```

All models are wrong, but some are usefull.

George E. P. Box

Chapter 4

Navier-Stokes lattice Boltzmann model

Urban wind flow is a multi-scale phenomena, covering from time scales of milliseconds and meters for the smallest resolved turbulent scales to hours and kilometres for the changes in mean velocity at a city wide scale. The time scale can be extended to years if the averaged velocity and gusts are needed for climate studies as for example pedestrian wind comfort [19]. In that case, the results are also affected by the climatic weather conditions of the region, with an scale of thousands of kilometers. Modelling urban wind flow using a single model that is able to resolve all the scales is computationally taxing and often unrealisable.

The Multiscale Modelling and Simulation Framework (MMSF) [75] divides a multi scale simulation in multiple submodels connected via *smart conduits*. Each submodel focuses in a sub-set of scales of the multi-scale phenomena and works autonomously, not aware of the scales of the submodels it is connected to. This separation has two main advantages: (1) a submodel can be substituted by a better version of the code or algorithm if needed without interfearing with the others, (2) each submodel can work in a different hardware architecture (e g. GPU and CPU) and be optimised for it without interfearing with the other submodels. The *smart conduits* are scale aware and transfer and map the data between submodels.

The description of a multiscale model using the MMSF is divided in 2 parts:

- *Theoretical framework*: describes the

  - **modelling**: division of the multiscale phenomena in submodels and which scales each submodel focuses in
  - **architecture**: execution loop of each submodel and the coupling templates, i e. at which points of the execution loop each submodel transfers and receives information to which other submodel.

- *Computational framework*: describes the
  - **implementation** of each submodel and of the smart conduits using either a *monolithic approach*, a *coupling framework* or a *coupling library.*
  - **execution** of the multiscale model and the hardware architecture it uses.

This chapter presents the description of the multiscale urban wind model developed in this thesis and divided following the MMSF. The NSLB method presented in this thesis couples two submodels, a NS submodel for the large scales and a LB submodel for the small scales.

4.1 Theoretical framework

The first step is to identify the relevant scales and processes (submodels) involved in the phenomena we want to study and display them in a scale separation map (SSM).

The second step is to express the time loop of each submodel using a submodel execution loop (SEL). SEL is a generic, abstract execution temporal loop that includes the temporal loop of each submodel and the comunication between submodels at specific steps of the temporal loop.

4.1.1 Modelling

We consider two cases:

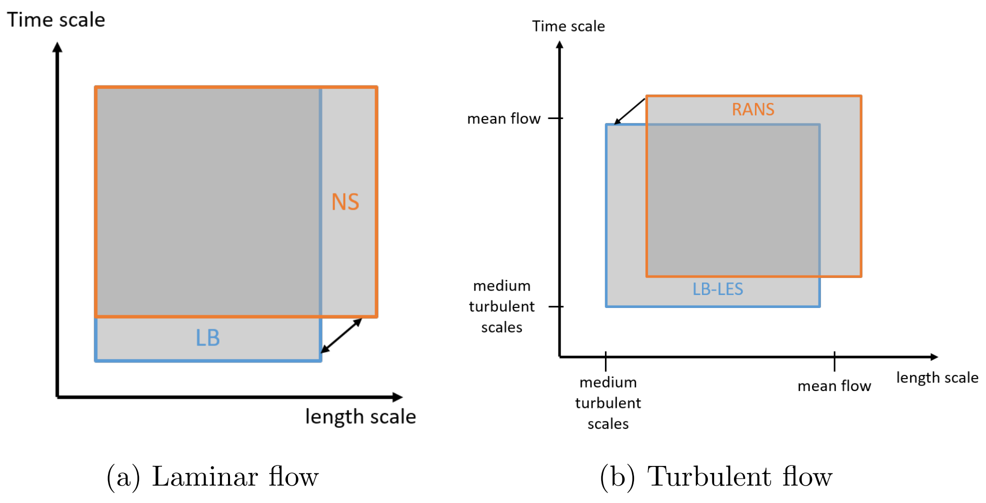


Figure 4.1: SSM for the multiscale NSLB method. (a) Laminar flow case; (b) turbulent flow case. The arrows indicate information transfer between the NS and LB submodels

- Laminar / low Reynolds number flow: in which both the NS and LB submodels run without turbulence modelling.

- Turbulent flow: in which the NS submodel implements a RANS turbulence model to resolve the mean flow and model turbulence and the LB submodel implements an LES Smagorinsky turbulence model to resolve large to medium turbulent scales.

Fig. 4.1 presents the scale separation map for each case. For laminar flow (Fig. 4.1a), both the NS and the LB submodels are able to solve the same minimum scale. However, the NS submodel is able to resolve larger scales than the LB submodel; due to its low memory consumption and mesh refinement capabilities. Both submodels are able to solve the same maximum time scale. However, the LB submodel requires smaller time steps, and thus resolves smaller time scales, due to restrictions in the maximum stable LB units macroscopic velocity (see section 3.2). Simulating turbulent flow with turbulence modelling (Fig. 4.1b) increases the difference in the temporal and spatial scales each submodel resolves. In this case the LB-LES submodel resolves smaller time and spatial scales than the RANS submodel; but there is still overlap, a range of scales are solved by both submodels.

The NS and LB submodels for both the laminar and turbulent case present a multi-domain interaction coupled through boundary conditions. The computational domain of each submodel is divided in different sized cells and applied in different sections of the simulated domain. The LB subdomains include the regions where more detailed flow results are needed, while the NS subdomains model the rest of the domain. This is especially noticeable for the turbulent case, while in the laminar case the difference in scales is less.

Fig. 4.2 shows some examples of the positioning of the NS and LB subdomains. The NS and LB submodels are coupled at their internal boundaries (see boundaries marked as NS to LB or LB to NS in Fig. 4.2). The NS to LB boundaries are boundaries of the LB subdomain that are completely inside the NS subdomain. In these boundaries the information is transferred from the NS submodel to the LB submodel. The LB to NS boundaries are boundaries of the NS subdomain that are completely inside the LB subdomain. In these boundaries the information travels from the NS submodel to the LB submodel. The NS and LB subdomains can be partially overlapped (Fig. 4.2 (a) and (b)) or completely overlapped (Fig. 4.2 (c)). In Fig. 4.2 (c) the information only travels from the NS submodel to the LB submodels.

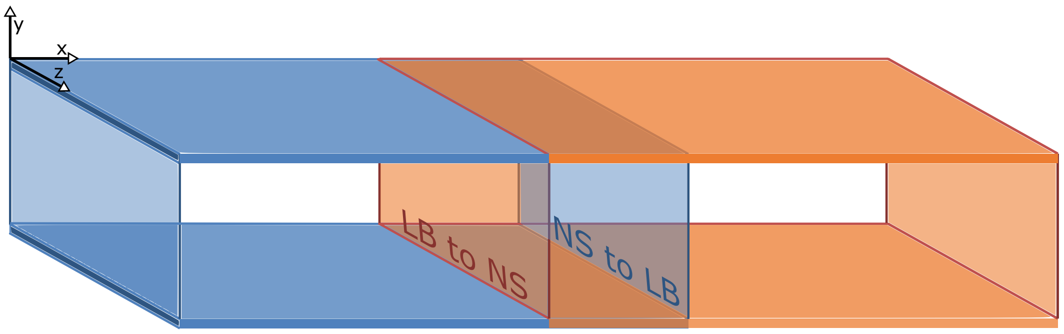
The submodels in a multi-domain multiscale method can be coupled one-way or two-way. The MMSF refers to one-way coupled submodels as loosely coupled, and two-way coupled submodels as tightly coupled.

In **one-way coupled** methods the information only travels in one direction, for example submodel A informs the boundary conditions of submodel B. An example of a one-way multi-domain method is the lattice Boltzman - Navier-Stokes method in [76] or the nested weather forecast method in [77].

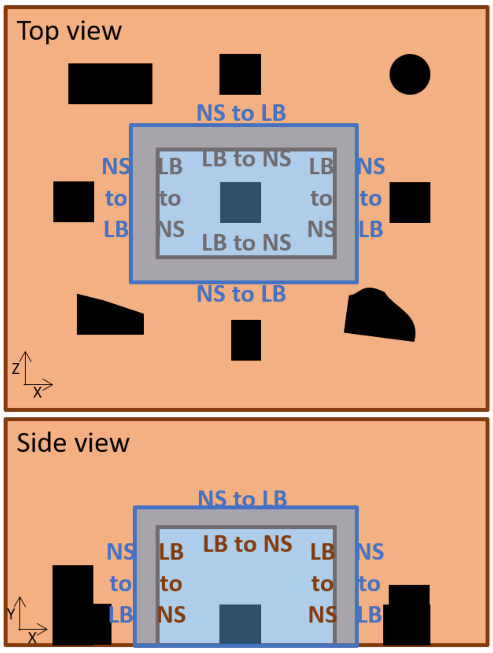
In **two-way coupled** methods the information travels in two directions, for example submodel A transfers information to the boundaries of submodel B and submodel B transfers information to the boundaries of submodel A. Two-way coupled methods can be further classified into two groups depending on whether the exchanged information comes from the previous time step (*explicit coupling*) or from the current time step (*implicit coupling*).

Two-way coupled models can be further divided by their coupling algorithm on implicit and explicit:

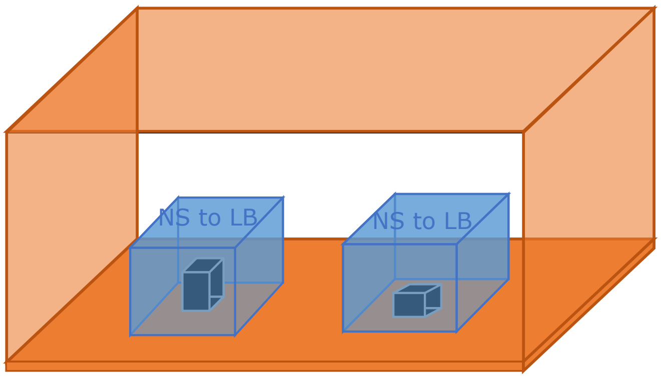
- **Implicit** coupled methods (Fig. ??) are generally required to iteratively



(a)



(b)



(c)

Figure 4.2: Examples of domain partition.The LB subdomains are blue and the NS subdomains are orange. All the blue boundaries in c) are LB to NS boundaries. The black prisms represent geometry the flow goes around, like buildings.

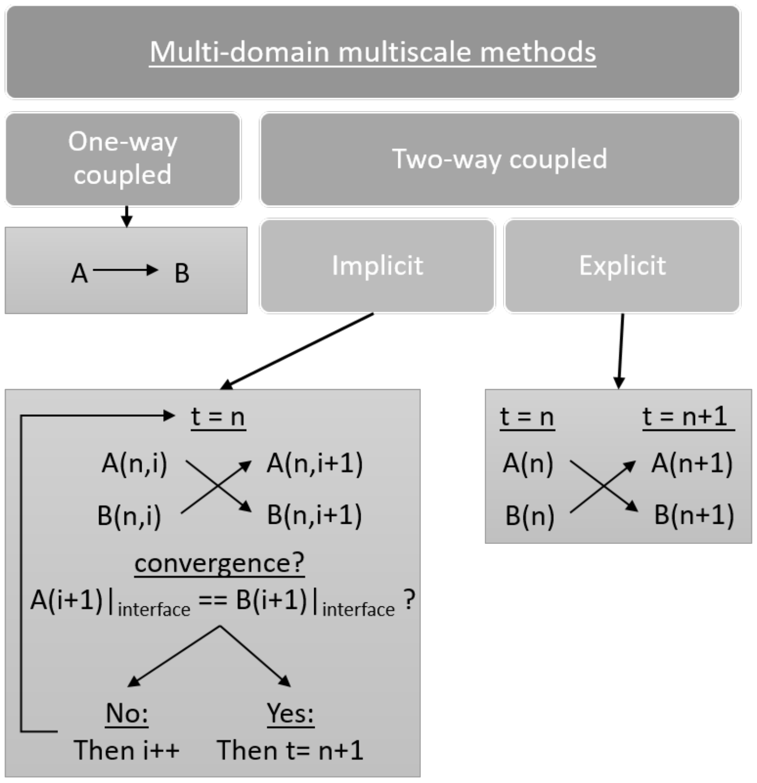


Figure 4.3: Scheme of the coupling methodologies for multi-domain multiscale methods.

solve for the current time step until the solution of both submodels agrees at their common interface. Implicit coupling is more accurate than explicit coupling, since the exchange takes place at the current time step, however, it is more computationally expensive, since the solution has to be iterated and usually requires extra stabilization steps [78].

- **Explicit** coupled methods exchange information from each submodel at the previous time step, such that no iterations are needed. Explicit coupling is more computationally efficient than implicit coupling, since there are no iterations, but less temporally accurate, since the results are not checked for consistency.

4.1.2 Coupling Navier-Stokes to lattice Boltzmann methods

Table 4.1 summarises the previous research studies coupling NS based solvers with LB solvers. The commonly-stated objective of these studies is to reduce the computational resources needed to obtain accurate results by using LB and NS in the regions of the domain they are more suited to. For example Neumann [8] and Tong and He [9] use lattice Boltzmann around the complex geometry porous region. Mivehchi et al. [76] apply lattice Boltzmann to solve the water flow next to the hull of a ship, where small cell size and time steps is needed to resolve the flow and Boundary Element method (BEM) to solve potential flow elsewhere, where sufficiently accurate results can be obtained using a much coarser resolution and time step.

	Mivehchi et al. [76]	Neumann [8]	Tong and He [9]	Velivelli and Bryden [79]	Atanasov et al. [10]
Application	Naval hydrodynamics	Transient laminar flow	Transient flow and heat transfer, porous medium	Transient laminar backward facing step	Porous media flow and others, steady state
Components	Potential flow solved using BEM for all the domain + MRT-LB-LES	Fine LB embedded in a coarse NS	Finite volume NS with incompressible LB	Finite differences vorticity-stream function to BGK LB	Eulerian NS and BGK LB.
Dimensions	3D	2D	2D-3D	2D	3D
Objective	Reduce computational resources				Show efficiency of Anderson accelerated coupling.
Coupling	one-way from BEM to LB	two-way explicit			two-way implicit
Data coupled	Forcing term from BEM added to LB	Velocity, presure, fluid stresses from NS to LB f_α using moments conservation.LB velocity and pressure to NS. There is an overlap region	Velocity, pressure and their spatial and temporal gradients from NS to LB f_α using reconstruction operators. LB pressure and velocity + mass flow correction to NS. Overlap region	Velocity from NS to LB f_α using forced equilibrium. LB velocity to NS. There is a overlap region.	Velocity, presure, fluid stresses from NS to LB f_α using moments conservation.LB velocity and pressure to NS. No overlap region.
Architecture	CPU/GPU	CPU			

Table 4.1: Characteristics of previous Navier-Stokes lattice Boltzmann hybrid methods.

All studies in Table 4.1 use multi-domain two-way coupling except Mivehchi et al. [76], which uses one-way coupling (from BEM to LB). All the two-way coupling studies are restricted to laminar flow and the coupling is achieved by exchanging pressure and velocity data at the interfaces between the NS and the LB regions. All the studies use essentially the same method to calculate the lattice Boltzmann macroscopic velocity from the distribution functions f_α before interpolating them to the NS interface, where they are used to inform the appropriate boundary conditions for the NS method. Moreover, Tong and He [9] add volumetric flow rate correction to the interpolated LB data at the NS interface. When it comes to the NS to LB coupling, the reviewed studies differ in the method of approximating the particle distribution functions at the LB interface from NS data. Velivelli and Bryden [79] calculate the particle distribution functions as the equilibrium distribution functions using the NS velocity. This is the simplest and more computationally efficient method to obtain f_α; however, all the non-equilibrium information in the NS pressure and velocity is lost, thus reducing the accuracy of the method. Neumann [8] and Atanasov et al. [10] solve a moment conservation minimisation problem to complement the equilibrium particle distributions with their non-equilibrium part. The minimisation problem is solved analytically using Lagrange multipliers and reduced to a matrix vector product calculated locally at each cell [10]. Tong and He [9] derive generalised Recontruction Operators to obtain f_α from NS pressure and velocity for any lattice Boltzmann formulation. This Reconstruction Operators also add additional information to the equilibrium distribution functions, but need to transfer more information than the method used by Neumann [8] and Atanasov et al. [10].

A fundamental issue to resolve in multi-domain coupling is the location and treatment of the interface between the coupled solvers. In Neumann [8] and Tong and He [9] the two interfaces are separated by more than one cell, creating an overlap region solved by the two solvers (Fig. 4.4); information is exchanged only at the boundaries. On the other hand, Atanasov et al. [10] presents matching LB to NS and NS to LB interfaces, with a one cell thick overlap region (Fig. 4.5). The discrepancy in the position of the interfaces might be related to the coupling scheme used, since Atanasov et al. [10] is the only paper using implicit coupling, the remaining papers use explicit coupling. Moreover, Tong and He [9] suggest that the size of the overlap region has to be enough to let the influence of the boundary condition at the NS to LB interface propagate across the LB domain before feeding the LB data to the NS submodel at the LB to NS boundary. This region is not needed in an implicit method, since the coupling is iterated until

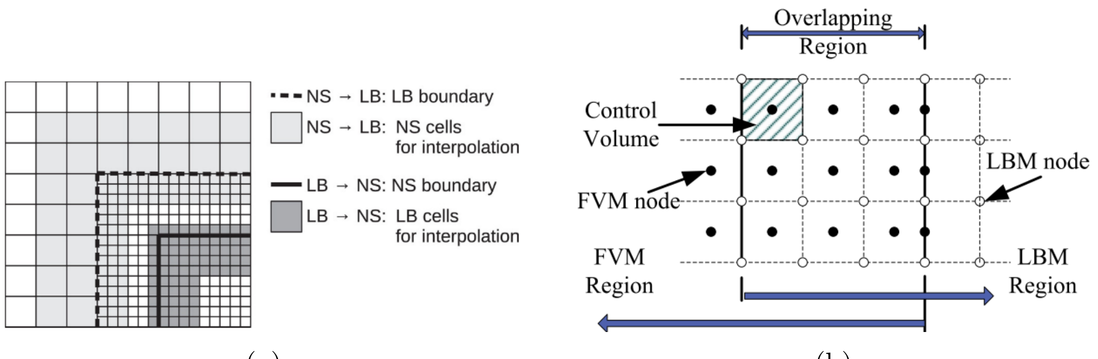


Figure 4.4: Solver interfaces and overlap region in Neumann [8] (a) and Tong and He [9] (b).

the solutions from both submodels match at the interfaces.

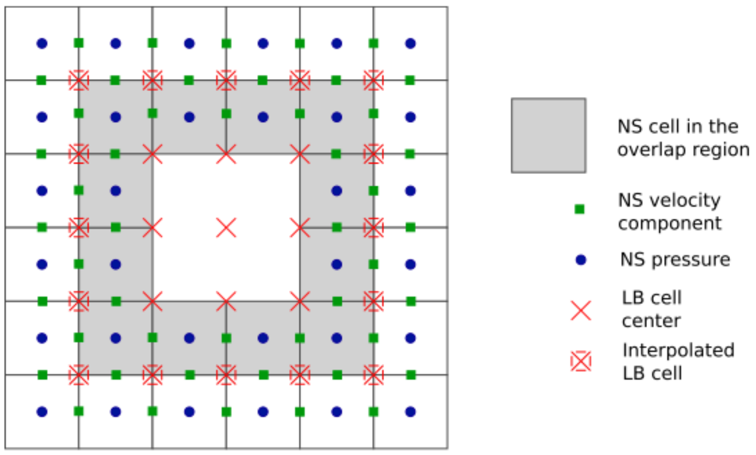


Figure 4.5: Solver interfaces and overlap region in [10].

Of the papers reviewed here, Mivehchi et al. [76] is the only to use a heterogeneous architecture, thus benefiting from the speed up achieved by implementing the lattice Boltzmann solver in GPU and the potential flow solver in CPU. An added advantage of using heterogeneous architecture is that the two methods do not compete for the same hardware resources, making load balancing easier.

Finally, all the papers that present two-way coupling simulated laminar flow. Only Mivehchi et al. [76] models turbulent flow, but it is one-way coupled.

4.1.3 Architecture of the NSLB method

The second of the theoretical framework is to describe the execution loop of the NS and LB submodels and and its coupling templates. The SEL divides the time loop execution of each submodel in the following generalised operations:

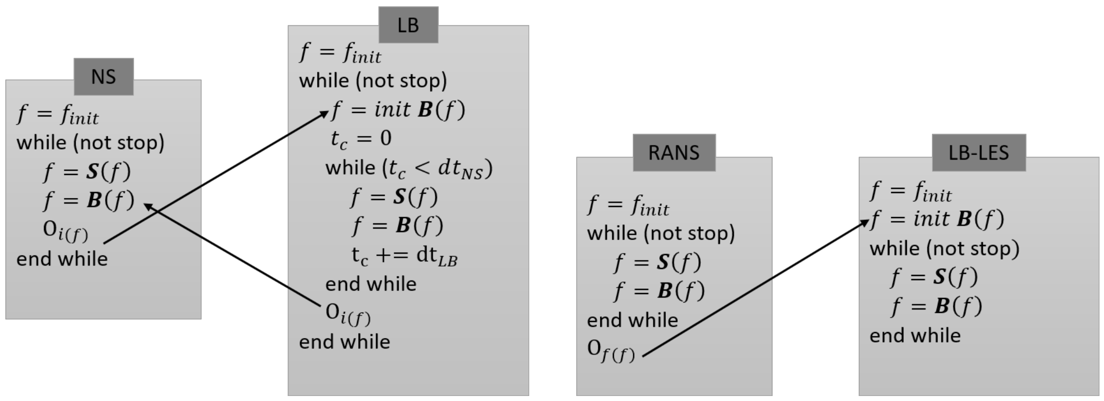
- S : refers to solver and it includes all the core operations in one time step of the the submodel.
- B: refers to boundary and it includes the application of boundary conditions. In some cases, as is for the LB submodel, the BC are integrated into the core solver, however it is useful to separate them in the conceptual SEL.
- f_{init}: initialisation of the submodel; including mesh generation, initialising solved variables and the termination condition for the time loop if needed.

The SEL also defines two observation operators O_i and O_f, where i denotes intermediate and f denotes final. Observation operators compute desired quantities from the variables of the submodel. These computed quantities are then handed to a smart conduit, which will transform them as needed and send them to the coupled submodel. The coupled submodel will receive the quantities in either the S, F, or f_{init} operator.

Using the previous work on coupled NS and LB solvers presented in the previous section as a guide and the SSM and modelling framework, we developed the SELs for the laminar case (Fig. 4.6a) and for the turbulent case (Fig. 4.6b).

In the laminar flow case (Fig. 4.6a), the NS submodel is two-way coupled with the LB submodel. Both submodels simulate the same amount of physical time and are coupled at the end of each NS time step. The LB time step is smaller than the NS time step, thus the LB submodel *subcycles* through a number of time steps until it synchronises with the NS time step; at this point the LB submodel sends data to the NS submodel boundary and the NS submodel sends data to reinitialise the LB submodel boundary before starting the next LB subcycle. The observation operators compute the macroscopic instantaneous velocity, which is imposed at the coupled submodel's boundaries using a Dirichlet boundary condition.

In the turbulent flow case (Fig. 4.6b), the RANS sumbodel is one-way coupled



(a) Laminar flow (b) Turbulent flow

Figure 4.6: Submodel execution loop (SEL) of the NSLB method for the laminar case (a) and turbulent case (b). S refers to solver, B refers to boundary, f_{init} refers to solver initialisation. O_i and O_f are the observation operators; the subscript i denotes intermediate observations and the subscript f denotes final observations. dt_{NS} is the time step of the NS submodel and dt_{LB} is the time step of the LB submodel.

with the LB-LES submodel. The results of the RANS submodel are used to initialise the boundary data of the LB-LES submodel. The time and spatial scale differences are bridged using a SEM method applied to the LB-LES boundary. The SEM method used is described in section 3.3.3 of this thesis. The observation operator computes the RANS velocity, Reynolds stresses and turbulent kinetic energy dissipation rate. The LB-LES submodel, then uses this data as mean flow data for the SEM boundary condition. In this case, SEM acts as a bridge from the RANS scales to the LB-LES scales.

4.2 Computational framework

4.2.1 Implementation

The information is transferred and adapted between submodels using scale aware smart conduits, which can be coded using three different approaches: *monolithic*, using a *coupling framework* or using a *coupling library* (Fig. 4.7):

- **Monolithic** methods (Fig. 4.7 a)) do not reuse any of the submodels code, they are a completely new piece of software. They are efficient in the sense

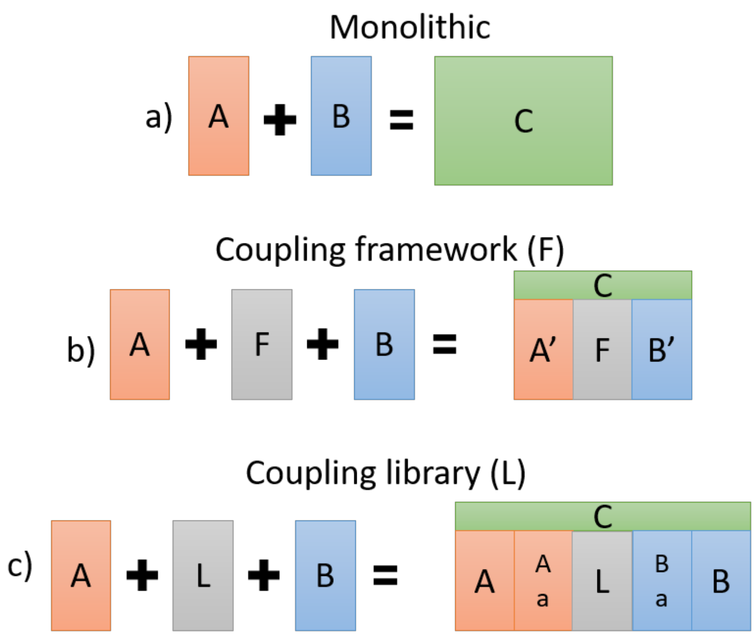


Figure 4.7: Summary of the coding strategies for multiscale methods: a) monolithic, b) coupling framework, c) coupling library. C is the resulting multiscale method, A and B are the two coupled submodles in their original form, A' B' are modified to plug into the framework (F) and Aa and Ba are the adaptors to connect A and B respectively to the coupling library (L).

that the code is tailored and optimised to the multiscale method, however, they are rigid, since to couple one submodel to another means rewriting the code and programming basic features present in all methods like writing results, parallel running etc.

- **Coupling frameworks** (Fig. 4.7 b)) (for example Cactus [80] and OpenPalm [81]) aim to palliate the rigidity of monolithic methods. They offer a set of basic software components (like MPI, data input/writing, grid creation...) that are general enough to accept different solvers to be plugged in. Using a *coupling framework* allows the user to change the coupled submodels without having to rewrite/duplicate code, however, the original submodels have to be modified to be compatible with the *coupling framework* structure.

- **Coupling libraries** (Fig. 4.7 c)) (see for example preCICE [78], OASIS 3-MCT [82] and MxUI [83]) communicate information between the submodels of a multiscale method. The only modification needed to the original submodels is to add an *adaptor* code, which transmits the needed information from each solver to the *coupling library*. The *coupling library* facilitates the exchange of information including data communication protocols and interpolation of the exchanged data to each solver's grid amongst others, which depend on the library used. *Coupling libraries* couple two submodels with minimal modifications to their original code. However, their code is less tailored to each submodel, which may result in a loss of efficiency compared to the *coupled framework* and the *monolithic* approaches.

The enumeration above names a handful coupling frameworks and coupling libraries. The reader is referred to the review by Groen et al. [84] for an in depth view and comparison of coupling libraries and frameworks.

The smart conduits in the NSLB method are coded using the coupling library preCICE [78]. The NS and LB submodels that form the NSLB method are existing software that is used both as part of the NSLB method and as independent solvers. Thus, we needed to code the coupling minimising the modifications to the original code of the submodels in order to maintain their usability and performance as independent solvers. preCICE is a coupling library, which only needs an adapter code to be incorporated to the submodels; this adapter code is decoupled from the core of both the NS and LB solvers, thus making it easier to maintain. We chose preCICE because it is open source and offers clear documentation, tutorials and user support. Moreover, it controls the SEL of each submodel, synchronising it with the coupled submodels and sending / receiving information when needed. preCICE also offers one-way and two-way implicit and explicit coupling, providing different mapping and interpolation methods.

Implementation of the NSLB method

The software used to for the NS and RANS submodels is the *pisoFoam* solver from the open source CFD package OpenFOAM v4.x [27]. We chose OpenFOAM because it is open source, robust, stable and widely used in the research community. Moreover, preCICE provides and mantains an adapter code to connect to OpenFOAM. The software used to program the LB and LB-LES submodels is GASCANS (see Appendix B). GASCANS is a multi-GPU LB code programmed to accept boundary conditions from external sources at run time and extract data from the domain at run time; which makes it an ideal choice for the NSLB method. OpenFOAM is programmed in C++ and GASCANS is programmed in C++ and CUDA.

Fig. 4.8 shows the code structure of the NSLB method emphasising the connection with preCICE and the classes that enable the exchange of information between the NS and the LB codes.

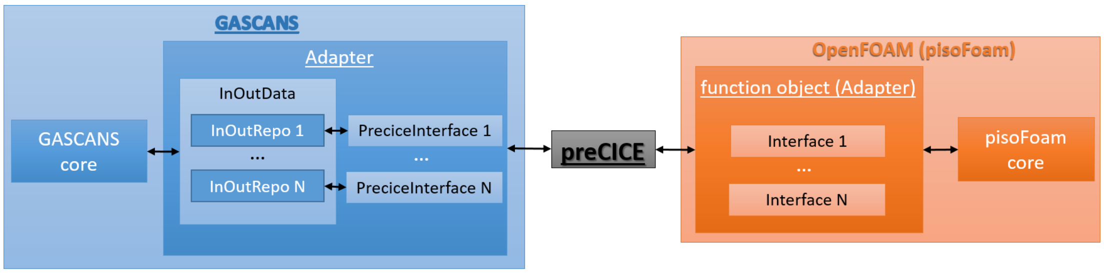


Figure 4.8: Class structure of the preCICE adapters for both GASCANS and OpenFOAM. GASCANS stores the coupled data in the InOutRepo objects and OpenFOAM stores it in the Interface objects. N is the number of coupled interfaces. The arrows show the flow of the coupled data.

The OpenFOAM preCICE adapter is a modified version of the adapter developed by Chourdakis [85] [1]. The main advantages of Chourdakis' adapter are that it does not modify the original OpenFOAM code, can be coupled to a range of OpenFOAM solvers and can be activated and deactivated using only configuration files. Moreover it is compatible with parallel OpenFOAM simulations.

Chourdakis' preCICE adapter can only exchange information stored in OpenFOAM *patches*, which are the boundaries of the domain. However, the NS to LB boundaries are not boundaries in the NS sub-domain (Fig. 4.9 and Fig. 4.2) Thus, we modified Chourdakis adapter [2] to exchange data at *cellSets*; a *cellSet* is group of cells set by the user anywhere in the OpenFOAM domain. We also added a volumetric flow rate correction at the receiving OpenFOAM boundaries as described by Tong and He [9] (see section 4.2.1).

The preCICE adapter for GASCANS is based on the OpenFOAM preCICE adapter developed by Chourdakis [85]; see Appendix B for an in depth description of the GASCANS adapter.

[1]Code available in https://github.com/precice/openfoam-adapter/tree/FF

[2]Modified preCIE adapter code available in https://github.com/martacamps/openfoam-adapter/tree/FF

Information transfer

The information to transfer is extracted from the sending sub-domain and copied to the *sending mesh*. Then preCICE interpolates the information on the sending mesh to the *receiving mesh*, which is extracted from the receiving sub-domain. The sending and receiving meshes are represented by their cell's 3D coordinates and data that needs to be comunicated to/from the other sub-domain. Fig. 4.9 illustrates an example of the sending and receiving meshes for NS to LB boundary and a LB to NS boundary. The receiving mesh is formed by the boundary cells of the receiving solver. The sending mesh is formed by the cells in the sending sub-domain that overlap with the receiving sub-domain's boundary. A pair of sending and receiving meshes is created for each coupled boundary in the domain.

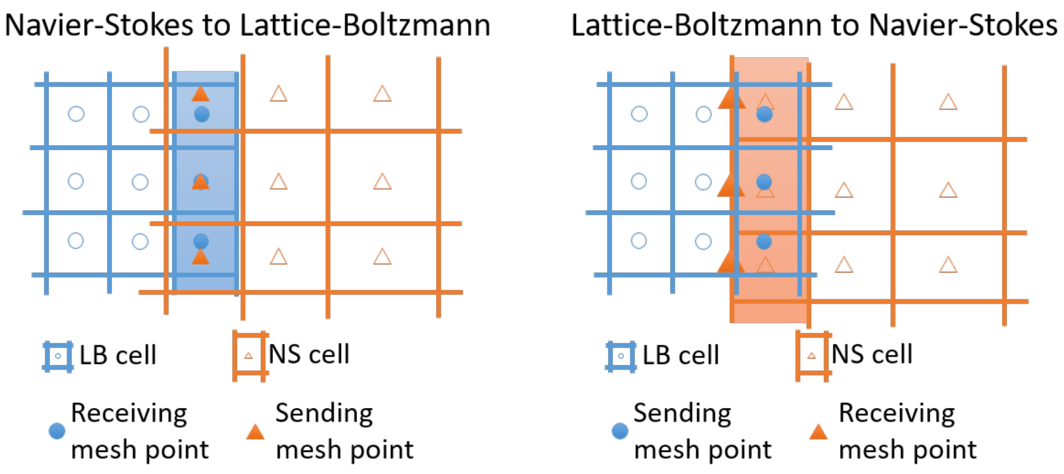


Figure 4.9: 2D sketch of the receiving and sending meshes for a NS to LB boundary (left) and a LB to NS boundary (right). The cells belonging to the boundary of the LB sub-domain are coloured in blue, while the cells belonging to the boundary of the NS sub-domain are orange.

The NS and LB meshes are non-conformal, thus the data sent needs to be interpolated to the receiving mesh. preCICE offers different interpolation methods with different complexity and accuracy (see preCICE wiki [86]). From them, we chose the nearest neighbour interpolation since it is the simplest and does not require mesh connectivity data. The difference between the NS and LB sub-domain meshes in regions with high velocity gradients is not anticipated to be big enough to require a higher order interpolation method.

Boundary conditions

The **LB to NS boundaries** are part of the Navier-Stokes sub-domain and implement a Dirichlet boundary condition for velocity and a zero gradient boundary condition for pressure. The velocity data comes from the LB sub-domain and is interpolated to the LB to NS boundary by preCICE. The LB sub-domain solves the macroscopic velocity in lattice units u_i^l and converts it to the dimensionless/-physical units u_i^d used by the NS solver via Eq. 4.1 before transferring it to preCICE; see section 3.2.3 for more information about lattice units.

$$u_i^d = u_i^l \frac{\delta x}{\delta t} \tag{4.1}$$

where δx is the cell size and δt is the time step of the lattice Boltzmann sub-domain in the Navier-Stokes units.

The LB to NS boundary can correct the interpolated velocity so that it matches a prescribed volumetric flow rate as described by Tong and He [9]. First, the NS solver calculates the volumetric flow rate from the interpolated velocity at the LB to NS boundary; then it divides the calculated flow rate by the prescribed one. Finally, it multiplies the velocity at each cell of the boundary by the ratio between the two flow rates. This correction is often needed in 1D two-way coupled NSLB simulations (Fig. **??** a)). Some Navier-Stokes solvers, for example OpenFOAM [27], modify the velocity at the fixed pressure outlet boundaries to compensate for the mass imbalance due to numerical errors. In the case of the NSLB method, one of the sources of mass imbalance comes from the interpolation of the lattice Boltzmann velocity to the LB to NS boundary. The outlet velocity correction applied by OpenFOAM also affects the velocity that will be interpolated to the lattice Boltzmann sub-domain, which in turn affects the LB velocity that will be interpolated to the NS sub-domain. The change in the velocity interpolated to the LB to NS boundary will affect the mass correction at the Navier-Stokes sub-domain outlet. If left uncorrected this feedback loop reduces the quality of the results and might destabilize the coupling. Fig. 4.10 shows an example of this effect.

The **NS to LB boundaries** of the LB sub-domain implement a forced equi-

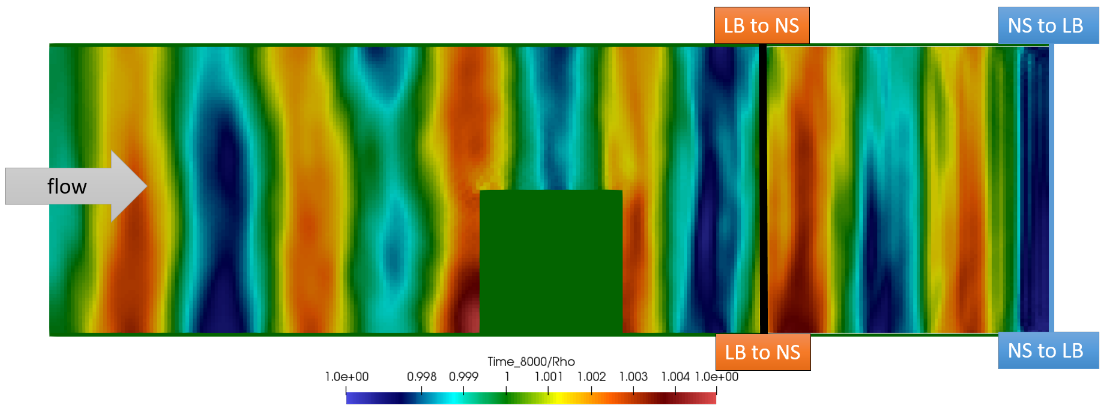


Figure 4.10: Lattice density on a 2D slice through the lattice Boltzmann sub-domain of a NSLB simulation of a wall mounted cube in a channel flow, Re = 150. Oscillations in density caused by the mass flow correction in the Navier-Stokes sub-domain. The NS to LB (blue) and LB to NS (black) boundaries are marked in the figure.

librium boundary condition (see section 3.2.5). The density at the NS to LB boundaries cells is set to 1, while their macroscopic velocity is set to the velocity interpolated form the Navier-Stokes sub-domain. The LB solver then converts the dimensionless/physical velocity u_i^d from the NS sub-domain to lattice velocity units u_i^l using Eq. 4.1.

The forced equilibrium boundary condition is the simplest of the lattice Boltzmann Dirichlet boundary conditions. It is also the least accurate and the most stable (see section 3.2.5). The laminar test case in section 4.3.1 shows that the accuracy at the LB to NS boundary is lower than at the LB to NS boundary, which might be due in part to the lattice Boltzmann forced equilibrium boundary condition. The accuracy might be increased by using a more accurate boundary condition as for example reconstructor operators [9], non equilibrium minimisation [10] or regularised boundary (section 3.2.5). At present, the results obtained with forced equilibrium are deemed sufficient for the scope of this thesis.

4.2.2 Execution

The LB submodel GASCANS is designed for small clusters and consumer level workstations, it runs in single node multi-GPU. On the other hand, both preCICE and the NS submodel pisoFoam are able to run in multi-node clusters. Then the NSLB method can run using multi-node clusters for the NS submodel, but the

LB submodel has to be run in a single node.

4.3 Validation and examples

Validation of the NSLB method is presented in the two following papers:

- Paper I, section 4.3.1 applies the NSLB method for laminar flow to a $Re_H =$ 150 flow over a wall mounted cube in a channel flow. The results of the NSLB method are validated against DNS data by Hwang and Yang [87]. The paper also studies the computational resources used by the NS, LB and NSLB methods for the test case and the effect of the overlap region's size and position.

- Paper II, section 4.3.2 applies the NSLB method for turbulent flow. A LB-LES sub-domain is embedded inside a pre-calculated RANS sub-ddomain modelling flow around a rectangular building at $Re_H = 47893$. The RANS results are used to set the boundary values of the LB-LES sub-domain. The paper compares the RANS and LB-LES results and computation time with the experimental data by Meng and Hibi [88] and DDES.

4.3.1 Paper I - Two-way coupled Navier-Stokes / lattice Boltzmann solver to reduce the resources used by CFD simulations of flow around bluff objects

The following journal paper applies the NSLB method described in chapter 4 to a $Re_H = 150$ flow around a wall mounted cube. The paper requires some revision and the addition of a turbulent test case before submission to the *Journal of Comptational Physics*, which is the journal we intend to submit this paper for.

Two-way coupled Navier-Stokes / lattice Boltzmann solver to reduce the resources used by CFD simulations of flow around bluff objects.

Marta Camps Santasmasas[a], Alistair Revell[a], Ben Parslew[a]

[a]*School of Mechanical, Aerospace and Civil Engineering, University of Manchester, UK*

Abstract

The use of accurate CFD for external aerodynamics for industrial applications is limited by the availability of computational resources and time. This work was motivated by urban wind flow simulations, but it is expected to have impact in simulations where computational domains include an interest region where more flow detail is needed. One way to reduce the computational demand of the simulation is to combine two different solvers in order to exploit the advantages of each one in different regions of the domain. In this lines, we present a Navier-Stokes solver two-way coupled with a lattice Boltzmann solver for three-dimensional unsteady flow around bluff objects. The simulation domain is divided into a sub-domain modelled using Navier-Stokes equations computed using CPU and a sub-domain modelled using lattice Boltzmann equations computed using graphics processing units (GPUs). Both domains are coupled at their overlapping boundaries. We tested the Navier-Stokes lattice Boltzmann (NSLB) solver on flow around a wall mounted cube at Re 150. The NSLB results agree with the results obtained by the individual solvers with a difference below 1%. Moreover, the NSLB simulation requires less computational time than the equivalent NS simulation and less GPU memory than the single LB simulation; allowing to run simulations that would spend too many GPU resources if run with single LB, at a faster pace than the single NS solver. The accuracy and stability of the NSLB solver is affected by the complexity of the flow at the coupled boundaries and the size of the region where the NS and LB sub-domains overlap.

Keywords: Lattice boltzmann method, GPU, code optimisation, computational resources, coupled methods.

1. Introduction

Computational fluid dynamics (CFD) is a well established tool to predict the behaviour of fluid flow in various engineering fields. However, the amount of information provided and accuracy of the results is restricted by the computational time and resources available for each project. CFD simulations are often performed on super-computer clusters using from hundreds to thousands of CPU cores. On the other hand, graphic processing units (GPUs) are being increasingly used in CFD due to their ability to perform massive parallel computations at a low cost. Moreover, GPUs are usually part of workstations and personal computers.

CFD methods based in the incompressible Navier-Stokes equations (NS) and CFD methods based in the lattice Boltzmann equations (LB) are able to predict flow behaviour and are well established in the literature. Both methods present strengths and weaknesses that make each method more suitable to different applications and domain configurations. The main strength of the LB methods lies in the simplicity and locality of their algorithm; which enables an efficient implementation in massivelly parallel architectures like GPU. Besides, LB mesh configuration is straightforward, since all the cells are the same size and the introduction of geometry only requires marking the cells part of the geometry as solid cells. This simple implementation of geometry makes LB a popular method to model flow through complex geometries (f e. porous media [1]). However, the basic LB method is memory intensive which, combined with the limited memory in GPU cards, limits the size of the domain/resolution that it is practically attainable with a single computer. The main strength of the NS methods lies in their flexibility, low memory consumption and stability. NS methods work efficiently with mesh refinement, thus reducing the number of cells in regions with low flow complexity and saving computational resources. However, the complexity and iterative nature of the NS algorithms generally leads to an increase in computational cost and a more difficultû GPU implementation.

Mivechi et al. [2], Neumann [3], Tong and He [4] and Velivelli and Bryden [5] couple a LB method with a NS method to reduce the computational resources of the CFD simulation. All of them implement segregated coupling algorithms, i e. the NS method and LB method are run in two separated sub-domains coupled at their boundaries. Neumann [3], Tong and He [4] and Velivelli and Bryden [5] are two-way coupled, i. e. the information travels from the NS sub-domain to the LB sub-domain and vice versa. Mivechi et al. [2] is the only reviewed paper that implements the LB sub-domain in GPU and the NS sub-domain in CPU but the two solvers are only coupled one-way, from NS to LB. The reviewed papers demonstrate the advantages of coupling a NS method to a LB method. However, the two-way coupled solvers do not take advantage of the GPU implementation of the lattice Boltzmann model. Moreover, the reduction on computational resources by using the coupled methods or the effect of the size and position of the region of the domain solved by both the NS and the LB solver is not thoroughly investigated.

We present a LB flow solver implemented on GPU coupled with a NS solver implemented on CPU. The GPU accelerated LB solver models the region of interest at high resolution; since the LB region is limited, the NSLB method overcomes the mesh and memory related drawbacks of using GPU accelerated LB for the whole domain. The CPU NS solver models the remainder of the domain, thus taking advantage of the mesh flexibility and lower memory consumption of the NS method. The coupling is done by exchanging the velocity field values at the interface boundaries. The NSLB model is able to reproduce the results of a single NS and LB model while reducing the resources used. However, the accuracy of the NSLB method is affected by the size and position of the region where the NS sub-domain overlaps with the LB sub-domain.

2. Navier-Stokes model

The NS sub-domain solves the three-dimensional incompressible Navier-Stokes equations plus the continuity equation,

$$\begin{aligned} \frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} &= -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} && (1) \\ \frac{\partial u_i}{\partial x_i} &= 0 \qquad i,j = 1-3 && (2) \end{aligned}$$

where u_i is the velocity of the fluid, p is its pressure and ν its kinematic viscosity. Eq. 1 is discretised employing a finite volume method configured to use linear interpolation schemes and an implicit Euler scheme for time integration. The pressure-velocity coupling is solved using the PISO algorithm [6].

The software used is the *pisoFoam* solver from the open source CFD package OpenFOAM [7]. However, this software can be substituted for any other NS solver that is able to provide the velocity field at a boundary during run time.

2.1. Advantages and disadvantages

The main strength of finite volume NS models is that the most common algorithms are unconditionally stable and the cell size and time step do not need to be constant. Thus allowing to increase the resolution in the regions of interest and where the flow is more complex and decrease it elsewhere. Moreover, the NS solver only stores four quantities at each cell: u_x, u_y, u_z and p.

The main weakness of the NS solver is the complexity of its numerical algorithm. It requires non-local operations and iterative procedures such as discretisation of velocity gradients and pressure-velocity coupling, which decrease the efficiency of a GPU implementation. Moreover, the basic model requires the mesh to be adapted to the contour of the geometry, which increase the complexity of the meshing algorithms and the implementation of wall boundary conditions.

3. Lattice Boltzmann method

The LB sub-domain solves the three-dimensional lattice Boltzmann equations with a 3DQ19 lattice model and a BGK collision operator. The lattice Boltzmann equations solve for the particle distribution f_α, which models the probability of a group of fluid particles to move at a velocity $\mathbf{c}_\alpha$ at a certain point of space and time. The particle velocity space is discretised in a number of velocities $\mathbf{c}_\alpha$ defined by the lattice model. 3DQ19 contains the following particle velocities:

$$\mathbf{c}_\alpha = \begin{cases} 0, & \alpha = 0, \\ (\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1) & \alpha = 1 \quad to \quad 6, \\ (\pm 1, \pm 1, 0), (\pm 1, \pm 0, \pm 1), (0, \pm 1, \pm 1) & \alpha = 7 \quad to \quad 18. \end{cases} \tag{3}$$

The time evolution of a particle distribution f_α is

$$f_\alpha(\mathbf{x} + \mathbf{c}_\alpha dt, t + dt) = f_\alpha(\mathbf{x}, t) - \frac{1}{\tau}(f_\alpha(\mathbf{x}, t) - f_\alpha^{eq}(\mathbf{x}, t)) \tag{4}$$

$$f_\alpha^{eq}(\mathbf{x}, t) = w_\alpha \rho \left(1 + \frac{u_i c_{i\alpha}}{c_s^2} + \frac{(u_i c_{i\alpha})^2}{2c_s^4} - \frac{u_i u_i}{2c_s^2}\right) \tag{5}$$

$$i = 1 \to 3 \qquad \alpha = 0 \to 18$$

where $f_\alpha(\mathbf{x}, t)$ is the particle distribution for the microscale velocity $\mathbf{c}_\alpha$ at position $\mathbf{x}$ and time t, τ is the relaxation time and $f_\alpha^{eq}(\mathbf{x}, t)$ is the equilibrium particle distribution for the particle velocity $\mathbf{c}_\alpha$ at position $\mathbf{x}$ and time t. u_i and ρ are respectively the macroscopic velocity and density at position $\mathbf{x}$ and time t; $c_s = 1/\sqrt{3}$ is the lattice sound speed and $w_0 = 1/3$, $w_\alpha = 1/18$ for $\alpha = 1 \to 6$ and $w_\alpha = 1/36$ for $\alpha = 7 \to 18$. The macroscopic density ρ and velocity u_i are obtained from the moments of f_α as

$$\rho = \sum_{\alpha=0}^{18} f_\alpha \tag{6}$$

$$u_i = \frac{c_{i\alpha} f_\alpha}{\rho} \quad \alpha = 0 \to 18 \; ; \; i = 1 \to 3 \tag{7}$$

A weakly compressible form of the Navier-Stokes equations can be obtained via a Chapmann-Enzkog expansion of equation 4 [8]; which relates the relaxation time τ to the kinematic viscosity ν as

$$\nu = c_s^2 \left(\tau - \frac{\Delta t}{2}\right) \tag{8}$$

The LB spacial domain is usually discretised in a uniform grid of cell size 1 lattice space unit. Similarly, time is discretised in time steps of 1 lattice time unit. This way, the fluid particles in a cell travelling at the discretised particle velocities $\mathbf{c}_\alpha$ reach their neighbour cell at the next time step, thus simplifying the LB algorithm. Quantities in lattice units can be converted to physical units using the cell size and time step in physical units, thus:

$$u_i^{phys} = u_i^{lbm} \frac{\delta x}{\delta t} \tag{9}$$

$$\nu^{phys} = \nu^{lbm} \frac{\delta x^2}{\delta t} \tag{10}$$

where *phys* denotes physical units, *lbm* denotes LB units, δx is the cell size in physical units and δt is time step size in physical units.

The LB algorithm contains two steps: streaming and collision. Streaming models convection and it consists in copying each particle distribution f_α from one cell to its neighbouring cell in the direction of $\mathbf{c}_\alpha$. Collision models the collisions of the fluid particles in a cell and it consists in applying eq. 4, eq. 6 and eq. 7.

The LB software used to solve the LB sub-domain is GASCANS, a GPU-accelerated LB code developed at the University of Manchester [9].

3.1. Advantages and disadvantages

The main strength of the LB method lies on its simplicity. The interactions between cells during streaming are linear (one cell only affects its nearest neighbours), and all the heavy computations included in the collision step are local. This simplicity and locality increases the efficiency of GPU acceleration. Moreover, the streaming step models advection without numerical diffusion.

The main weakness of the LB method is that it is only conditionally stable. The explicit time integration and its weakly compressible nature restricts the stable combinations of temporal and spatial discretisation. Moreover, the assumption that the discrete particle velocities reach from one cell to exactly the boundary of its neighbours combined with that the viscosity (eq. 8) depends also on the time and space discretisation makes mesh refinement cumbersome to apply. Another disadvantage is that the LB method is memory intensive, since the 3DQ19 BGK model has to store 23 quantities for each cell: 19 f_α, u_x, u_y, u_z and ρ.

4. Navier-Stokes lattice Boltzmann (NSLB) model

The NSLB solver couples a sub-domain solved using a NS based solver with a sub-domain solved by a LB based solver. The main objective of the NSLB solver is to combine the simple algorithm, accuracy and efficiency on GPU of the LB method with the grid refinement, stability and low memory consumption of the NS method. The current section describes the NSLB method configuration and algorithm applied to the flow around the wall mounted cube test case described in section 5.1.

The total domain simulated is divided into a LB sub-domain and a NS sub-domain. Both sub-domains exchange information at their boundaries: the LB to NS boundary is the inlet of the NS sub-domain and receives data from the LB sub-domain; the NS to LB boundary is the outlet of the LB sub-domain and receives data from the NS sub-domain.

The LB to NS boundary implements a Dirichlet boundary condition for velocity and a Neumann boundary condition for pressure. The NSLB method interpolates the macroscopic velocity (eq. 7) in the LB sub-domain to the LB to NS boundary of the NS sub-domain and converts its values from lattice units to physical units (eq. 9). It also corrects the interpolated velocity to match a prescribed volumetric flow rate as described by Tong and He [4]. This correction is needed to compensate the mass imbalance at the LB to NS boundary due to interpolation precision and numerical errors. If left uncorrected, some NS solvers will correct the mass imbalance by modifying the velocity at the NS sub-domain outlet, which will in turn modify the velocity transmitted to the LB sub-domain and thus affect the velocity at the LB to NS boundary and so on. Thus reducing the accuracy of the results and potentially destabilising the coupling.

The NS to LB boundary on the LB sub-domain implements a forced equilibrium boundary condition, which fixes the density and velocity field and calculates the corresponding particle distribution functions f_α with the equilibrium distribution function (eq. 5). At the NS to LB boundary, the LB density is fixed to 1 and the LB macroscopic velocity is fixed to the NS velocity in LB units (eq. 7) interpolated from the NS sub-domain. The forced equilibrium boundary condition assumes that the LB flow is in equilibrium, so the non-equilibrium information carried by the NS velocity is lost. The forced equilibrium boundary condition is the simplest and most stable of the LB boundary conditions; but it is also the less accurate.

The overlap region is the volume between the LB to NS boundary and NS to LB boundary and it is solved by both the NS and LB solver. The overlapping region acts as a buffer to prevent the data transferred from one solver to overwrite the data transferred by the other solver. The optimum size of the overlap region depends on the characteristics of the flow (see section 5.2.3) and the coupling algorithm. Atanasov et al. [10] present an implicit coupling algorithm without an overlap region. However, implicit coupling requires both the NS and LB sub-domain simulations to be iterated until agreement is achieved at the coupled boundary, thus increasing the computational time.

The NSLB model advances in time using a parallel-explicit coupling scheme [11] (see Fig. 1). Both solvers exchange the data at their coupled boundaries once at the end of the coupling time step without checking if the information send by one solver is coherent with the results present in the receiving sub-domain. This coupling algorithm is computationally efficient, since the sub-domains are only solved once for each time step. However, the simulation might become unstable if the two sub-domains transmit contradictory information in close proximity.

Both the NS solver and the LB solver, are executed simultaneously with synchronisation barriers that halt their execution when reaching the end of the coupling time step until both solvers are ready to exchange data. The BGK LBM presents a lower range of stable Courant–Friedrichs–Lewy (CFL) numbers, thus the time step needed to obtain accurate results for the LB solver is usually smaller than the one needed by the NS solver. Thus the LB model will need to perform more than one time step for each NS in order for them to exchange information at the same simulation time. This process is called subcycling and it is illustrated in Fig. 1. Note that if the time step size of a solver is fixed, it needs to be a multiple of the coupling time step in order to allow the solvers to reach the same physical time to exchange information.

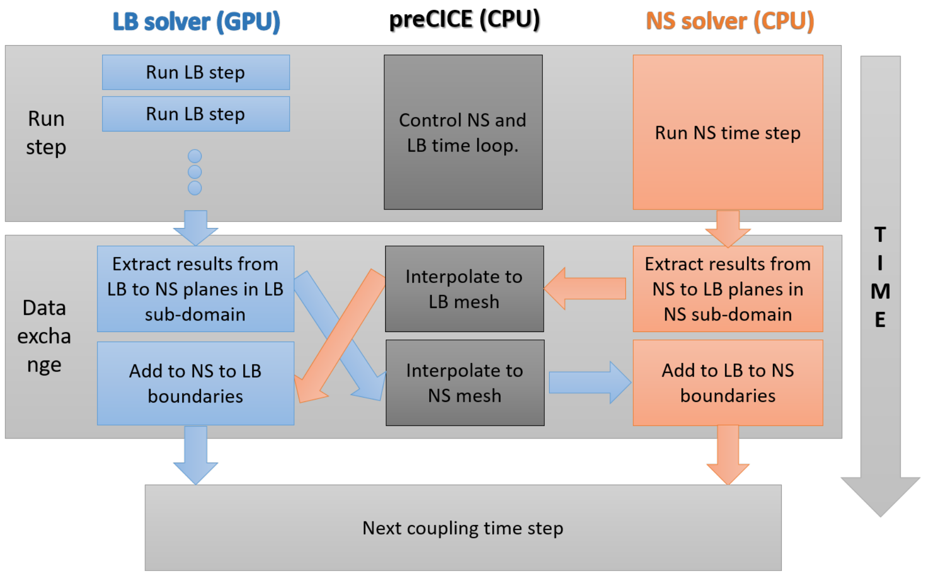

LB solver (GPU)
preCICE (CPU)
NS solver (CPU)
Run step
Run LB step
Run LB step
Control NS and LB time loop.
Run NS time step
Data excha nge
Extract results from LB to NS planes in LB sub-domain
Interpolate to LB mesh
Extract results from NS to LB planes in NS sub-domain
Add to NS to LB boundaries
Interpolate to NS mesh
Add to LB to NS boundaries
Next coupling time step
T I M E

Figure 1: Scheme of a time step of the NSLB model.

The coupling algorithm is implemented using preCICE libraries [11] and the OpenFOAM preCICE adapter by Chourdakis [12]. We modified Chourdakis' adapter to transfer and receive pressure and velocity data from both the domain boundary and internal cells. preCICE controls the time stepping of both solvers and the data exchange between them. It also interpolates the data received from one solver to the mesh of the other. Our NSLB solver uses the nearest neighbour interpolation from preCICE.

5. Test case: Flow around a wall mounted cube

5.1. Case description

This test case models laminar flow around a wall mounted cube in a channel (see Fig. 2) and is based on the numerical study by Hwang and Yang [13]. We chose this test case due to its bounded top boundary and its periodic spanwise boundaries, which allows us to use smaller grids and thus test more configurations with a relatively low computational cost.

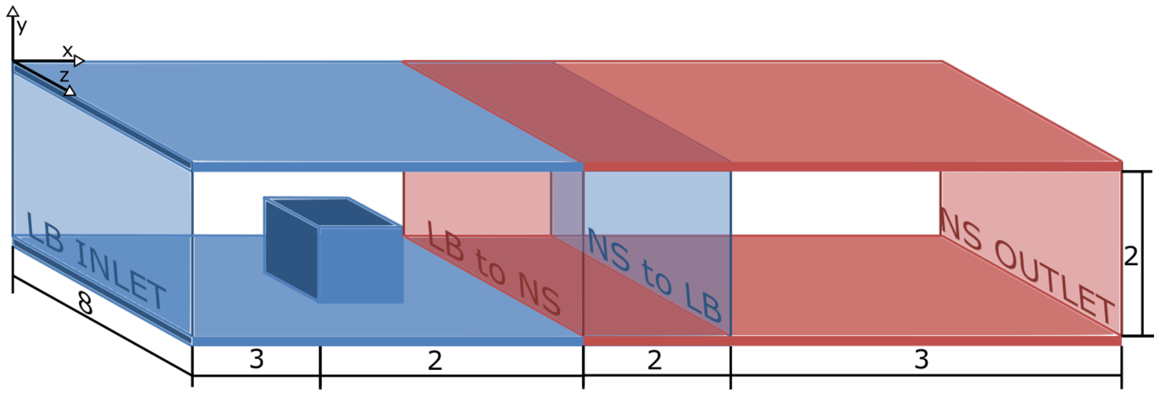


Figure 2: Sketch of the test case. The opaque forms denote geometry while the transparent ones are boundaries. Not to scale. Distances are in dimensionless units.

The Reynolds number of the flow is $Re_H = 150$ based on the height of the cube H and the bulk inlet velocity U_b. The cube sides measure $1H$, the cube is centred in the z direction and placed at $3H$ from the inlet. The inlet boundary condition is a laminar parabolic velocity profile; the outlet is fixed pressure, zero velocity gradient boundary; the spanwise boundaries are periodic; the top, bottom and cube walls are no-slip boundary conditions.

The base simulation domain is divided in two sub-domains that completely cover the vertical and spanwise directions (Fig. 2). The LB sub-domain starts at $x = 0$, ends at $x = 7H$ and is solved by the LB software GASCANS; the NS sub-domain starts at $x = 5H$, ends at $x = 10H$ as is solved by the `pisoFoam` NS solver of the OpenFOAM suite. The size of the overlap region is then $2H$ in the streamwise x direction and covers all the domain in the vertical and spanwise directions. This default size and position of the overlap region is arbitrary. We present a study of the effect on the results of various sizes and locations of the overlap region in section 5.2.3.

The boundary conditions for the LB sub-domain are forced equilibrium for the LB inlet and NS to LB boundary. For the LB inlet boundary the LB velocity is fixed to a fully develop laminar profile and for the NS to LB boundary the velocity is fixed to the velocity interpolated from the NS sub-domain. The density for both boundaries is set to 1 and the particle distribution functions are set to equilibrium (eq. 5). The spanwise boundaries are periodic; and the top, bottom and cube boundaries are set to no-slip using a half-way bounce-back boundary condition [8].

The boundary condition for the LB to NS boundary of the NS sub-domain is fixed velocity and zero pressure gradient, where the velocity is interpolated from the LB sub-domain. Moreover, the interpolated LB velocity is corrected to ensure that the NS sub-domain mass flow is the same as the LB sub-domain mass flow. The NS outlet boundary is set to fixed pressure and zero gradient velocity, the top and bottom walls are set to no-slip, and the spanwise boundaries are periodic.

The LB sub-domain is discretised in a a constant cubic mesh with cell size dx_{LB}, while the NS sub-domain is discretised in a structured mesh refined towards the bottom wall and the cube walls (see Fig. 3. The cell size for the NS sub-domain dx_{NS} refers to the height of the cells next to the bottom boundary. The NS time step yields a CFL number less than 1 and LB time step yields a CFL number less than 0.1, which are the recommended values for both solvers. The LB sub-domain runs 16 time steps for each NS time step. The information is coupled at the end of every NS time step (see Fig. 1).

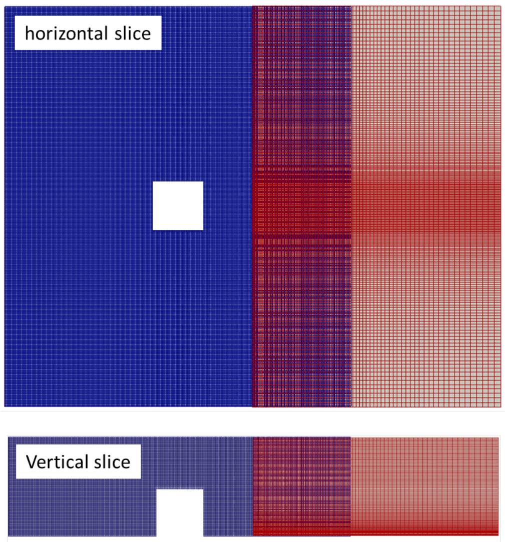


Figure 3: Example of a mesh for the current NSLB test case. The blue mesh (left) is the LB sub-domain, while the red mesh (right) is the NS sub-domain.

5.2. Results and discussion

5.2.1. Validation

The first step is to validate the results of the NS and LB models separately against the direct NS (DNS) simulation by Hwang and Yang [13]. The mesh used for both models covers the whole domain in Fig. 2 and the two solvers run separately. The mesh used for the NS simulation is refined towards the bottom wall and the cube, with a bottom cell height equal to $0.006H$. The LB mesh is an uniform mesh with cubic cells of side equal to $H/80 = 0.0125H$, which is the finest LB mesh we could afford to run with the available hardware. Hwang and Yang [13] do not present velocity results to validate the NS and LB solvers with. We then use their results for the wall shear stress at the bottom wall along a streamwise line centred in the spanwise direction (Fig. **??**). Both the LB and the NS results match the Hwang and Yang [13] data to an adequate degree. The LB solver slightly over estimates the minimum shear and under estimates the maximum shear. This results are to be expected since the shear stress is very sensitive to the size of the first cell and we only used a first order derivative scheme to calculate the velocity shear at the wall. In this case the LB solver is at a clear disadvantage; due to its requirement to use a constant cell size, decreasing the cell size for the wall adjacent cells comes with a prohibitive computational cost.

The main intended application of the NSLB solver is to predict the velocity and pressure near the ground, so we chose a line along the streamwise direction at heigh $y = 0.04H$ and centred in the spanwise direction to test mesh convergence for both the single NS and the single LB solvers. Fig. 5b shows the root mean squared (rms) difference between each solver simulation and the maximum resolution used, $dy = 0.003H$ for the NS solver and $dy = 1/80H$ for the LB solver. The consecutive points left of the maximum resolution one are the result of doubling the size of all the cells in all

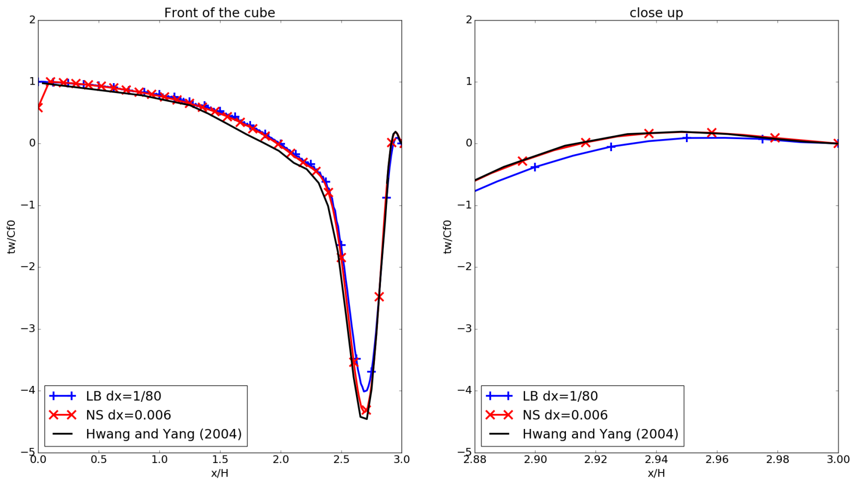


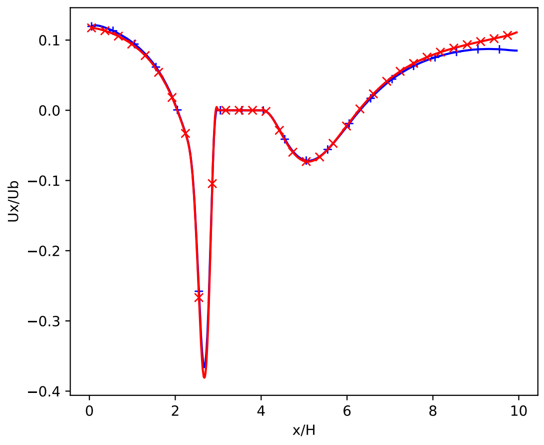
Figure 4: Normalised skin friction at the lower wall and centreline z = 0 in front of the cube (left) and in a close up in front of the cube (right) for LB simulation (blue crosses), NS simulation (red crosses) and Hwang and Yang [13] (black line). Cf0 is the shear stress at the inlet.

directions for both the NS and the LB simulations. Both solvers present mesh convergence. It can also be derived that the order of the NS solver is 2.029 and the order of the LB solver is 1.17.

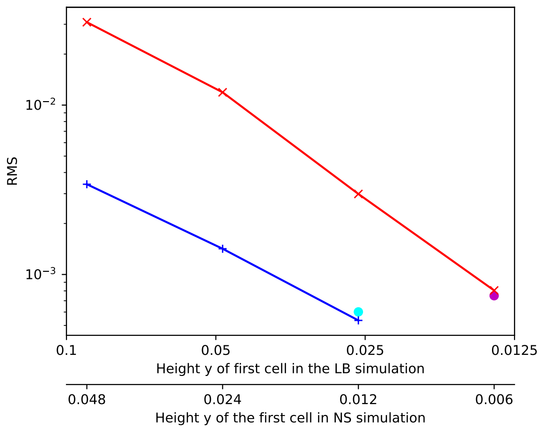
For the NS simulations, the rms of the velocity difference between a first cell height $dy = 0.003H$ and a first cell height $dy = 0.006H$ is less than 10^{-3}, so we consider the results from $dy = 0.006$ to be mesh converged. We use $dy = 0.006$ for the NSLB base case and for the overlap region tests and we use $dy = 0.003$ as a reference case for the accuracy and performance analysis (Fig. 5b and Fig. 7). For the LB simulations, the rms of the velocity difference between a cell size $dy = 1/80H$ and $dy = 1/40H$ is also below 10^{-3}, so we consider the results from $dy = 1/40H$ to be mesh converged. We use $dy = 1/40H$ for the NSLB base case and for the overlap region tests and we use $dy = 1/80H$ as a reference case for the accuracy and performance analysis.

The results obtained using a cubic uniform mesh with cell size $dx = 1/40H$ for the LB solver and a structured mesh (see red mesh in Fig. 3) with first cell height $dy = 0.006H$ for the NS solver are equivalent along the studied line (see Fig. 5a) with only discrepancies near the outlet due to the different outlet boundary conditions used in the LB and NS solvers. Note that for the NSLB simulations the outlet of the LB sub-domain (NS to LB boundary), is set to a forced equilibrium boundary condition, not to the zero gradient boundary condition shown in Fig. 5a. Thus the discrepancy in boundary conditions is not present in the NSLB test cases. Moreover, the interest region used to calculate the accuracy of the NSLB solver ends at $x = 7H$ which is not significantly affected by the different outlet boundary conditions in the single solver test cases.

The second step is to run the NSLB solver with the arbitrarly defined size and position of the overlap region. We decided to start the NS sub-domain at $x = 6$ to observe the effect of situating the LB to NS boundary in the centre of the recirculation zone downstream the cube. We deemed sufficient for the overlap region to be $2H$ long in the streamwise direction to minimise the effect of the NS to LB inteface on the LB to NS interface. We calculated the rms difference between the streamwise velocity along the line $y = 0.04$ in the LB sub-domain and the reference single LB domain ($dy = 1/80H$). This rms has been calculated from $x = 0.05$ to $x = 5$, which is half of the overlap region and shown as a cyan dot in Fig. 5b. The rms for the remainder of the studied line is shown as a magenta dot in Fig. **??** and has been calculated between $x = 5.025$ and $x = 9.95$ as the rms difference between the streamwise velocity in the NS sub-domain and the streamwise velocity for the reference single NS



(a)



(b)

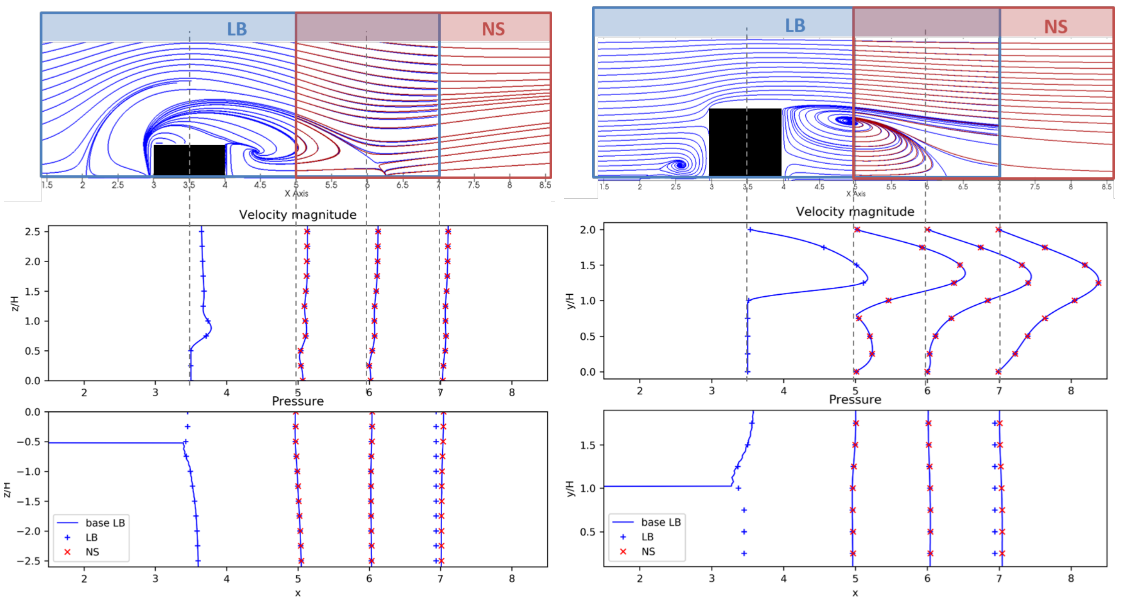
Figure 5: Mesh convergence plots for velocity for both LB and NS base simulations. (a) Streamwise velocity in the x direction at $y = 0.04H$ and $z = 0$ at the mesh converged grid resolution for the LB (blue) and the NS (red). (b) root mean squared difference between the results at the sampled line with each mesh resolution and the results at the sampled line with the finest resolution for each solver.The cyan and magenta dots correspond to the rms of the LB sub-domain (cyan) and the NS sub-domain (magenta) for the base NSLB simulation compared to their respective single solver simulation at the finest resolution. The finest resolution is $dy = 0.0125H$ for LB and $dy = 0.003H$ for NS

domain with $dy = 0.003$. Both values are close to their respective single LB and single NS simulations, with a difference of 11% for the LB subdomain and a difference of -7% for the NS subdomain. Both values are still below 10^{-3}, so we consider the NSLB mesh converged.

Fig. 6 shows velocity streamlines and velocity magnitude and pressure around the cube for both the NS and LB subdomains compared to the single LB simulation at $dx = 1/40H$. The streamlines are continuous along the whole domain for both the NS and the LB zones. However, there is a small discrepancy between the NS and LB stream lines at the NS to LB boundary. The NS and LB subdomains present the same velocity magnitude results in the overlap region, and both match the single LB velocity values across the whole domain. The maximum discrepancy between the NS sub-domain and the LB sub-domain can be observed in the pressure at the NS to LB boundary: the LB results show the constant pressure boundary condition, while the single LB and the NS sub-domain results show a varying pressure. The pressure results elsewhere are consistent for all the solvers. The boundary condition in the NS to LB boundary of the LB sub-domain is forced equilibrium. This boundary condition assumes that the particle distribution functions in the cells of the NS to LB boundary are in the equilibrium state corresponding to the interpolated NS velocity and a constant density; thus the non-equilibrium information carried by the NS velocity is lost. This loss of information and the constant LB density could explain the discrepancies in the streamlines in Fig. 6; the velocity at the NS to LB boundary matches because the LB macroscopic velocity is set to the NS velocity.

5.2.2. Computer resources

Comparing the computational resources used for NS, LB, and NSLB methods requires a bit of thought. The NS and LB methods solve different equations with completely different algorithms, meshing strategies and cell size and time step restrictions. Thus NS and LB need different time steps and resolution to obtain comparable accuracy (see Fig. 5b). A raw comparison of the time it takes each solver to calculate a time step or the number of cells solved per second is not possible. Moreover, the NS and LB solvers run in completely different architectures, so comparing the CPU hours or GPU hours for both of them is also not possible. Instead, we compare the performance of the the single LB, single NS and NSLB methods by plotting the time taken to simulate 1 second of physical time .vs. the rms between the results and the base case for each solver (Fig 7). For the NSLB simulations we use



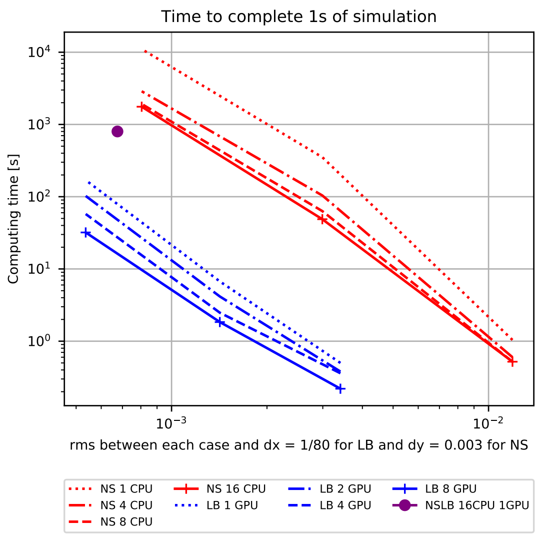
(a) (b)

Figure 6: Streamlines on the horizontal plane y=0.04 (top-left) and the central vertical plane z = 0 (top-right) for NSLB laminar base simulation, LB streamlines (blue), NS streamlines (red). The left edge of the NS sub-domain (red vertical line) is the LB to NS boundary, and the right edge of the LB sub-domain (blue vertical line) is the NS to LB boundary. Velocity magnitude and pressure lines at $x = 3.5, 5, 6, 7$ in each plane compared with the results of a single LB model (bottom)

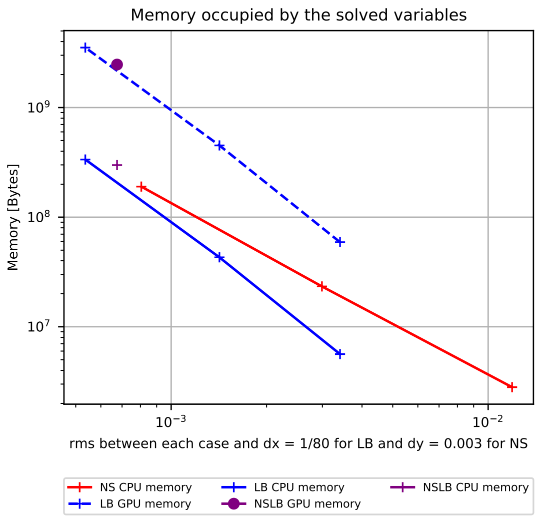
the average between the rms for the NS subdomain and the rms for the LB subdomain.

This performance has been evaluated using a single node computer with Intel Xeo E5-2660 CPU card and 8 Nvidia GeForce GTX 180 Ti GPUs. The NS sub-domain is decomposed by splitting the domain in equal parts in the y and z directions and run in parallel using MPI. Fig. 7a shows the computer time needed to solve one second of physical time for the single NS simulation with different number of CPU, the single LB simulation with different number of CPU and the NSLB simulation with the combination of CPU and GPU that resulted in a lower computational time. The time taken by both methods increases as the rms decreases; however, the single LB computer time is smaller than the NS computer time. For the largest rms simulations (rms order 10^{-2} for single NS and 10^{-3} for single LB), the single LB simulation witH 8GPU is approximately 2 times faster than the single NS simulation with 16CPU; for the intermediate rms (rms of order 10^{-3} for both single LB and NS) the single NS simulation with 16CPU is approximately 26 times faster than the single NS simulation with 16CPU; and for the smallest rms (rms of order 10^{-4} for both single NS and single LB) the single LB simulation wiht 8GPU is approximately 55 times faster than the single NS simulation with 16CPU. The NSLB simulation (rms of order 10^{-4}), run with 1GPU and 16 CPUs, is approximately 2 times faster than the single NS simulation with 16CPUs but still 25 times slower than the single LB simulation with 8GPU and 5 times slower than the single LB simulation with 1GPU.

Fig. 7b shows the CPU and GPU memory occupied by the variables stored in the meshes of each simulation. The memory is counted as 8 Bytes for each solved variable stored in the computational meshes. The NS solver stores the three components of velocity and pressure on each cell in CPU memory. The LB solver stores the three macroscopic velocity components and density in CPU memory and the three macroscopic velocity components, pressure and two copies of the 19 particle distribution functions in GPU memory. As expected, the memory occupied by both solvers increases as the rms decreases. However, the memory required by the single LB simulation is significantly larger than the memory required by the single NS simulation with a rms of the same order of magnitude. The NSLB



(a)



(b)

Figure 7: Time and memory consumption by single LB and single NS simulations with different resolutions (different rms respect reference simulations) and by the NSLB method. (a) Time spend to run one physical second of simulation. The crosses mark the rms of the performed simulations. (b) CPU and GPU memory.

GPU memory occupancy is 1.4 times lower than the memory occupied by the single LB simulation with a rms of 10^{-4}, the CPU memory is 1.1 times lower.

The NSLB method runs the NS sub-domain and the LB sub-domain concurrently and sets synchronisation barriers at the points where information needs to be send to and received from the coupled solver. Table 1 shows the ratio of the total running time used for each sub-domain to execute its core solver, receive and send data. The receiving time includes the time the solver is stopped waiting at the synchronisation barrier. Table 1 shows that the time both sub-domains spend sending data is negligible. However, the LB sub-domain spends 94% of its running time waiting to receive and receiving data from the NS sub-domain and only 19% of the time running its core solver; while the NS sub-domain spends 94% of its time running its core solver and only 0.87% of its time waiting to receive and receiving data from the LB sub-domain. The effect of increasing the number of CPUs used by the NS sub-domain and the number of GPUs used by the LB sub-domain is estimated to be low. Fig. 7a shows that the gain in computer time at 16CPU is already reduced respect 8CPU. Moreover, since the LB sub-domain spends 78.9% of its time waiting for NS sub-domain to transfer the results, using more GPUs will not decrease the overall time of the simulation.

	NS sub-domain	**LB sub-domain**
Core solver	0.94	0.19
Receive data	0.0087	0.789
Send data	3.8×10^{-5}	9.6×10^{-4}

Table 1: Time spend for the NS sub-domain and the LB sub-domain on the core solver, data receiving and data sending for the NSLB method simulation run with 1GPU and 16CPU. The times are normalised to the total running time of the simulation.

For the current test case and configuration, the NS sub-domain is a bottleneck in the simulation, thus reducing the potential performance gain of the NSLB method. The NS sub-domain bottle neck can also be viewed as an opportunity for the LB sub-domain to execute a much larger domain or to run at a higher resolution while minimising the impact on the overall running time of the NSLB method. It is then anticipated that the performance benefits of the NSLB method will increase with the difference in the resolved flow scales by the NS and LB sub-domains.

Finally, note other factors like the number of cells in each sub-domain and the characteristics of the mesh refinement in the NS sub-domain could also affect the speed of the simulation.

5.2.3. Overlap region

Finally, we present a study of the influence of the size and position of the overlap region over the accuracy of the results. To do so, we ran the NSLB simulation with a range of overlap regions starting at $x_s = 1.75, 2.6, 3.0, 3.5, 4.4, 5.25, 6.15$ with the following sizes $o_w = 0.05, 0.15, 0.375, 0.8, 1.55, 2.3$ in the streamwise direction. The start of the overlap region is the position of the LB to NS boundary and the size of the overlap region is the distance between the LB to NS boundary and the NS to LB boundary (Fig. 2).

Fig. 8 shows the variation in the velocity error of the NSLB method respect the velocity for the single LB simulation in the interest region for the different positions and sides of the overlap region. Fig. 8 presents a higher density of points in regions where the rms difference presents a higher gradient. The interest region is defined as a box next to the bottom wall of size 5H $\times$ 1.5H $\times$ 4H in the streamwise, vertial and spanwise direction respectively, starting at 2H in the x direction and centred in z. The error is calculated as the root mean square of the difference between the sub-domain result and the single LB simulation result, and shown respect the bulk velocity U_b. The LB sub-domain results are compared in the first half of the overlap region and the NS sub-domain results are compared in the second half of the overlap region.

The base test case has an overlap region starting at $x = 5$ and ending at $x = 7$; which situates it in the lowest error region in Fig. 8 with an error below 1% of the bulk velocity. The error in the velocity diminishes as the size of the overlap region increases for all starting points except $x_s = 2.6$; all starting points except $x_s = 2.6$ and $x_s = 5.25$ present an error below 10% for sizes $o_w \geq 0.8$. Regarding the minimum size of the overlap region, all the tested starting positions except $x_s = 1.75$ and $x_s = 6.15$ produce unstable results for an overlap region size $o_w = 0.05$. The unstable cases present oscillations in the velocity and pressure results that amplify as time advances. Another special case is $x_s = 3.5$. This case is unstable at $o_w = 0.05$ but presents a lower error at $o_w = 0.15$ than its neighbouring overlap region starting points. Finally, note that the combination $x_s = 2.6$, $o_w = 1.55, 2.3$ show a higher error than the error for smaller sizes, which contradicts the trend of the other tested starting positions.

We selected two of the most interesting overlap sizes and positions to study in more detail:

- Test A: $x_s = 3.5$, $o_w = 0.15$; smallest overlap region with an error below 10 %. Its error value is 8% respect the bulk velocity.
- Test B: $x_s = 2.6$, $o_w = 2.3$; largest overlap region with an error above 1%. Its error value is 1.2% respect the bulk velocity.

Test A and Test B are in the same colour band in Fig. 8 but Test B is near the lower error band and test A is near the higher error band. The results for both cases (Fig. 9) are significantly different. The streamlines for Test A (Fig. 9a) are continuous and match through the overlap region; however the difference in the pressure and velocity magnitude between the NS sub-domain results and the single LB simulation are significant; the shape of the wake also presents differences with the base NSLB simulation (Fig. 6). The streamlines for Test B (Fig. 9b) are also continuous through the overlap region but there are some discrepancies between the NS sub-domain and LB sub-domain streamlines in front of the cube near its front face. The streamlines on the wake are also slightly different but the differences are not significant. The velocity magnitude results for the NS sub-domain, the LB sub-domain and the single LB simulation match and the pressure results presents only a slight variation in $x = 2.6$.

The size and position of the overlap region affects the accuracy of the results. The LB to NS boundary in test B cuts through the horseshoe vortex in front of the cube and its NS to LB boundary cuts through the centre of the recirculation in the wake of the cube. The fact that the two boundaries

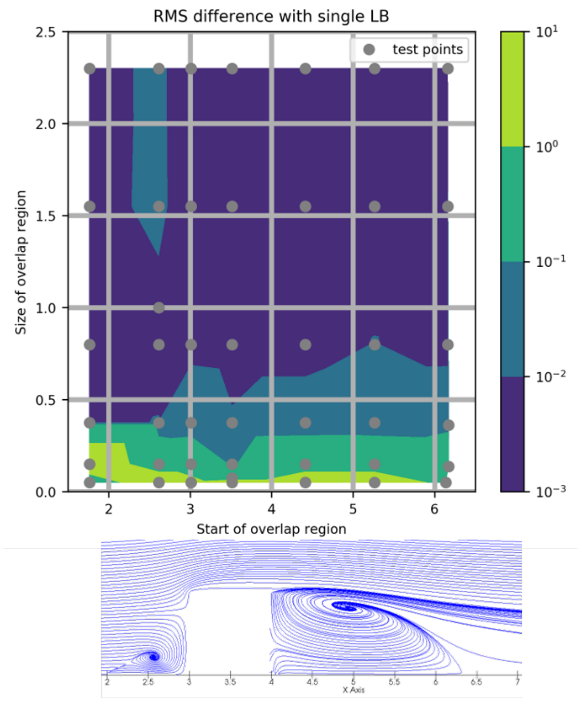


Figure 8: 2D plot of the root mean squared difference between the NSLB and single LB streamwise velocity in the interest region referred to the inlet bulk velocity. The interest region is shown at the bottom for reference; its size in the spanwise direction is 4 centred on the cube. The computed simulations are shown as grey dots. Light green colour corresponds to an error above 100% or an unstable simulation.

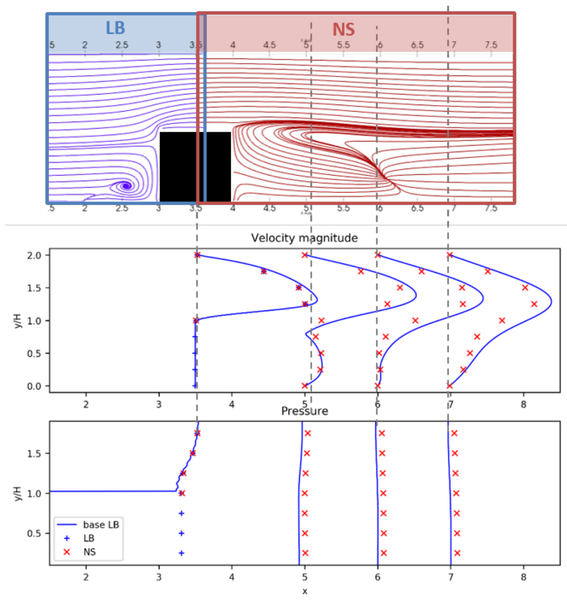
cut through complex flow patterns might explain why the error for this test case is higher than the error for its surrounding tests. However, Test B presents an error just slightly over 1% and the differences between the NSLB results and the single LB results are not significant. Test B results suggest that, even if stable at that position, an overlap region with a width of $o_w = 0.15$ is too short to properly reproduce the results of the single LB simulation.

6. Conclusions and outlook

We presented a LB solver two-way coupled with a NS solver able to accurately model three-dimensional laminar flow around a bluff object. The NSLB solver couples a sub-domain solved by the NS solver OpenFOAM [7] run using multi-CPU with a sub-domain solved by the GPU-accelerated LB solver GASCANS [9].

We have tested the coupling of the two solvers by reproducing the results of Huang and Yang [13] for laminar flow around a wall mounted cube in a channel at $Re_H = 150$. The root mean squared error on the streamwise velocity and pressure in the interest region compared to the single LB results is below 1% of the bulk velocity for overlap region sizes above $0.8H$.

The computational efficiency of the NSLB model was tested by running the Huang and Yang [13] test case with mesh converged resolution using single NS, single LB and NSLB with 16 cores of an Intel Xeo E5-2660 CPU and 1 Nvidia GTX 1080 Ti GPU. With the presented computational resources and case configuration, the NSLB model is approximately 2 times faster than single NS while consuming 1.4 times less GPU memory than the single LB simulation. Thus, the NSLB model presents potential to run simulations not available for the single LB solver due to the limited memory available in the GPU card and too time consuming to run with single NS solver due to the limited CPU resources. However,


(a) Test A

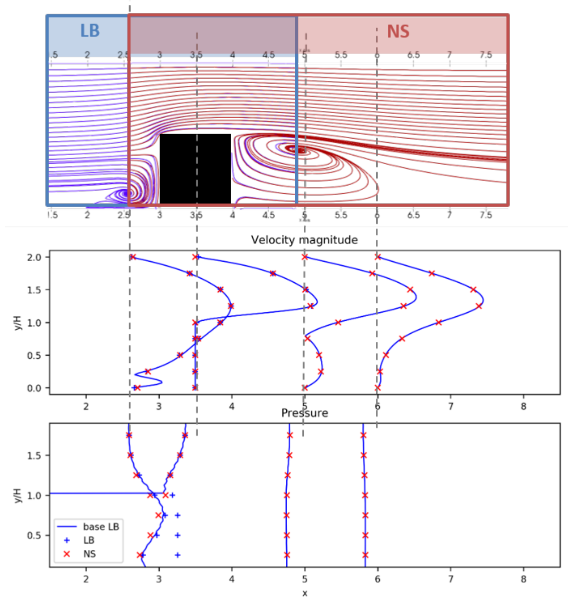

(b) Test B

Figure 9: Streamlines on the vertical plane $z = 0$ (top) for the NSLB simulation, LB sub-domain (blue) and NS sub-domain (red). Velocity magnitude and pressure for the NS sub-domain (red crosses), the LB sub-domain (blue crosses) and the single LB simulation (blue lines) at 4 positions in x. (a) overlap region starting at $x = 3.5$, ending at $x = 3.65$; sampled lines positions $x = 3.5, 5, 6, 7$. (b) overlap region starting at $x = 2.6$, ending at $x = 4.9$; sampled lines positions $x = 2.6, 3.5, 5, 6$.

the NS sub-domain constitutes a bottleneck in our NSLB solver configuration; then we anticipate that the performance benefits of the NSLB method could increase if the LB sub-domains resolves smaller scales, and thus more cells, than the NS sub-domain.

The size and position of the overlap region significantly affects the accuracy of the NSLB solver. An increase in the size of the overlap region decreases the error. The complexity of the flow at the coupled boundaries also affects the accuracy of the results, more complex flow through this boundary results in less accuracy. Finally the lowest error position in the vicinity of the cube is in the middle of it but the overlap region has to be over 0.15 long to produce accurate results. The main source of inaccuracy in the coupling is the NS to LB boundary in the LB sub-domain and it might be due to the NS non-equilibrium part of the NS velocity, which is lost due to the forced equilibrium boundary condition imposed in the NS to LB boundary.

Future work will entail modelling turbulent flows with the two-way coupled NSLB solver. Both, with an overlap region that covers the vertical and spanwise sides of the domain and with the LB sub-domain embedded in the NS sub-domain.

Acknowledgements This work has been supported primarily by the Samsug GRO programme with collaboration from The University of Manchester.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship and/or publica- tion of this article.

References

[1] E. Fattahi, C. Waluga, B. Wohlmuth, U. Rüde, M. Manhart, R. Helmig, Lattice Boltzmann methods in porous media simulations: From laminar to turbulent flow, Computers and Fluids 140 (2016) 247–259. doi:10.1016/j.compfluid.2016.10.007.

[2] A. Mivehchi, J. Harris, S. Grilli, J. Dahl, C. O'Reilly, K. Kuznetsov, C. Janssen, A hybrid solver based on efficient BEM-potential and LBM-NS models: Recent BEM developments and applications to naval hydrodynamics, Proceedings of the International Offshore and Polar Engineering Conference (2017) 713–720.

[3] P. Neumann, On transient hybrid Lattice Boltzmann–Navier-Stokes flow simulations, Journal of Computational Science 17 (2016) 482–490.
URL `http://dx.doi.org/10.1016/j.jocs.2016.02.003`

[4] Z. X. Tong, Y. L. He, A unified coupling scheme between lattice Boltzmann method and finite volume method for unsteady fluid flow and heat transfer, International Journal of Heat and Mass Transfer 80 (2015) 812–824.
URL `http://dx.doi.org/10.1016/j.ijheatmasstransfer.2014.09.067`

[5] A. C. Velivelli, K. M. Bryden, Domain decomposition based coupling between the lattice Boltzmann method and traditional CFD methods - Part II: Numerical solution to the backward facing step flow, Advances in Engineering Software 82 (2015) 65–74. doi:10.1016/j.advengsoft.2014.11.006.
URL `http://dx.doi.org/10.1016/j.advengsoft.2014.11.006`

[6] H. K. Versteeg, W. Malalasekera, An Introduction to Computational Fluid Dynamics, 2nd Edition, Pearson Prentice Hall, 2007.

[7] The OpenFOAM Foundation, Openfoam.
URL `https://openfoam.org/version/4-0/`

[8] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, E. M. Viggen, The Lattice Boltzmann Method Principles and Practice, Springer, 2017. doi:10.1007/978-3-319-44649-3.

[9] M. Camps Santasmasas, A. R. G. Harwood, S. Fan, B. Owen, J. O'Connor, A. Revell, GPU-Accelerated Solver for Coupled Approaches to Navier-Stokes (GASCANS), Computer Physics Communications In preparation.

[10] A. Atanasov, B. Uekermann, P. Neumann, Anderson Accelerated Coupling of Lattice Boltzmann and Navier–Stokes Solvers for Parallel Applications, Computation 4 (4) (2016) 38–57. doi:10.3390/computation4040038.

[11] H. J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, preCICE – A fully parallel library for multi-physics surface coupling, Computers and Fluids 141 (2016) 250–258.
URL `http://dx.doi.org/10.1016/j.compfluid.2016.04.003`

[12] G. Chourdakis, A general OpenFOAM adapter for the coupling library preCICE (December) (2017).

[13] J. Y. Hwang, K. S. Yang, Numerical study of vortical structures around a wall-mounted cubic obstacle in channel flow, Physics of Fluids 16 (2004) 2382–2394.

4.3.2 Paper II - Synthetic eddy method applied to the lattice Boltzmann model

The following journal paper presents a lattice Boltzmann model embedded in a pre-calculated RANS simulation. The free stream turbulence from the RANS results is introduced into the lattice Boltzmann simulation via a synthetic eddy method at the lattice Boltzmann inlet. The paper requires some revision before submission to the *Journal of Wind Engineering and Industrial Aerodynamics*, which is the journal we intend to submit this paper for.

Synthetic eddy method applied to lattice Boltzmann for wind around a rectangular prism building.

Marta Camps Santasmasas[a], Xutong Zhang[a], Alistair Revell[a]

[a]*School of Mechanical, Aerospace and Civil Engineering, University of Manchester, UK*

Abstract

Modelling urban wind flow resolving turbulence is computationally expensive and often requires computational resources that are out of reach for engineering applications. One of the main issues is that the computational domain required is significantly larger than the region of interest. We present a lattice Boltzmann large eddy simulation (LB-LES) solver that can be embedded in a pre-calculated RANS simulation. The LB-LES domain uses the mean RANS velocity as boundary condition for the top and the side boundaries; and incorporates the RANS turbulence using a synthetic eddy method at the lattice Boltzmann inlet. We tested the SEM implementation in a $Re_\tau = 395$ channel flow and we used the embedded LB-LES domain to model an atmospheric boundary layer flow around a rectangular building at $Re_H = 47893$, where H is the height of the building. The SEM boundary incorporates the mean turbulence from the RANS data into the LB-LES resolved velocity, enabling the LB-LES model to capture the physics of the flow correctly and show good agreement with the experimental results. Moreover, we ran the embedded LB-LES domain using a single Nvidia V100 graphics card and 8CPUs. These results show the potential of the SEM LB-LES solver to run high accuracy turbulent wind engineering flows with consumer level computational resources.

Keywords: Hybrid RANS/LES, Embedded LES, Turbulence, Industrial CFD, lattice Boltzmann, GPU.

1. Introduction

The most used Computational fluid dynamics (CFD) models in wind engineering solve the Reynolds averaged Navier-Stokes (RANS) equations (see f e. Toparlar et al. [1]). RANS models provide mean flow data at a low computational cost. However, RANS models do not provide information on the time dependent fluctuating velocity which is relevant for contaminant dispersion and building structural analysis studies. On the other hand, lattice Boltzmann solvers are used to model turbulent urban wind flow (see f e. Merlier et al. [2] and Lenz et al. [3] and Jacob and Sagaut [4]). An important advantage of lattice Boltzmann methods over Navier-Stokes methods is lattice Boltzman simple and local algorithm, which is efficient when implemented in massive parallel architectures like graphic processing units (GPUs). GPU cards are common in most personal computers and are not used by traditional RANS models. However, the high memory consumption of the lattice Boltzamann method combined with the restricted memory available in GPU cards limits the accuracy / domain size that is practical to model with lattice Boltzann using a single node. For example Onodera et al. [5] modelled a 10km × 10km area of metropolitan Tokyo at 1m resolution using lattice Boltzmann, which required 4032 GPUs.

The main challenge on the domain size for urban wind flows is that the interest region usually lacks defined boundaries apart from the ground and the modelled urban geometry. In order to run the simulation, artificial boundaries have to be created and the wind profile there estimated using profiles that are not representative of the wind in that region and usually only contain mean flow data. To paliate the inaccuracies caused by the boundary conditions, the boundaries are placed as far from the interest region as possible, thus increasing the computational cost of the simulation.

One way to obtain more accurate boundary conditions is to embed a turbulence resolving large eddy simulation (LES) model covering the interest region inside a less computationally expensive mean flow resolving Reynolds-averaged Navier-Stokes (RANS) model. The boundaries of the underlying RANS simulation are set to the estimated wind profiles, while the boundaries of the LES are set to the results of the underlying simulation as shown in Fig. 4. Mathey and Cokljat [6] embed a LES domain inside a pre-calculated RANS simulation to obtain more accurate results in a region of high flow complexity around the Ahmed body simplified car geometry; the LES nested region uses the results of the RANS simulation as boundary conditions. Embedded LES techniques follow the same principle with the difference that the RANS and embedded LES domains are run concurrently (see f e. [7]). Jadidi and Bazdidi-Tehrani [8] and Wijesooriya et al. [9] use embeded LES to model wind flow around an isolated building. Mathey and Cokljat [6] report pressure coefficient C_p in the LES nested region closer to the experimental data than the ones obtained by the RANS simulation; Jadidi and Bazdidi-Tehrani [8] report a similar distribution of time-averaged pollutant concentration for both the embedded LES and LES. Wijesooriya et al. [9] embedded LES results comparable with full LES results using a maximum of 54% of the computational resources of the full LES. Similarly, Jadidi and Bazdidi-Tehrani [8] report that the CPU time to complete the embedded LES is about 49 % lower than that of the full LES.

LES models require the instantaneous wind velocity to be prescribed at their inlet, while RANS models only provide mean flow data. Mathey and Cokljat [6], Jadidi and Bazdidi-Tehrani [8] and Wijesooriya et al. [9] use the vortex method by Sergent [10] to generate the instantaneous velocity at the LES inlet from RANS mean flow. Vortex methods are only one of the synthetic turbulence methods that can be applied at the LES inlet [11]. Both, Poletto et al. [12] divergence free synthetic eddy method (DFSEM) and Skillen et al. [11] improved synthetic eddy method inlet boundary conditions yield improved results over the vortex method in channel flows. Besides, Millar et al. [13] implements DFSEM as inlet boundary condition in an urban wind flow setting, reporting good agreement with experimental data.

A common characteristic of the reviewed embedded LES models is the use of Navier-Stokes based solvers run on CPU architectures for both the RANS and LES domains. We present a lattice Boltzmann LES (LB-LES) simulation embedded in a pre-computed RANS domain. The instantaneous velocity at the lattice Boltzmann LES inlet is obtained using Skillen et al. [11] synthetic eddy method (SEM). We first test the implementation of SEM into LB-LES in a $Re_\tau = 395$ channel flow; then we model the wind tunnel test case by Meng and Hibi [14], which is one of the validation benchmark cases in the Guidebook for CFD Predictions of Urban Wind Environment by the Architectural Institute of Japan. Meng and Hibi [14] provide wind tunnel experimental data of wind flow around a rectangular building at $Re_H = 47893$, where H is the height of the building. All the LB-LES in the present paper are run in single node using a single GPU to demonstrate the potential of running engineering relevant turbulent flows with a consumer grade computer.

The results from the tests show that the lattice Botlzmann implementation of SEM is able to obtain results comparable to the ones obtained by Skillen et al [11]. The embedded LES domain with SEM inlet boundary condition is able to capture the physics of the flow correctly and yields the closest results to the experimental data amongst all the tested methods.

2. LES lattice Boltzmann: LES LB-BGK

The LES lattice Boltzmann method implemented in this paper uses a 3DQ19 velocity discretisation with BGK collision operator and implements the Hou et al. [15] LES Smagorinsky turbulence model.

The lattice Boltzmann equations solve for the particle distribution functions f_α; which model the probability of a group of fluid particles to have a certain velocity at a certain position and time. The particle velocity space is discretised in a number of velocities $\mathbf{c}_\alpha$ and corresponding weights w_α defined by the lattice model. The lattice Boltzmann method in this paper implements a 3DQ19 lattice model, which contains the following particle velocities their weights:

$$\vec{c}_\alpha = \begin{cases} 0, & \alpha = 0, \\ (\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1) & \alpha = 1 \quad to \quad 6, \\ (\pm 1, \pm 1, 0), (\pm 1, \pm 0, \pm 1), (0, \pm 1, \pm 1) & \alpha = 7 \quad to \quad 18. \end{cases} \tag{1}$$

$$\vec{w}_\alpha = \begin{cases} 1/3, & \alpha = 0, \\ 1/18 & \alpha = 1 \quad to \quad 6, \\ 1/36 & \alpha = 7 \quad to \quad 18. \end{cases} \tag{2}$$

$$\tag{3}$$

The lattice Boltmzann LES model solves for the filtered particle distribution functions $\bar{f}_\alpha$. $\bar{f}_\alpha$ only represents motion at scales larger than the filter width Δ. The lattice Boltzmann equations for the filtered particle distribution functions are:

$$\bar{f}_\alpha(\mathbf{x_0} + \Delta x, t_0 + \Delta t) - \bar{f}_\alpha(\mathbf{x_0}, t_0) = \Delta t \bar{\Omega}(\mathbf{x_0}, t_0) \qquad \alpha = 0:18 \tag{4}$$

$$\tag{5}$$

where Δt is the time step and Δx is the spacial step in x, y and z in lattice units. Lattice units measure space in cells and time in time steps; thus, for example, a velocity in lattice units of $u_x = 0.1$ cells/timeSteps indicates that the fluid covers 0.1 cells in one time step. Δx and Δt in lattice units are usually set to 1 to simplify the LB algorithm and will be omitted from the following LB equations. Note that this form of the lattice Boltzmann equations assumes that the cell size is the same in all directions and that all the cells are the same size. $\bar{\Omega}(\mathbf{x_0}, t_0)$ is the filtered collision operator, which for a BGK collision model is:

$$\bar{\Omega}_\alpha = \frac{f^{eq}(\bar{\rho}, \bar{u_i}) - \bar{f}_\alpha}{\bar{\tau}} \tag{6}$$

where $\bar{\rho}$ and $\bar{\mathbf{u}}$ are the filtered macroscopic pressure and velocity respectively and are obtained from the filtered particle distribution functions $\bar{f}_\alpha$ as

$$\bar{\rho} = \sum_\alpha \bar{f}_\alpha \tag{7}$$

$$\overline{\rho u_i} = \sum_\alpha \bar{f}_\alpha c_{\alpha i} \qquad i = 0:2 \tag{8}$$

$$\tag{9}$$

and f^{eq} is the equilibrium particle distribution function for a 3DQ19 velocity model with the filtered velocity and density.

$$f^{eq}_\alpha = w_\alpha \rho \left(1 + \frac{u_i c_{i\alpha}}{c_s^2} + \frac{(u_i c_{i\alpha})^2}{2c_s^4} - \frac{u_i u_i}{2c_s^2}\right) \tag{10}$$

$c_s = \frac{1}{\sqrt{3}}$ is the lattice speed of sound.

Hou et al. [15] LES turbulence model introduces the modified relaxation time $\bar{\tau}$ to the BGK collision operator

$$\bar{\tau} = \tau + 3\nu_t \tag{11}$$

where $\tau = 3\nu\frac{\delta t}{\delta x^2} + 1/2$ is the BGK relaxation time; ν is the viscosity of the fluid in physical units, δx is the cell size in physical units and δt is the time step in physical units. ν_t is the Smagorinsky eddy viscosity and it is defined as:

$$\nu_t = C_s \Delta^2 |\bar{\mathbf{S}}| \tag{12}$$

$$\tag{13}$$

We set the LES filter width $\Delta = 1$ and the Smagorinsky constant $C_s = 0.01$. $|\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$ is the intensity of the strain rate of the filtered velocity

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \tag{14}$$

How et al. [15] LES implementation calculates $\bar{\tau}$ involving only local operations. The modified relaxation time is:

$$\bar{\tau} = \frac{1}{2\bar{\rho}}\left(\sqrt{\bar{\rho}^2\tau^2 + 18\sqrt{2}\bar{\rho}C_s\Delta^2 Q^{1/2}} - \tau\right) + \tau \tag{15}$$

where Q is the second variance of the filtered velocity stress tensor and can be approximated by [15]:

$$Q = \bar{\Pi}_{ij}^{(1)}\bar{\Pi}_{ij}^{(1)} \tag{16}$$

$$\bar{\Pi}_{ij}^{(1)} = \sum_{\alpha} c_{\alpha_i} c_{\alpha j} \bar{f}_{\alpha}^{neq} \tag{17}$$

where $\bar{f}_{\alpha}^{neq} = \bar{f}_{\alpha} - \bar{f}_{\alpha}^{eq}$ is the non-equilibrium distribution function.

All the operations in eq. 15 are local, thus the implementation of the LES Smagorinsky using eq. 15 does not significantly increase the computational cost of the lattice Boltzmann method in GPU. Finally, note that eq 15 depends on the filtered density $\bar{\rho}$; which we set to $\bar{\rho} = \rho_0 = 1$.

Note that we implemented an LES Smagorinsky model without special near wall treatment.

2.1. Lattice Boltzmann vs traditional CFD models

Navier-Stokes equations discretised using finite volumes and solved using algorithms derived from SIMPLE [16] and linear equation solvers are commonly used in engineering urban wind simulations [1].

The main strength of the presented form of lattice Boltzmann method over the widely used Navier-Stokes models is the simplicity of the lattice Boltzmann algorithm, which is explicit and only requires information from the current cell and its nearest neighbours. This simplicity and locality make the lattice Boltzmann method very efficient when implemented in massively parallel architectures like graphic processing units (GPUs). Another strength is that it models convection by transferring the particle distribution functions from one cell to the neighbouring cell. Since the position of the particles is not interpolated, the presented lattice Boltzmann method presents low numerical diffusion.

The main weakness of the presented form of the lattice Boltzmann method is that it assumes that all the cells are of size δx in all directions, which makes mesh refinement cumbersome to implement. Another disadvantage is that it is memory intensive, since it has to store the values of all the discretised particle distribution functions plus velocity and density at each cell. Finally, its algorithm is only conditionally stable and the cell size and time step are linked to the viscosity of the fluid, which limits the combination of δt and δx that produce accurate results.

3. Synthetic eddy method (SEM)

We use the synthetic eddy model by Skillen et. al [11] to generate instantaneous velocity data at the lattice Boltzmann inlet using mean flow statistics at each cell of the inlet. Concretely, SEM needs time averaged velocity $U_i(\mathbf{x},t)$, Reynolds stresses $R_{ij}(\mathbf{x},t)$ and a turbulent length scale

$$\sigma_i(\mathbf{x},t) = \frac{R_{ii}^{3/2}(\mathbf{x},t)}{\epsilon(\mathbf{x},t)} \tag{18}$$

where $\epsilon(\mathbf{x},t)$ is the mean turbulent kinetic energy dissipation rate at each cell of the inlet. $\sigma_i(\mathbf{x},t)$ determines the radius of the synthetic eddies in the position $\mathbf{x}$ in each direction. Two of the SEM configuration parameters are the minimum eddy size σ_i^{min} and the maximum eddy size σ_i^{max}, which limit the size of the synthetic eddies generated by the SEM. Fig. 1 shows an example of the eddy size in a $Re_\tau = 395$ channel and the effect of the maximum and minimum synthetic eddy size.

Skillen et al. [11] SEM first generates a fixed number of three dimensional synthetic eddies centred at random positions of the inlet plane. At each time step, the eddies are advected through the inlet by the bulk velocity of the prescribed mean flow. When an eddy no longer intersects the inlet, it is regenerated to a new random starting position, which is recomputed until the eddy lands in a region with low eddy density.

The instantaneous inlet velocity field $u_i(\mathbf{x},t)$ is generated in three steps:

1. Calculate the preliminary fluctuating velocity field $u_i^*(\mathbf{x},t)$ by adding the contribution of each synthetic eddy normalised by the running average of the eddy concentration (eq. 19)

$$u_i^*(\mathbf{x},t) = \frac{\sum_{e=1}^{N} \epsilon_i f(\mathbf{x}-\mathbf{x_e},\sigma_\mathbf{e})}{\sqrt{\left\langle \sum_{e=1}^{N} f^2(\mathbf{x}-\mathbf{x_e},\sigma_\mathbf{e}) \right\rangle^{AVG}}} \tag{19}$$

   where ϵ_i is an integer representing the sign of the eddy and is assigned a ± 1 value at random, $\mathbf{x}$ is the current inlet cell, $\mathbf{x_e}$ is the position of the centre of the current eddy, $\sigma_\mathbf{e}$ is the length scale of the current eddy, N is the number of eddies and f is a weight function based on the distance from the centre of the current eddy. Skillen, et al. [11] use a truncated Gaussian function with extents defined by the length scale of the current eddy $\sigma_\mathbf{e}$. Skillen, et al. [11] contains more information on the weight function f and the implementation of the running average of the eddy concentration ($\langle\rangle^{AVG}$).

2. Correct the preliminary fluctuating velocity u_i^* by multiplying it by the Cholesky decomposition of the inlet Reynolds stress tensor L_{ij} (eq. 20). This correction sets the mean value of u_i^* to zero and its second order statistics to the inlet Reynolds stresses.

$$L = \begin{bmatrix} \sqrt{R_{11}} & 0 & 0 \\ R_{21}/L_{11} & \sqrt{R_{22}-L_{21}^2} & 0 \\ R_{31}/L_{11} & (R_{32}-L_{21}L_{31})/L_{22} & \sqrt{R_{33}-L_{31}^2-L_{32}^2} \end{bmatrix} \tag{20}$$

3. Add the corrected preliminary fluctuating velocity to the inlet mean velocity (eq. 21)

$$u_i(\mathbf{x},t) = L_{ij}(\mathbf{x},t)u_j^*(\mathbf{x},t) + U_i(\mathbf{x},t) \tag{21}$$

3.1. SEM for lattice Boltzmann

The instantaneous inlet velocity at the LES lattice Boltzmann inlet is generated using the synthetic eddy method by Skillen et al. [11] as described above. The resulting velocity is then translated to lattice units and the particle distribution functions, or filtered particle distribution functions for the LES model, are obtained using the regularised velocity boundary condition by Latt et al. [17].

The main difference between the Navier-Stokes and lattice Boltzmann implementation of Skillen et al. SEM is the mesh. The constant cell size in lattice Boltzmann allowed us to optimise the calculation of the preliminary velocity field (eq. 19). This calculation is the most computationally expensive part of the SEM algorithm because it involves a nested search loop: the algorithm has to search for all the cells and then for all the eddies to determine which eddies contributes to each cell. Our optimisation reduces the number of operations in this step around 2 orders of magnitude, thus significantly reducing the computing time of the SEM boundary condition.

Another consequence of the constant cell size in the lattice Boltzmann mesh is that it does not allow for refinement next to the walls. The minimum synthetic eddy size is usually set to the cell size δx, thus the minimum eddy occupies 2 cells in each direction. In coarse meshes, the minimum eddy size σ_i^{min} might be larger than the smallest value of $\sigma_i(\mathbf{x}, t)$, thus not representing the smallest eddies next to the wall. Moreover, two cells in each direction might not be enough to correctly represent a synthetic eddy. In this paper we chose $\sigma_i^{min} = 4\delta x$ because it was the value that yield more accurate results at coarse resolutions. However, the effect of σ_i^{min} in the generated velocity needs to be studied further.

Finally, the present implementation of SEM runs in the CPU part of the lattice Boltzmann code and incorporates further code optimisations implemented by Fan et al. [18] including single node parallelisation using OpenMP.

4. Test case: turbulent channel flow

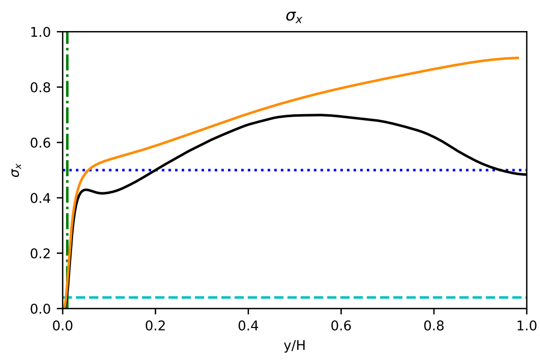
The aim of this section is to validate the implementation of Skillen et al. [11] SEM model in the lattice Boltzmann solver both with and without the LES Smagorinsky turbulence model. The chosen test case is the channel flow with a $Re_\tau = 395$ modelled by Kozuka et al. [19] using Navier-Stokes DNS. This case is the case `ch395_4th_D` of the Direct Numerical Simulation Data Base for Turbulent Channel with Heat Transfer from the Laboratory of Thermo-fluid dynamics from the Tokyo University of Science [1].

We first test the accuracy of the SEM implementation in lattice Boltzmann compared with Skillen's SEM Navier-Stokes implementation for both using the DNS mean flow to generate the inlet instantaneous velocity and using the mean flow data from a RANS EBRSM precursor simulation. Then, we test the effect of mesh resolution on the lattice Boltzmann results.

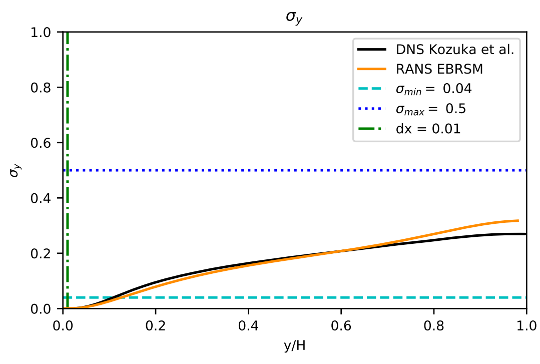
4.1. Case description

The test case is a $Re_{tau} = 395$ channel flow of dimensions 10H in x, 2H in y and 3.16H in z, where H is half the channel height. x is the streamwise direction, y is the vertical direction and z is the spanwise direction. The top and bottom walls implement a no-slip boundary condition via the half-way bounce back method. The outlet implements zero gradient for velocity and pressure via a extrapolated boundary condition, i e. the particle distribution functions entering the domain are copied from the cells parallel upstream of the outlet cells. For the lattice Boltzmann DNS (simulations with no turbulence modelling), the 10% of the domain next to the outlet implements a sponge layer in order to stabilise the simulation. The sponge layer increases the viscosity of the fluid in a cubic manner until the viscosity doubles at the outlet cells. The introduction of the LES Smagorinsky model stabilises the simulation enough to not need the sponge layer. All the simulations have constant size cubic cells.

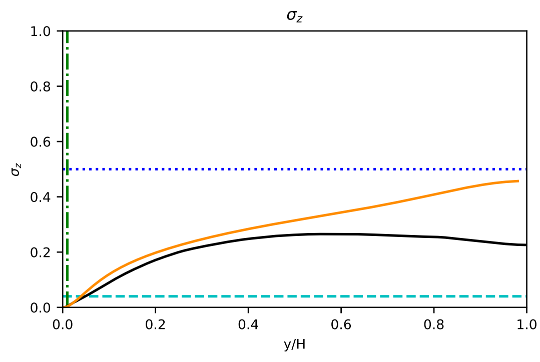
[1]https://www.rs.tus.ac.jp/t2lab/db/



(a)



(b)



(c)

Figure 1: Size of the SEM eddies σ_i calculated with the DNS data from Kozuka et al. (black) and the RANS EBRSM data from Mole (orange) together with the minimum (cyan dashed) and maximum (blue dots) eddy size in the lattice Boltzmann simulation. The position of the first cell is indicated with a dashed dot green line. (a) Streamwise component σ_x; (b) Vertical component σ_y; (c) Spanwise component σ_z.

The inlet boundary implements the SEM model described in section 3 with a maximum synthetic eddy size $\sigma_i^{max} = H/2$ and a minimum synthetic eddy size $\sigma_i^{min} = 4\delta x$, where δx is the cell size. We used two different sources for the mean velocity and Reynolds stress data input for SEM: the Navier-Stokes results for a periodic $Re_\tau = 395$ channel flow by Kozuka et al. [19] and the Reynolds averaged Navier-Stokes (RANS) results for a periodic $Re_\tau = 395$ channel flow. The RANS simulation implements an elliptic blending Reynolds stress model (EBRSM) turbulence model. Fig. 1 shows the effect of the limiters on the synthetic eddy size on the input data. The maximum effect of the minimum synthetic eddy size is in the vertical direction y, in which approximately the first 10 cells contain eddies larger than the ones specified by the inlet mean flow data. The cut on the maximum eddy size only affects the size of the synthetic eddies in the stream wise direction x, which limits the eddy size to its maximum value at $y = 0.2H$ for the DNS input data and $y = 0.04H$ for the RANS EBRSM input.

Table 1 shows the main characteristics of the channel flow simulations. The Smagorinsky constant for the LB LES is $Cs = 0.01$, which is commonly used for channel flow [20].

All the lattice Boltzmann simulations ran for 100 convective time units ($H/U_b = 100$) before averaging the results during a further 100 convective time units.

4.2. Results and discussion

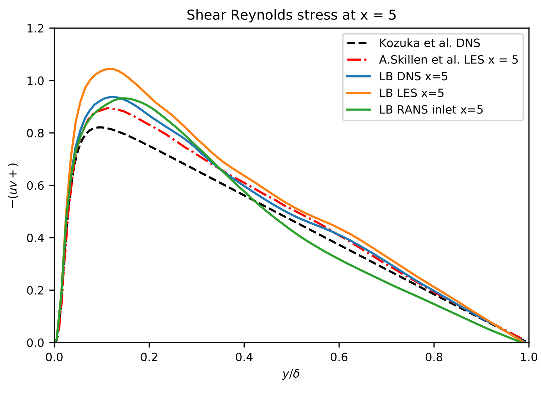
Fig. 2 shows the shear Reynolds stress and wall friction coefficient of the lattice Boltzmann solver compared to the periodic Navier-Stokes DNS data by Kozuka et al. [19] and the Navier-Stokes LES

Name	cell size/H	time step $[U_b/H]$	Turbulence model	Inlet data
Kozuka et al.	0.00028 (min)	-	DNS	Periodic
Skillen et al.	0.0025 (min)	-	LES	Precursor DNS
LB DNS	0.01	0.0001	None	Kozuka et al. DNS
LB LES	0.01	0.0001	LES	Kozuka et al. DNS
LB RANS inlet	0.01	0.0001	None	Mole RANS EBRSM
dx=1/50 LB LES	0.02	0.0004	LES	Kozuka et al. DNS
dx=1/25 LB LES	0.04	0.0016	LES	Kozuka et al. DNS
dx=1/10 LB LES	0.1	0.01312	LES	Kozuka et al. DNS

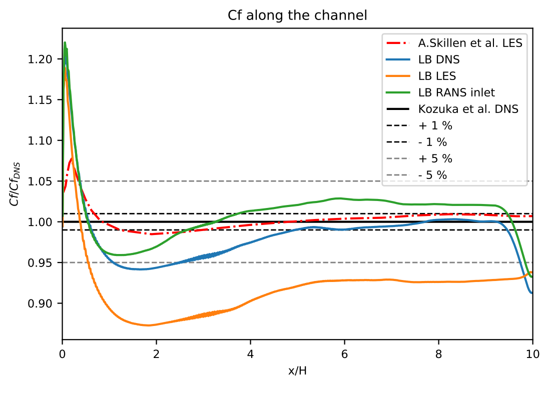
Table 1: Numerical parameters for the different lattice Boltzmann channel flow simulations. The cell size and time step values are adimensionalised using the half channel height H and the mean inlet bulk velocity U_b.

data obtained by Skillen et al. [11] using the original Navier-Stokes version of the SEM inlet described in this paper. The shear Reynolds stress (Fig. 2 a)) for all the tested cases is comparable to Skillen et al. [11] results. The main differences are that the maximum shear stress is displaced for the simulation with no turbulence model and RANS inlet and that the LES over predicts the maximum shear stress.

The skin friction plots in Fig. 2 b show the the distance needed by each simulation to recover the Navier-Stokes DNS results. We calculate the wall friction coefficient as the wall shear stress in the first cell next to the wall divided by the wall shear stress from Kozuka et al. DNS data. The wall shear stress is approximated using 1st order finite differences. First of all, note that the drop in the lattice Boltzmann DNS from $x/h \approx 9$ is due to the sponge layer outlet and thus it is disregarded for the analysis of the results. The results for the lattice Boltzmann simulation with no turbulence model and Kozuka et al. DNS inlet data are comparable to the ones obtained by Skillen et al., with the differece that the lattice Boltzmann SEM simulation recover within 1% of the periodic DNS value at $x/H \approx 5$ and Skillen's Navier-Stokes SEM recovers at $x/H \approx 3$. The lattice Boltzmann simulation with no turbulence model and RANS EBRSM data as the SEM flow statistics stabilises at $x/H \approx 3.5$ to a value within 5% of Kozuka et al. DNS value. The lattice Boltzmann LES underpredicts the wall shear coefficient, maintaining a value within 10% of the periodic DNS results.



(a)

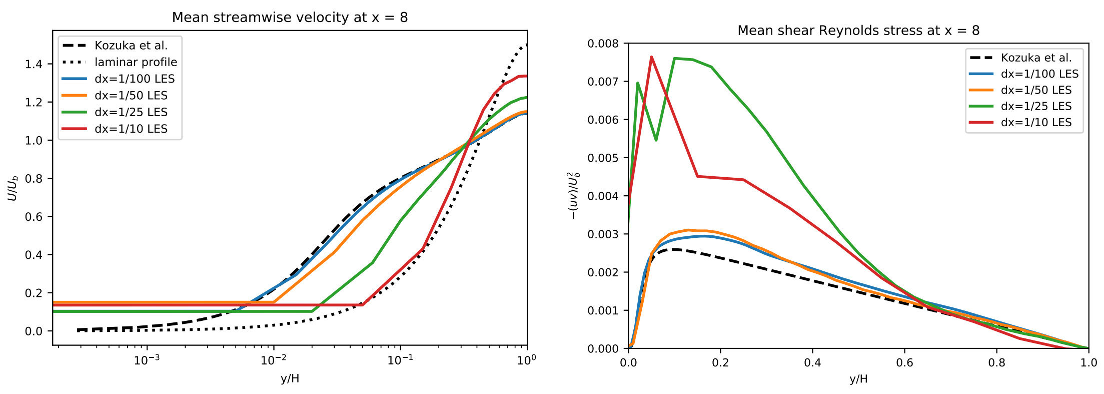


(b)

Figure 2: Shear Reynolds stress in wall units (a) and skin friction coefficient (b) for the channel flow at $Re_\tau = 395$. Periodic Navier-Stokes DNS results by Kozuka et al. [19] (dash black line), SEM inlet Navier-Stokes LES results by Skillen et al. [11] (dash-dot red line), DNS SEM inlet lattice-Boltzmann without turbulence model results (blue line), DNS SEM inlet lattice-Boltzmann LES results (orange line), RANS EBRSM SEM inlet lattice-Boltzmann without turbulence model results (green line).

The differences between the lattice Boltzmann simulations using DNS data EBRSM RANS data at the SEM inlet are to be expected since the inlet Reynolds stresses and dissipation rate are different for EBRSM RANS and DNS (Fig. 1 and eq. 18). The basic Smagorinsky LES model over estimates the eddy viscosity near the walls [21], which explains the under prediction of the wall shear stress in the LES.

The second study for the lattice Boltzmann simulation is to test the sensibility of the results to increasing cell size δx. Fig. 3 shows the mean streamwise velocity and Reynolds stresses at $x/H = 8$ with an increasingly coarse mesh. The lattice Boltzmann solver without turbulence modelling is unstable for meshes with $H \geq \delta/50$. The simulations become stable by activating the LES Smagorinsky model with a Smagorinsky constant $C_{sm} = 0.01$. The velocity and Reynolds stresses in Fig.3 are adimensionalised using the bulk velocity and the reference height H instead of using wall units as in the previous figures. The reason for not adimensionalising using the friction velocity u_τ is that each coarser resolution calculates a less accurate approximation of the shear stress in u_τ regardless of the accuracy of the results; thus the results at different resolutions will be adimensionalised following different criteria and would not be comparable. As expected, increasing the cell size reduces the accuracy of the results especially close to the wall and the mean velocity profiles are closer to the laminar profile. However, the shear Reynolds stress (Fig. 3b)) is present even at the coarser resolutions.



(a) (b)

Figure 3: (a) Non-dimensional streamwise mean velocity and (b) shear Reynolds stress at $x = 8$ of the $Re_\tau = 395$ channel flow with SEM inlet and LES Smagorinsky turbulence model with different mesh cell sizes dx: $dx = 1/100$ (blue), $dx = 1/50$ (orange), $dx = 1/25$ (green) and $dx = 1/10$ (red). Compared with the periodic Navier-Stokes DNS results by Kozuka et al. (cite) (dash line) and the corresponding laminar profile (dotted line).

The BGK lattice Boltzmann model described in this paper discretises the convection term of the Boltzmann equation without numerical diffusion, unlike the common discretisation schemes used to discretise convection in the Navier-Stokes equations. This makes the lattice Boltzmann simulation unstable even at a coarse grids, where the numerical diffusion would act as a stabiliser in a finite volume Navier-Stokes model. This stabilising effect is provided in lattice Boltzmann by the Smagorinsky LES model (see eq. 15), which allows for lattice Boltzmann simulations to maintain turbulent flows at coarse resolutions. However, the coarse resolution and the extra turbulent viscosity near the walls, make the results at coarse resolution near the wall significantly inaccurate.

5. Test case: Isolated building

The aim of this section is to test our lattice Boltzmann LES model in an urban wind setting. The chosen test case is the single rectangular building in a wind tunnel by Meng and Hibi [14] at $Re_H = 478000$, where H is the height of the building and the reference velocity is the inlet velocity at height H.

The LB-LES method presented in this paper models the flow around the building and is embedded inside a pre-calculated Reynolds averaged Navier-Stokes (RANS) simulation (Fig. 4), thus transforming our LB-LES model in a simple RANS-LES model. This section presents the results and execution time of our LB-LES method with and without SEM at the inlet. This results are then compared with the experimental data by Meng and Hibi [14], the underlying RANS simulation and the results obtained using an industry standard unified RANS-LES model, a delayed detached eddy simulation with a $k - \omega$ SST turbulence model (DDES $k - \omega$ SST).

One of the advantages of embedding an LB-LES domain on top of a pre-calculated RANS simulation is that this method can be used to obtain more flow details in any region of a previously calculated RANS domain after the RANS simulation is completed. However, the resolution of the underlying RANS mesh will likely affect the accuracy of the LB-LES results.

5.1. Case description

The simulation of this test case has been run in two stages. First we performed the background unsteady RANS simulation. Then used its results to set up the boundary conditions of a lattice Boltzmann LES embedded in the RANS domain around the building. Fig. 4 shows the position of both domains. The RANS domain covers all the volume inside the orange prism, including the volume covered by the LB-LES domain (blue in Fig. 4).

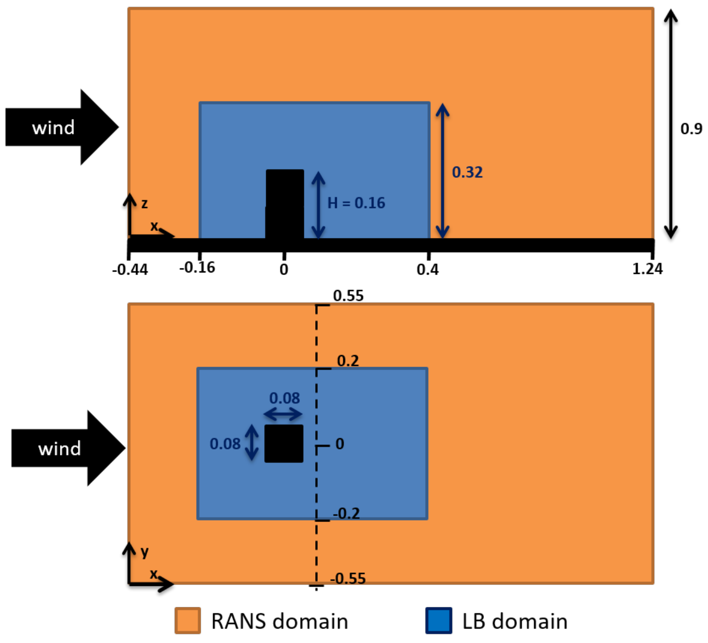


Figure 4: Sketch of the domain for the isolated building test case. The orange region is the underlying RANS simulation; the LB-LES domain is delimited by the blue region. All the values are in meters.

5.1.1. RANS simulation

The size of the RANS domain is the same as the wind tunnel size in Meng and Hibi [14]. The floor and building surfaces implement a no-slip boundary condition with wall modelling, the top and sides in the spanwise direction implement symmetry boundary conditions and the outlet has a fixed pressure and zero gradient boundary condition. The inlet mean velocity and turbulent kinetic energy profiles are set to the inlet profiles used at the inlet of the wind tunnel by Meng and Hibi [14], the rest of the inlet turbulent quantities are deduced using that production of turbulent kinetic energy is in equilibrium with dissipation.

The RANS domain is divided in grid blocks of different cell size with added refinement around the building and the floor (Fig. 5a). The mesh is formed by 3.3 million cells. The smallest block cells measure $0.002m$ and are further refined next to the walls. This mesh was deemed sufficient for the current test case after checking that the change in the mean velocity and Reynolds stress with a finer mesh were not significant.

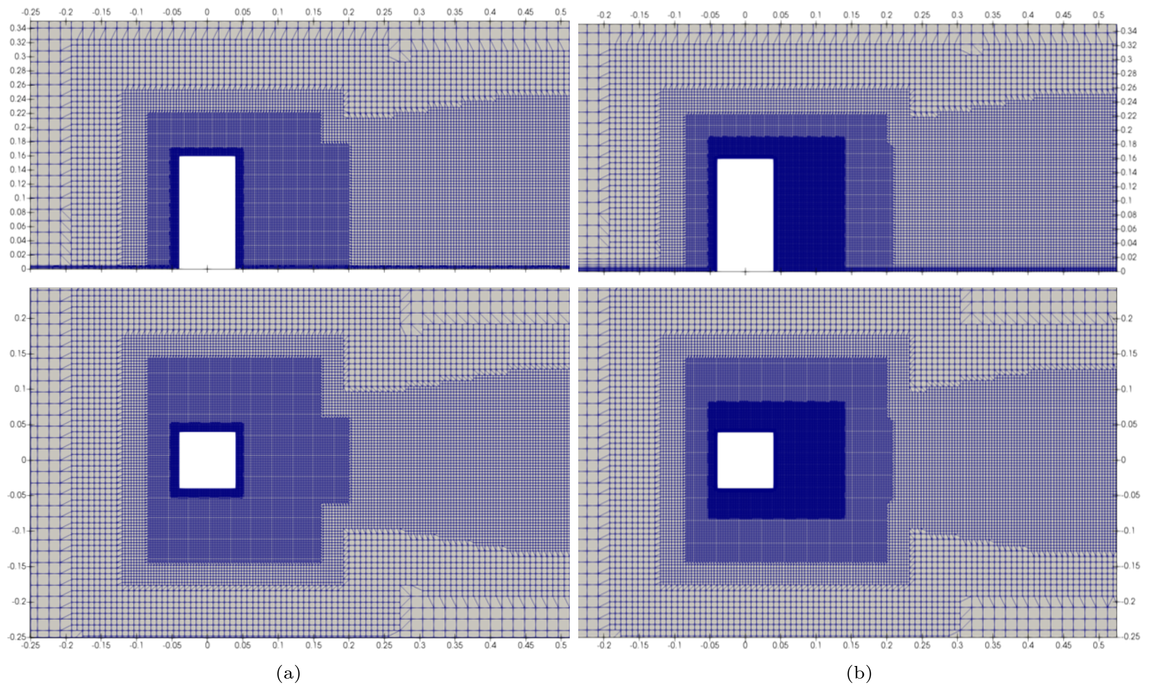


Figure 5: Detail of (a) mesh for the RANS simulation and (b) mesh for DDES simulation around the building. Vertical plane at $y = 0$ (top) and horizontal plane at $z = 0.08$ (bottom).

We ran the unsteady RANS simulation with different turbulence models including Spalart-Allmaras, realizable $k-\epsilon$ and $k-\omega$ SST and compared their results with the mean velocity and turbulent kinetic energy (TKE) profiles reported by Meng and Hibi [14]. The $k - \omega$ SST model was the closest to the experimental results so we decided to use its results as boundary conditions for the LB-LES domain.

5.1.2. $k - \omega$ SST DDES

We also performed a simulation covering the whole domain using a $k - \omega$ SST delayed detached eddy simulation ($k-\omega$ SST DDES) model. The objective of this simulation is to compare the RANS + LB-LES results to the results of an existing hybrid RANS-LES model. The $k - \omega$ SST DDES domain is the same size as the RANS domain but with a higher resolution around the floor and builing and on the wake of the building (Fig. 5b). The smallest block cells measure $0.001m$ and are further refined next to the walls.

The $k - \omega$ SST DDES was run for 14 convective flow through before averaging and then time averaged for an additional 266 convective flow through.

5.1.3. LB-LES

The lattice Boltzmann LES (LB-LES) domain encases the volume surrounding the building (blue in Fig. 4). The objective was to minimise the distance between the cube and the edges of the LB-LES region to reduce the computational resources needed to run the LB-LES region. Its size is 3.5H in x, 2.5H in y and 2H in z. x is the streamwise direction, y is the spanwise direction and z is the vertical direction; H is the height of the building. This domain is divided into constant size cubic cells of size $dx = H/100$ and time is divided in equal time steps of size $dt = 0.00001$ seconds. The floor and building surfaces implement a no-slip boundary condition via half-way bounce back. The velocity at the sides and top of the domain is fixed to the RANS velocity; the top implements a forced equilibrium boundary condition [22] and the sides implement a regularised velocity boundary condition [17]. The outlet is not coupled to the RANS results, instead it implements an extrapolated boundary conditions, i e. the particle distribution functions entering the domain are copied from the cells parallel upstream of the outlet cells. The inlet boundary also implements a regularised velocity boundary condition using data from the RANS results. We ran two simulations with different inlet data:

- LB-LES U inlet: The inlet velocity is fixed to the RANS velocity. In this case, the free stream turbulence in the RANS simulation is neglected at the LB-LES. Thus the LB-LES contains only the turbulence generated by the building.
- LB-LES SEM inlet: The inlet velocity is calculated and updated at each LB-LES time step using the synthetic eddy method (SEM) described in section 3.

The SEM needs mean velocity, Reynolds stresses and turbulent dissipation rate ϵ to generate the instantaneous velocity. In LB-LES case B the mean velocity is set to the RANS velocity, the Reynolds stresses are approximated from the RANS velocity and turbulent kinetic energy using the Boussinesq eddy viscosity assumption and the turbulent dissipation rate ϵ is calculated using $\epsilon = TKE * 0.09$. The maximum synthetic eddy size is set to $\sigma_{max} = H/8$ and the minimum eddy size is set to $\sigma_{min} = 4dx$, where dx is the cell size. The synthetic eddy size calculated from the approximated RANS Reynolds stresses and ϵ presents only a small portion of the inlet of the domain with eddy sizes between $4dx$ and $H/8$; which yields a constant synthetic eddy size of $H/8$ across the inlet except in the region close to the ground. We chose to limit the eddy size to $H/8$ because it is a standard measure in channel flow and it reduces averaging time.

The RANS velocity is interpolated from the RANS mesh to the LB-LES mesh using lineal interpolation.

Both LB-LES domains were run for 561 convective flow through before averaging, and then time averaged for an additional 1685 convective flow through.

5.1.4. Computational resources

The Navier-Stokes solver used in this paper for the DDES and RANS simulations is the open source CFD software OpenFOAM [23]. We ran the DDES and RANS simulation using 288 and 120 Ivy Bridge CPU cores respectively across multiple nodes of the ARCHER supercomputer. Each core is assigned approximatelly 29000 cells in both simulations so that the RANS and DDES simulation time can be compared in a situation in which each CPU core has approximately the same computational load.

The lattice Boltzmann solver used in this paper for the LB-LES is the open source CUDA software GASCANS [24]. The computational resources available to us to run GASCANS were 8 Intel Skyline CPU cores and one Nvidia V100 GPU in a single node of the Computational Shared Facility (CSF) at the University of Manchester.

Table 2 summarises the characteristics of the 4 simulations.

	RANS	**DDES**	**LB-LES A**	**LB-LES B**
Domain size [H]	$10.5 \times 6.875 \times 5.625$	$10.5 \times 6.875 \times 5.625$	$3.5 \times 2.5 \times 2$	$3.5 \times 2.5 \times 2$
Cells[millions]	3.3	8.3	17.5	17.5
Time step[s]	10^{-4}	10^{-5}	10^{-5}	10^{-5}
Resources	120 CPUs	288 CPUs	1 CPU, 1 GPU	8 CPUs, 1 GPU

Table 2: Numerical parameters and computational resources for the isolated building simulations.

5.2. Results and discussion

Fig. 6 displays a snapshot of the streamwise solved velocity for the RANS simulation (left) and the DDES (right). The DDES results show turbulence in the wake of the cube that dissipate as the mesh coarsens downstream.

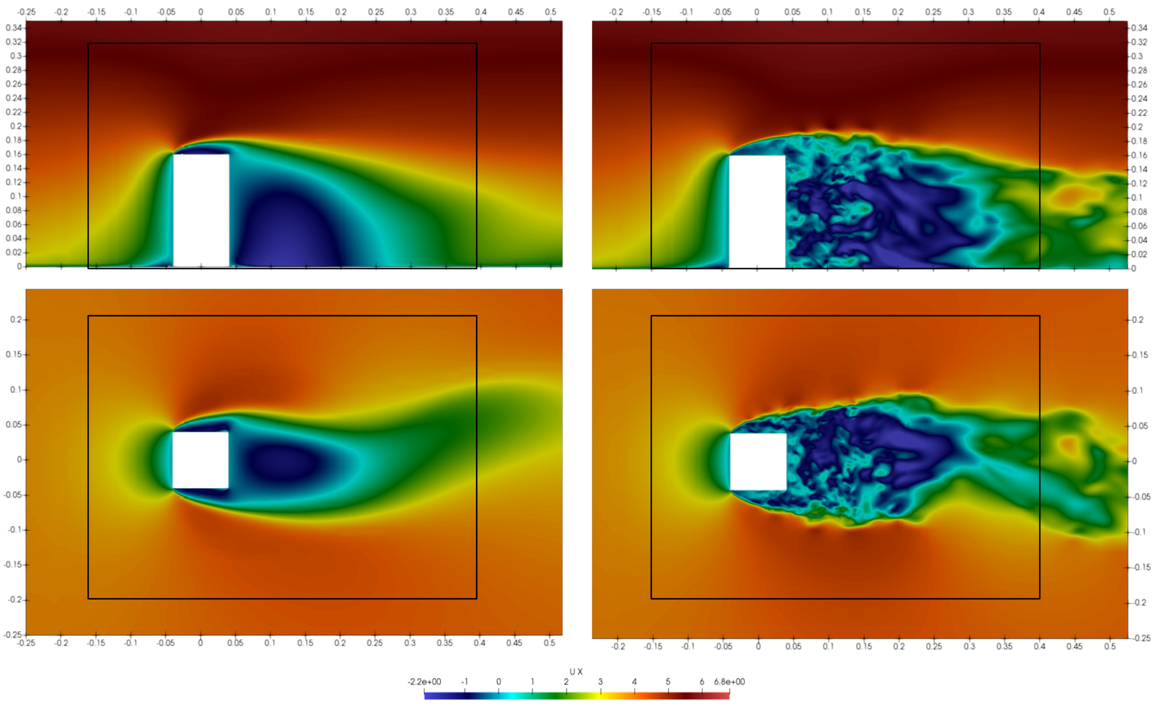


Figure 6: Streamwise velocity u_x contours for the unsteady $k-\omega$ SST RANS simulation (left) and the $k-\omega$ SST DDES (right) at the vertical plane at $y = 0$ (top) and horizontal plane at $z = 0.08$ (bottom). The black rectangles mark the boundaries of the LB-LES.

The LB-LES resolved streamwise velocity $\bar{u}_x$ for both the LB-LES U inlet and LB-LES SEM inlet is shown in Fig. 7; LB-LES U inlet at the left and LB-LES SEM inlet at the right.

In the LB-LES U inlet plot (Fig. 7 left), the velocity at the LB-LES domain inlet matches the RANS velocity. There are some discrepancies between the LB-LES and the RANS velocity at the top boundaries and there exists an instability not generated by the RANS data or the building at the side boundaries of the LB-LES domain; however, this instability does not seem to interact with the wake of the building. We can also observe spurious reflections from the sides of the LB-LES domain, which are particularly visible in front and on top of the building. Finally, note that the fact that the wake position in the LB-LES U inlet results is synchronised with the wake position in the RANS results is coincidence; since the RANS results are fixed to the state shown in Fig. 7 for the duration of the LB-LES.

The LB-LES SEM inlet results (Fig. 7 right) show the instantaneous velocity generated by the inlet SEM, while the mean velocity at the LB-LES domain inlet matches the RANS velocity (see section 3). The LB-LES resolved streamwise velocity at the top and sides of the domain is fixed to the RANS velocity, however the turbulent eddies are present in close proximity to the top boundary without any observable dissipation effect. At the vicinity of the LB-LES inlet the eddy size is roughly $H/8$, except just intermediately above the floor where they are smaller. Fig. 7 shows the eddies propagate across the domain and become smaller in the region past the building. The instability on the side wall and the spurious reflections near the inlet corners can still be observed but they are masked by the turbulent eddies from the SEM inlet. Finally, there are differences in the shape and velocity profiles of the wake in both cases. LB-LES SEM inlet presents a larger area with midrange velocities (shown in yellow) and the shape of its wake is less defined.

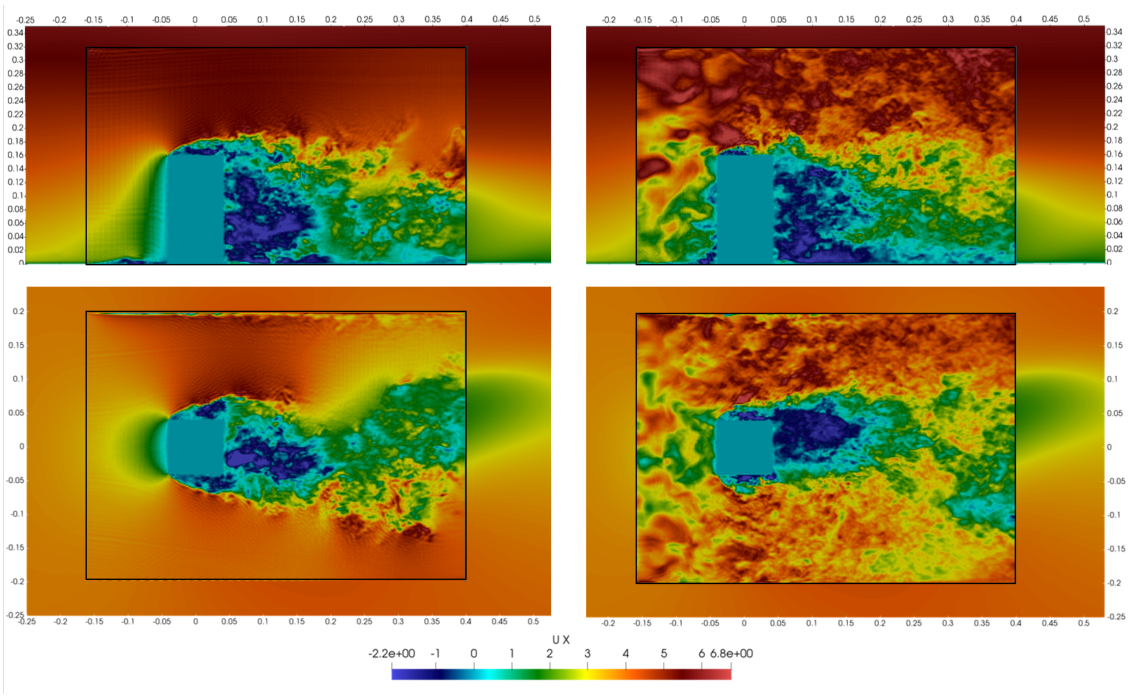
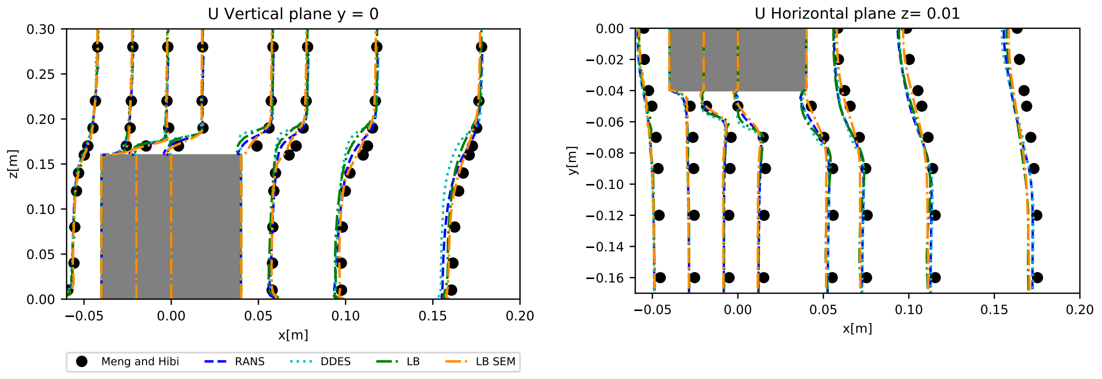


Figure 7: Instantaneous streamwise velocity u_x contours in the LB-LES domain overlayed on the RANS streamwise velocity contours in the RANS simulation. LB-LES case A (left), LB-LES case B (right), vertical plane at $y = 0$ (top) and horizontal plane at $z = 0.08$ (bottom). The black rectangles mark the boundaries of the LB-LES.

Meng and Hibi [14] provide experimental data at 8 locations x/H = -0.375, -0.25, -0.125, 0.0, 0.25, 0.375, 0.625, 1 upstream and downstream the building on the vertical plane $y/H = 0$ and the horizontal plane $z/H = 0.0625$. Fig. 8 shows the mean streamwise velocity for all the computational cases compared with Meng and Hibi's measurements. As expected, all models present resonable agreement with the measurements. The main discrepancies are found in the vicinity of the building in the wake region. In the vertical plane all models except the LB-LES SEM inlet over estimate the mean velocity value; in the horizontal plane the mean velocity is under estimated by all the models, however LB-LES SEM inlet presents the closest agreement with the experimental data.

Fig. 9 show the root mean squared of the resolved streamwise velocity $< \bar{u}' >$ (Fig. 9(a) and (b)), spanwise velocity $< \bar{v}' >$ (Fig. 9(c) and (d)) and vertical velocity $< \bar{w}' >$ (Fig. 9(e) and (f)) compared with Meng and Hibi's experimental root mean squared of the velocity components. LB-LES SEM inlet is able to capture the physics of the flow correctly, showing the closest agreement with the experimental data. However, it understimates the streamwise fluctuating velocity on the horizontal plane near the ground in the building's vicinity and the wake region. A noticeable difference between the LB-LES SEM inlet results and the DDES and LB-LES U inlet results upstream and above the building is that both DDES and LB-LES U inlet show a nearly no fluctuating velocity away from the building, while LB-LES SEM inlet shows turbulence levels according to the experimental data.

The introduction of SEM in the LB-LES SEM inlet simulation has a significant effect on the simulation results, both for the instantaneous velocity and its statistics, and it alters the shape and characteristics of the building's turbulent wake, even increasing the accuracy of the mean velocity near the cube (Fig. 8), which is where the other models were less accurate. The main effect of the SEM inlet can be seen in the fluctuating velocity plots (Fig. 9). The LB-LES SEM inlet simulation accurately reproduces the free stream turbulence levels present on the wind tunnel experiment using the RANS



(a) (b)

Figure 8: Mean streamwise velocity U on the vertical plane (left) and the horizontal plane (right). Meng and Hibi (cite) experimental data (dots), $k-\omega$ SST RANS (dashed blue line), $k-\omega$ SST DDES (dotted cyan line), LB-LES U inlet (dash-dot green line) and LB-LES SEM inlet (dash-dot orange line).

mean flow data at the LB-LES SEM inlet simulation. The resolved velocity of the LB-LES U inlet and the $k-\omega$ SST DDES lacks the free stream turbulence data incorporated into the LB-LES SEM inlet by SEM, which affects not only the upstream results and the results above the building but also the wake. The turbulent kinetic energy results with SST might improve since they incorporate the effect of the subgrid scale turbulence.

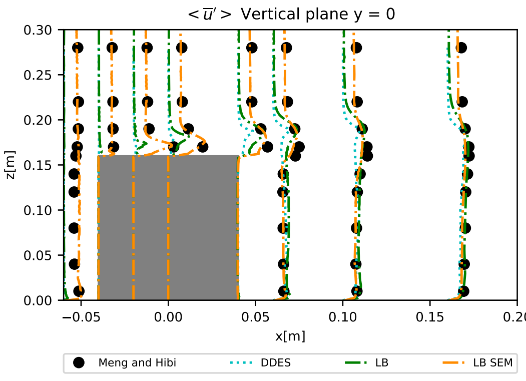
The boundary condition on the top of both LB-LES domains is forced equilibrium, while the boundary condition on the sides and inlet is regularised velocity. The artificial instabilities near the side walls might be caused by the interaction between the boundary conditions on the sides and outlet of the LB-LES domains; since the inlet boundary does not present the instabilities. The top boundary does not present instabilities either. Moreover, the instantaneous velocity at the top boundary is also set to the RANS velocity and a adaptation region in which the RANS velocity becomes turbulent would be expected. However, this region is negligible probably due to the inviscid nature of the forced equilibrium boundary condition (i e. the lattice Boltzmann equations assuming that all the particles are in equilibrium are equivalent to the inviscid Navier-Stokes equations [22]).

Finally, the SEM minimum and maximum synthetic eddy size σ_{min} and σ_{max} have been chosen in a somewhat arbitrary manner and their effect of the results and computational resources needs further study.

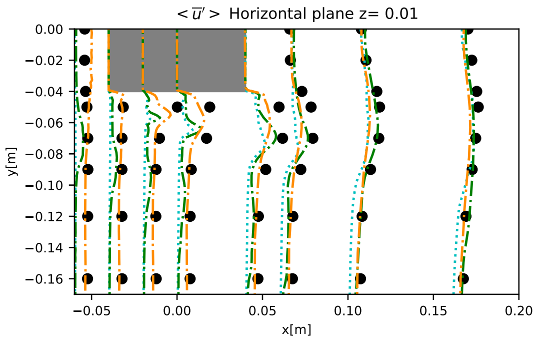
5.2.1. Computational resources

We measured the time taken by each model to simulate a convective time unit, defined as the physical time taken by the flow moving at the reference velocity to cover the width of the building. The reference velocity is the mean velocity at the inlet of the wind tunnel at $z = H$, $U_{ref} = 4.491m/s$. The width of the building is $h = 0.08m$. Thus a convective time unit $t_c = h/U_{ref} = 0.0178s$. Fig. 10 shows the computer time taken by each simulation. Note that $k-\omega$ SST DDES and RANS model the whole experimental domain while the lattice Boltzmann simulations model only the region around the building. $k-\omega$ SST DDES is the slowest simulation, taking 4 times more than the RANS simulation. The application of the SEM inlet in the LB-LES SEM simulation increases its computational time 3.5 times over the computational time of the same solver and domain run using the RANS velocity as inlet. However, it is still 3 times faster that the $k-\omega$ SST DDES domain.

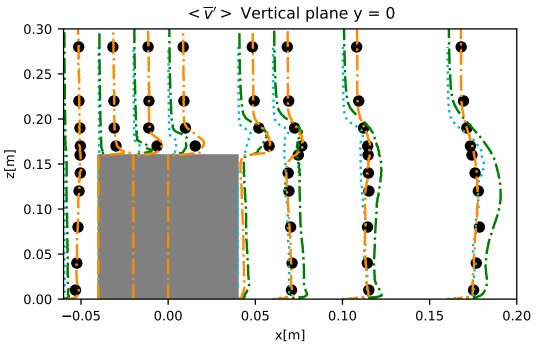
The LB-LES SEM inlet test case is the most accurate and its computational time is lower than the DDES. Besides, the added computational resources of the RANS and the LB-LES SEM simulation are less than the $k-\omega$ SST DDES computational resources. The difference between the run time for LB-LES SEM inlet and LB-LES U inlet is the execution of the SEM boundary condition; which is



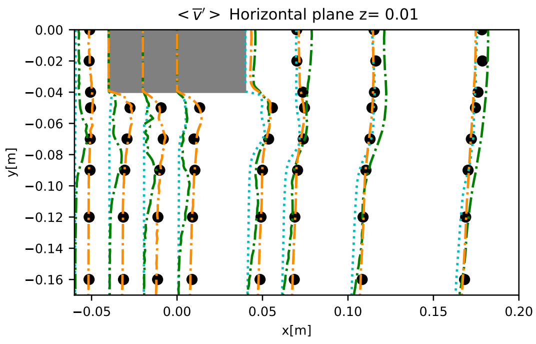
(a)



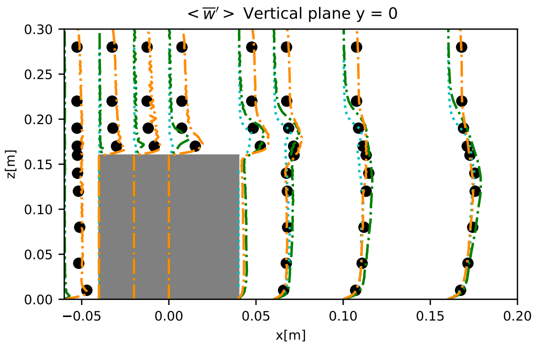
(b)



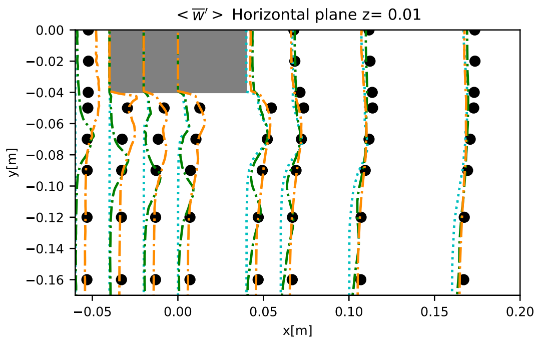
(c)



(d)



(e)



(f)

Figure 9: Resolved root mean squared of the velocity components $< \bar{u}' >$, $< \bar{v}' >$, $< \bar{w}' >$ on the vertical plane (left) and the horizontal plane (right). Meng and Hibi (cite) experimental data (dots), $k - \omega$ SST DDES (dotted cyan line), LB-LES U inlet (dash-dot green line) and LB-LES SEM inlet (dash-dot orange line)..

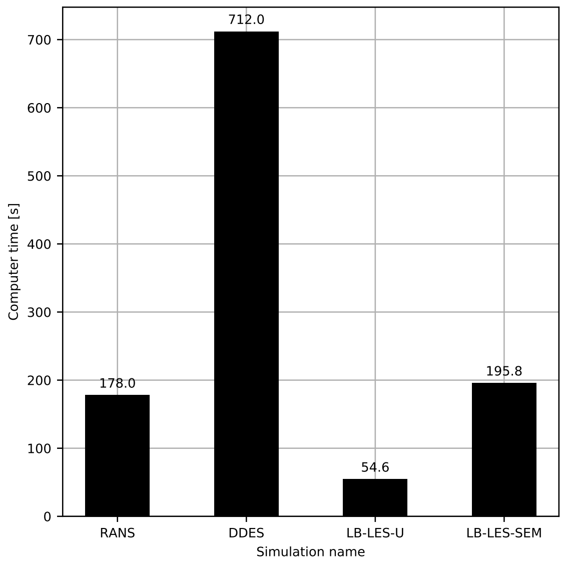


Figure 10: Computational time needed to run a convective flow through for the RANS simulation, the $k-\omega$ SST DDES the LB-LES U inlet simulation and the LB-LES SEM inlet simulation.

still computationally expensive and executed on CPU even if the LB-LES method is run on GPU. We predict that the computational time of the LB-LES SEM simulation might be reduced by optimising the SEM algorithm further and including it in the LB-LES algorithm executed in GPU.

6. Conclusions

This paper presents a BGK lattice Boltzmann LES solver (LB-LES) able to use RANS data as boundary conditions. The RANS turbulence information is input to the inlet of the LB-LES solver domain via Skillen et al. [11] synthetic eddy method. The SEM implementation is first tested in a $Re_\tau = 395$ channel flow and compared with the results from Skillen et al.. The second test case is the wind tunnel experiment by Meng and Hibi [14], which models wind around a rectangular building at $Re_H = 47893$. For this case, the LB-LES solver domain is thigh around the building and uses the results from a pre-calculated RANS simulation as boundary conditions for its top, sides and SEM inlet. We then compared the LB-LES results and computational time with the underlying RANS simulation and a $k-\omega$ SST DDES.

The main conclusion from the test cases results is that the SEM inlet incorporates the mean turbulence from the RANS data into the LB-LES resolved velocity, enabling the LB-LES model to capture the physics of the flow correctly and show good agreement with the experimental results. Regarding the computational resources, the LB-LES SEM simulation had a 3 times shorter running time than the $k-\omega$ SST DDES even if we ran all the LB-LES using a maximum of 8CPU cores and a single Nvidia V100 GPU card. This reduction in computational resources show the potential to run SEM LB-LES for industry relevant cases using consumer level computational resources.

The BGK lattice Boltzmann solver using Skillen et al. [11] SEM is able to reproduce Skillen et al. Navier-Stokes results within acceptable accuracy. Activating the Smagorinsky LES turbulence model introduces excessive turbulent viscosity near the channel wall, which decreases the accuracy of the results. This effect increases with the cell size and could be addressed by using wall modelling. As expected, the innacuracy of the mean turbulence data provided by a RANS model also has a detrimental effect in the accuracy of the SEM lattice Boltzmann results.

The LB-LES solver is able to reproduce turbulent flow at coarse mesh resolutions due to the low numerical dissipation of the BGK lattice Boltzmann discretisation combined with the stabilizing effect of the Smagorinsky LES turbulent viscosity.

The effect of the SEM parameters σ_{min} and σ_{max} and the best combination of boundary conditions for the top and sides boundaries of the LB-LES model is still unclear and needs further investigation. Moreover, the computational cost of the SEM implementation might be further reduced by implementing the SEM model as part of the GPU side of the LB-LES solver.

Acknowledgements The authors would like to acknowledge the assistance given by Research IT and the use of the Computational Shared Facility at The University of Manchester and the use of the ARCHER UK National Supercomputing Service. The authors would also like to thank Andrew Mole for providing the RANS EBRSM mean flow data used in the channel flow test case.

Declaration of conflicting interests

The authorS declared no potential conflicts of interest with respect to the research, authorship and/or publica- tion of this article.

References

[1] Y. Toparlar, B. Blocken, B. Maiheu, G. J. van Heijst, A review on the CFD analysis of urban microclimate, Renewable and Sustainable Energy Reviews 80 (September 2016) (2017) 1613–1640.
URL `http://dx.doi.org/10.1016/j.rser.2017.05.248`

[2] L. Merlier, J. Jacob, P. Sagaut, Lattice-Boltzmann Large-Eddy Simulation of pollutant dispersion in street canyons including tree planting effects, Atmospheric Environment 195 (September) (2018) 89–103. doi:10.1016/j.atmosenv.2018.09.040.
URL `https://doi.org/10.1016/j.atmosenv.2018.09.040`

[3] S. Lenz, M. Schönherr, M. Geier, M. Krafczyk, A. Pasquali, A. Christen, M. Giometto, Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU, Journal of Wind Engineering and Industrial Aerodynamics 189 (October 2018) (2019) 151–162.

[4] J. Jacob, P. Sagaut, Wind comfort assessment by means of large eddy simulation with lattice Boltzmann method in full scale city area, Building and Environment 139 (March) (2018) 110–124. doi:10.1016/j.buildenv.2018.05.015.
URL `https://doi.org/10.1016/j.buildenv.2018.05.015`

[5] N. Onodera, T. Aoki, T. Shimokawabe, H. Kobayashi, Large-scales LES Wind Simulation using Lattice Boltzmann Method for a 10km x 10km Area in Metropolitan Tokyo, e-Science Journal, Global Scientific Information and Computing Center 9 02–08. arXiv:arXiv:1011.1669v3, doi:10.1017/CBO9781107415324.004.

[6] F. Mathey, D. Cokljat, Zonal Multi-Domain RANS/LES Simulation of Airflow Over the Ahmed Body, Engineering Turbulence Modelling and Experiments 6 (2005) 647–656doi:10.1016/B978-008044544-1/50062-5.

[7] J. Frohlich, D. von Terzi, Hybrid LES/RANS methods for the simulation of turbulent flows, Progress in Aerospace Sciences 44 (5) (2008) 349–377. doi:10.1016/j.paerosci.2008.05.001.
URL `http://linkinghub.elsevier.com/retrieve/pii/S0376042108000390`

[8] M. Jadidi, F. Bazdidi-Tehrani, M. Kiamansouri, Embedded large eddy simulation approach for pollutant dispersion around a model building in atmospheric boundary layer, Environmental Fluid Mechanics 16 (3) (2016) 575–601. doi:10.1007/s10652-016-9444-5.

[9] K. Wijesooriya, D. Mohotti, K. Chauhan, D. Dias-da Costa, Numerical investigation of scale resolved turbulence models (LES, ELES and DDES) in the assessment of wind effects on supertall structures, Journal of Building Engineering 25 (June) (2019) 100842. doi:10.1016/j.jobe.2019.100842.
URL `https://doi.org/10.1016/j.jobe.2019.100842`

[10] E. Sergent, Vers une méthodologie de couplage entre la Simulation des Grandes Echelles et les modèles statistiques (2002).

[11] A. Skillen, A. Revell, T. Craft, Accuracy and efficiency improvements in synthetic eddy methods., International Journal of Heat and Fluid Flow 62 (2016) 386–394.
URL `http://dx.doi.org/10.1016/j.ijheatfluidflow.2016.09.008`

[12] R. Poletto, T. Craft, A. Revell, A new divergence free synthetic eddy method for the reproduction of inlet flow conditions for les, Flow, Turbulence and Combustion 91 (3) (2013) 519–539. doi:10.1007/s10494-013-9488-2.

[13] J. Millar, R. Wayland, J. Holgate, Hybrid RANS-LES Simulation of a Tall Building in a Complex Urban Area, in: Progress in Hybrid RANS-LES Modelling: Papers Contributed to the 7th Symposium on Hybrid RANS-LES Methods, 17-19 September, 2018, Berlin, Germany, 2018, pp. 379–388.

[14] Y. Meng, K. Hibi, Turbulent measurments of the flow field around a high-rise building (1998).

[15] S. Hou, J. Sterling, S. Chen, G. D. Doolen, A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows, Pattern formation and lattice gas automata 6 (1996) 151–166. arXiv:9401004.
URL `http://books.google.com/books?hl=en&lr=&id=TIdy6-ItW7YC&oi=fnd&pg=PA151&dq=A+Lattice+Boltzmann+Subgrid+Model+for+Hig`

[16] H. K. Versteeg, W. Malalasekera, An Introduction to Computational Fluid Dynamics, Vol. M.

[17] J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Straight velocity boundaries in the lattice Boltzmann method, Physical Review E - Statistical, Nonlinear, and Soft Matter Physics 77 (5) (2008) 1–16. doi:10.1103/PhysRevE.77.056703.

[18] S. Fan, M. Camps Santasmasas, X. Guo, C. Yang, A. Revell, In preparation (2021).

[19] K. Kozuka, Y. Seki, H. Kawamura, Direct numerical simulation of turbulent heat transfer with a high spatial resolution, Proc. of the 7rd International Symposium on Engineering Turbulence Modelling and Mesurements - ETMM7 1 (September 2016) (2008) 163–168.

[20] P. Sagaut, Large Eddy Simulation for Incompressible Flows – An Introduction, 3rd Edition, Springer, 2005. doi:10.1108/aeat.2001.12773eae.001.

[21] M. Weickert, G. Teike, O. Schmidt, M. Sommerfeld, Investigation of the LES WALE turbulence model within the lattice Boltzmann framework, Computers and Mathematics with Applications 59 (7) (2010) 2200–2214. doi:10.1016/j.camwa.2009.08.060.
URL `http://dx.doi.org/10.1016/j.camwa.2009.08.060`

[22] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, E. M. Viggen, The Lattice Boltzmann Method Principles and Practice, Springer, 2017. doi:10.1007/978-3-319-44649-3.

[23] The OpenFOAM Foundation, Openfoam.
URL `https://openfoam.org/version/4-0/`

[24] M. Camps Santasmasas, A. R. G. Harwood, I. Hinder, S. Fan, F. Owen, J. O'Connor, A. J. Revell, In preparation (2021).

Nothing in life is to be feared, it is only to be understood. Now is the time to understand more, so that we may fear less

Marie Curie

Chapter 5

Contributions to the field

This chapter outlines the major contributions of this thesis and their relevance to its two main objectives:

- Expand the use of the lattice Boltzmann method for computational wind engineering (CWE) and urban wind flow (Papers II, III and IV).
- Develop a coupled Navier-Stokes lattice Boltzmann model (NSLB) to reduce the computational resources needed by both its component solvers; while achieving the desired accuracy in the interest region. (Papers I and IV).

5.1 Expand the use of lattice Boltzmann method in CWE

GASCANS, described in Paper IV section B, is the LB GPU code used and developed during this thesis with the objective to be used in CWE simulations. The author's main contributions to GASCANS and its main scientific contributions to CWE are:

- Adapting and implementing SEM to a LB method. GASCANS can generate instantaneous inlet velocity from mean flow data using a synthetic eddy method. This is a way to introduce the free stream turbulence into the LES simulation, thus improving its accuracy. Moreover, the mean flow data can be provided by an external solver and updated during run time .

- Couple GASCANS with external solvers. GASCANS is able to use data received from external solvers as boundary conditions. It can also send its velocity results in user-defined regions to an external solver. This exchange can be at run time. This allows to embed a GASCANS domain inside a less computationally expensive domain, thus reduce the size of the lattice Boltzmann domain and its computational cost.

Paper II in section 4.3.2 applies GASCANS to a urban wind modelling setting. Concretely, paper II models the wind around a rectangular building at $Re_H = 47893$, based on the wind tunnel experiment by Meng and Hibi [88]. The top and side boundaries of the GASCANS simulation domain are set to the velocity values of a pre-calculated RANS k-omega SST simulation that covers the size of the wind tunnel experiment. The mean flow results from the RANS simulation are interpolated to the GASCANS inlet, and the GASCANS inlet velocity is generated from them using SEM. Transferring the results from a RANS simulation into an embedded LB-LES domain using SEM to bridge from the scales resolved by RANS to the scales resolved by LB-LES is another of the scientific contributions of this thesis.

5.2 Develop and test the NSLB method

The main contribution of the present thesis in this regard is the development of a framework to couple NS solvers to LB solvers, with two-way coupling for laminar flow (NS coupled to LB) and one-way coupling for turbulent flow (RANS couppled to LB-LES). The NSLB method is applied to laminar flow (Paper I), and turbulent flow (Paper II).

Finally, in conclusion, let me say just this.

Peter Sellers

Chapter 6

Conclusions and future work

6.1 Conclusions

This PhD project has focussed on reducing the computational resources needed for turbulence resolving CFD simulations of urban wind flow. Traditional Eulerian Navier-Stokes solvers are efficient at modelling large regions due to their relatively low memory consumption and mesh refinement. On the other hand, BGK stream-collide lattice Boltzmann solvers resolve turbulent flows with an accuracy comparable to Eulerian Navier-Stokes solvers and their algorithm is easy to implement and efficient on GPUs; however, their high memory consumption and constant cell size meshes limits the size of the domain and/or accuracy. This thesis presents a coupled Navier-Stokes lattice Bolztman (NSLB) solver aimed at low computational resources while maintaining high accuracy and resolving turbulence in the interest region.

The coupled Navier-Stokes lattice Boltzmann (NSLB) method presented in this thesis can be configured for one-way and two-way coupling. The LB submodel is run on GPU and it can be executed as a DNS or with a LES Smagorinsky turbulence model. Smagorinsky LES enables the LB solver to run high Reynolds number flows with coarse meshes while reducing the loss of accuracy due to the mesh coarsening. The presented lattice Boltzmann solver can be coupled one-way to any model that provides velocity values at the boundaries of the lattice Boltzmann domain and two-ways to any model that is also able to receive velocity data from the lattice Boltzmann. The presented lattice Boltzmann solver implements the synthetic eddy method (SEM) boundary condition by Skillen et al. [7]. SEM generates an instantaneous velocity field from supplied mean flow statistics, thus incorporating the mean turbulence information to the lattice Boltzmann resolved velocity. Both the SEM and the Smagorinsky LES turbulence models can be activated and deactivated as needed.

The NSLB model is tested in the following cases:

- Test case 1: Two-way coupled NSLB solver to solve the $Re_H = 150$ flow over a wall mounted cube in a channel by Hwang and Yang [87]. In this case velocity at the outlet of the lattice Boltzmann domain is set to the Navier-Stokes domain velocity and the velocity at the Navier-Stokes inlet is set to the lattice Boltzmann velocity.

- Test case 2: The lattice Boltzmann LES model was embedded in a pre-calculated RANS simulation of flow around a building at $Re_H = 47893$ based on the wind tunnel measurements by Meng and Hibi [88]. The lattice Boltzmann domain covered the region around the building. Its top and side boundaries were set to the mean RANS velocity. Its inlet used SEM to generate turbulence from the RANS mean flow.

The results from the test cases show that the NSLB method, both one-way and two-way coupled, is able to recover and sometimes improve the accuracy of its components in the interest region. Moreover, it only adds GPU cards as computational resources, which are not used by the coupled Navier-Stokes solver.

Regarding to two-way coupled NSLB method, the results from test case 1 show that the NSLB method is able to reproduce the results of both the single lattice Boltzmann and the single Navier-Stokes domain below 1 % difference. The main advantage of two-way coupling over one-way coupling is that both models receive feedback from the other, which allows to put the boundaries complex flow regions. Regarding the computational resources, the NSLB method simulation uses less memory than the single lattice Boltzmann simulation and less computing time than the single Navier-Stokes simulation. The model is also robust to different positions and size of the overlap region (i e. region solved by both the Navier-Stokes and lattice Boltzmann domains), however there is a limit on the size of the region which depends on the complexity of the flow in the overlap region; smaller regions than the limit will destabilize the simulation.

Test case 2 is the most representative of urban wind flow. The LB-LES SEM results show good agreement with the experimental results, also adding fluctuating velocity data not provided by the RANS simulation. Part of the success of the embedded LB-LES domain is the instantaneous inlet velocity, which is generated using a synthetic eddy method and incorporates the mean turbulence statistics from the RANS results into the LB-LES resolved velocity. Regarding the computational resources used by the simulations, the embedded lattice Boltzmann model uses less computational resources than the DDES case and completes the simulation in less time.

6.2 Future work

The NSLB solver presented in this thesis can be expanded and improved in the following areas:

- Extend its applicability to wind engineering studies by extending our lattice Boltzmann solver GASCANS to model the following physical processes often present and studied in wind engineering projects.
    - **Thermal modelling and buoyancy effects**. The majority of urban wind modelling and wind engineering studies are concerned with temperature variations and buoyancy effects both for urban wind and internal ventilation [35].
    - **Passive scalar transport**; needed to model contaminant dispersion for both emergency toxic contaminant release and air pollution studies. This studies are an important part of wind engineering and need accurate non-steady solvers to produce accurate results.
    - **Complex geometries**. The NSLB model has only been tested with isolated rectangular shaped builings. However, urban geometry includes different shaped buildings that are rarely studied in isolation from the surrounding city. The NSLB model is programmed to accept any geometry input by the user but its accuracy needs to be tested, especially around curved geometries.

- Increase the accuracy of GASCANS to model turbulent flows. Concretely:
    - **Wall functions**. The Smagorinsky LES models introduces extra turbulent viscosity in the cells near the wall, which impairs the results of the LB-LES models in that region. Moreover, this detrimental effect is increased when the resolution is coarsened. Wall functions are special treatment for the cells around the walls aimed at correcting the Smagorinsky LES results. The simplest wall functions are just equations that correct the filter width of the Smagorinsky model (f e. Van Driest model [72]). Another way to correct the Smagorinsky LES results near the walls is to use RANS models near the wall and couple them to the LES results. This last technique is especially relevant for GASCANS, since it takes advantage of GASCANS's ability to exchange information with external solvers (the near-wall RANS solver in this case).
    - **Parameter study**. The NSLB solver applied to turbulent flows presented in this study contains configuration parameters whose effect in the results needs to be studied further. These parameters include the minimum and maximum synthetic eddy size for the synthetic eddy method (SEM) boundary conditions, the boundary conditions for the top and sides of the embedded LB-LES domain and the size of the LB-LES embedded domain.
- Increase the computational efficiency of the lattice Boltzmann solver GASCANS. Some suggestions are:
    - **SEM implementation**. We expect to increase the computational speed of the SEM boundary condition by implementing the current SEM model in GPU and run it together with the main GPU lattice Boltzmann code in GASCANS.
    - **Multi-node multi-GPU**. The current version of GASCANS can work with more than one GPU but always in a single node. Extending the software to use multiple GPU cards in different nodes will potentially increase its computational speed and would definetely increase the maximum number of lattice Boltzmann cells per simulation.

  - **Link-wise artificial compressibility method (LWACM).** The LWACM by Asinari et al. [89] is a lattice Boltzmann method that saves on computer memory by storing only the macroscopic variables velocity and density; it computes the particle distribution functions at each time step using the velocity and density. The savings in memory would be advantageous to run larger number of cells per simulation.

- Develop further the NSLB model by:

  - Implementing the **SEM boundary condition** for all boundaries and inlet mean flow directions.

  - Test and improve **Two-way coupling** for turbulent flow and for LB-LES domains embedded in Navier-Stokes domains.

  - Couple the lattice Boltzmann solver with a **compressible Navier-Stokes** solver. The lattice Boltzmann equations implemented in GASCANS are weakly compressible while the Navier-Stokes solver we coupled them to (pisoFoam) is incompressible. pisoFoam uses constant density, while the density in GASCANS changes due to compressibility effects. Coupling GASCANS to a compressible Navier-Stokes solver migth improve the results and stability of the coupling.

  - Implement and test more accurate boundary conditions for the NS to LB boundary in the LB sub-domain

Finally, we only tested the NSLB solver for flow around blunt objects with an application to wind engineering and urban wind modelling in mind. However, other CFD applications that involve turbulent flows could also benefit from the reduction in computational resources offered by the NSLB solver. Two of those applications are automotive flows applied to vehicle design and cardiovascular applications. We have already some experience working with this type of flows (see. Revell et al. [90]) so they are ideal candidates to start expanding the NSLB solver to other fields.

References

[1] Timm Krüger. *The Lattice Boltzmann Method Principles and Practice*. Number March 2015. 2017.

[2] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics*, 32(4):1, 2013.

[3] Matthias Müller, David Charypar, and Markus Gross. Particle-Based Fluid Simulation for Interactive Applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (5):154–159, 2003.

[4] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics*, 24(3):910, 2005.

[5] Jos Stam. Stable Fluids. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[6] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. 2002.

[7] A. Skillen, A. Revell, and T. Craft. Accuracy and efficiency improvements in synthetic eddy methods. *International Journal of Heat and Fluid Flow*, 62:386–394, 2016.

[8] Philipp Neumann. On transient hybrid Lattice Boltzmann–Navier-Stokes flow simulations. *Journal of Computational Science*, 17:482–490, 2016.

[9] Zi Xiang Tong and Ya Ling He. A unified coupling scheme between lattice Boltzmann method and finite volume method for unsteady fluid flow and

heat transfer. *International Journal of Heat and Mass Transfer*, 80:812–824, 2015.

[10] A Atanasov, B Uekermann, and Philipp Neumann. Anderson Accelerated Coupling of Lattice Boltzmann and Navier–Stokes Solvers for Parallel Applications. *Computation*, 4(4):38–57, 2016.

[11] Shuzo Murakami, Ryozo Ooka, Akashi Mochida, Shinji Yoshida, and Sangjin Kim. CFD analysis of wind climate from human scale to urban scale. *Journal of Wind Engineering and Industrial Aerodynamics*, 81(1-3):57–81, 1999.

[12] Yoshihide Tominaga, Akashi Mochida, Ryuichiro Yoshie, Hiroto Kataoka, Tsuyoshi Nozu, Masaru Yoshikawa, and Taichi Shirasawa. AIJ guidelines for practical applications of CFD to pedestrian wind environment around buildings. *Journal of Wind Engineering and Industrial Aerodynamics*, 96(10-11):1749–1761, 2008.

[13] J. U. Brackbill, D. B. Kothe, and H. M. Ruppel. Flip: A low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38, 1988.

[14] David E. Keyes, Lois C. McInnes, Carol Woodward, William Gropp, Eric Myra, Michael Pernice, John Bell, Jed Brown, Alain Clo, Jeffrey Connors, Emil Constantinescu, Don Estep, Kate Evans, Charbel Farhat, Ammar Hakim, Glenn Hammond, Glen Hansen, Judith Hill, Tobin Isaac, Xiangmin Jiao, Kirk Jordan, Dinesh Kaushik, Efthimios Kaxiras, Alice Koniges, Kihwan Lee, Aaron Lott, Qiming Lu, John Magerlein, Reed Maxwell, Michael McCourt, Miriam Mehl, Roger Pawlowski, Amanda P. Randles, Daniel Reynolds, Beatrice Rivière, Ulrich Rüde, Tim Scheibe, John Shadid, Brendan Sheehan, Mark Shephard, Andrew Siegel, Barry Smith, Xianzhu Tang, Cian Wilson, and Barbara Wohlmuth. Multiphysics simulations: Challenges and opportunities. *International Journal of High Performance Computing Applications*, 27(1):4–83, 2013.

[15] J Frohlich and D.a. von Terzi. Hybrid LES/RANS methods for the simulation of turbulent flows. *Progress in Aerospace Sciences*, 44(5):349–377, jul 2008.

[16] Yusuke Koda and Fue Sang Lien. The lattice Boltzmann method implemented on the GPU to simulate the turbulent flow over a square cylinder

confined in a channel. *Flow, Turbulence and Combustion*, 94(3):495–512, 2015.

[17] Yukio Tamura. WIND-INDUCED DAMAGE TO BUILDINGS AND DISASTER RISK REDUCTION. (November 2011):1–79, 2012.

[18] Bbc news. bridgewater place 'wind tunnel caused leeds injuries'. http://www.bbc.co.uk/news/uk-england-leeds-21633206. Accessed: 26-09-2020.

[19] Robert Murphy. Pedestrian Wind Comfort Analysis Report. Plot 18 Pedestrian Wind Comfort Analysis Report.

[20] W. D. Janssen, B. Blocken, and T. van Hooff. Pedestrian wind comfort around buildings: Comparison of wind comfort criteria based on whole-flow field data for a complex case study. *Building and Environment*, 59:547–562, 2013.

[21] Bert Blocken, Staf Roels, and Jan Carmeliet. Modification of pedestrian wind comfort in the Silvertop Tower passages by an automatic control system. *Journal of Wind Engineering and Industrial Aerodynamics*, 92(10):849–873, 2004.

[22] Top 12 non military uses for drones'. https://airdronecraze.com/drones-action-top-12-non-military-uses/. Accessed: 26-09-2020.

[23] Julie Pullen, Jay P. Boris, Theodore Young, Gopal Patnaik, and John Iselin. A comparison of contaminant plume statistics from a Gaussian puff and urban CFD model for two large cities. *Atmospheric Environment*, 39(6):1049–1068, 2005.

[24] F S Lien, E Yee, H Ji, a Keats, and K J Hsieh. Progress and challenges in the development of physically-based numerical models for prediction of flow and contaminant dispersion in the urban environment. *International Journal of Computational Fluid Dynamics*, 20(5):323–337, 2006.

[25] Fotini Katopodes Chow, Branko Kosović, and Stevens Chan. Source inversion for contaminant plume dispersionin urban environments using building-

resolving simulations. *Journal of Applied Meteorology and Climatology*, 47(6):1533–1572, 2008.

[26] Engineer innovation with cfd- focused multiphysics simulation. https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html. Accessed: 27-09-2020.

[27] The OpenFOAM Foundation. Openfoam.

[28] Adrian R.G. Harwood, Joseph O'Connor, Jonathan Sanchez Muñoz, Marta Camps Santasmasas, and Alistair J. Revell. LUMA: A many-core, Fluid–Structure Interaction solver based on the Lattice-Boltzmann Method. *SoftwareX*, 7:88–94, 2018.

[29] Jonas Latt, Orestis Malaspinas, Dimitrios Kontaxakis, Andrea Parmigiani, Daniel Lagrava, Federico Brogi, Mohamed Ben Belgacem, Yann Thorimbert, Sébastien Leclaire, Sha Li, Francesco Marson, Jonathan Lemus, Christos Kotsalos, Raphaël Conradin, Christophe Coreixas, Rémy Petkantchin, Franck Raynaud, Joël Beny, and Bastien Chopard. Palabos: Parallel Lattice Boltzmann Solver. *Computers and Mathematics with Applications*, (xxxx), 2020.

[30] Florian Ferstl, Ryoichi Ando, Chris Wojtan, Rüdiger Westermann, and Nils Thuerey. Narrow band FLIP for liquid simulations. *Computer Graphics Forum*, 35(2):225–232, 2016.

[31] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Transactions on Graphics*, 34(4):1–10, 2015.

[32] Zhanpeng Huang, Guanghong Gong, and Liang Han. Physically-based smoke simulation for computer graphics: a survey. *Multimedia Tools and Applications*, 74(18):7569–7594, 2015.

[33] Yongliang Feng, Pierre Boivin, Jérôme Jacob, and Pierre Sagaut. Hybrid recursive regularized lattice Boltzmann simulation of humid air with application to meteorological flows. *Physical Review E*, 100(2):023304, 2019.

[34] Stephan Lenz, Martin Schönherr, Martin Geier, Manfred Krafczyk, Andrea

Pasquali, Andreas Christen, and Marco Giometto. Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU. *Journal of Wind Engineering and Industrial Aerodynamics*, 189(October 2018):151–162, 2019.

[35] Y. Toparlar, B. Blocken, B. Maiheu, and G. J.F. van Heijst. A review on the CFD analysis of urban microclimate. *Renewable and Sustainable Energy Reviews*, 80(September 2016):1613–1640, 2017.

[36] T. van Hooff, B. Blocken, and Y. Tominaga. On the accuracy of CFD simulations of cross-ventilation flows for a generic isolated building: Comparison of RANS, LES and experiments. *Building and Environment*, 114:148–165, 2017.

[37] Sumei Liu, Wuxuan Pan, Hao Zhang, Xionglei Cheng, Zhengwei Long, and Qingyan Chen. CFD simulations of wind distribution in an urban community with a full-scale geometrical model. *Building and Environment*, 117:11–23, 2017.

[38] Hee Chang Lim, T. G. Thomas, and Ian P. Castro. Flow around a cube in a turbulent boundary layer: LES and experiment. *Journal of Wind Engineering and Industrial Aerodynamics*, 97(2):96–109, 2009.

[39] H. K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Pearson Prentice Hall, second edition, 2007.

[40] W. Zuo and Q. Chen. Real-time or faster-than-real-time simulation of airflow in buildings. *Indoor Air*, 19(1):33–44, 2009.

[41] Jonathan M. Cohen, Sarah Tariq, and Simon Green. Interactive fluid-particle simulation using translating Eulerian grids. *Symposium on Interactive 3D Graphics*, 1(212):15–22, 2010.

[42] J. Molemaker, J.M. Cohen, S. Patel, and J. Noh. Low viscosity flow simulations for animation. *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–18, 2008.

[43] 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.

[44] Rahul Narain, Jonas Zehnder, and Bernhard Thomaszewski. A Second-Order Advection-Reflection Solver. 2(2), 2019.

[45] Nelson S.-H Chu and Chiew-Lan Tai. MoXi: Real-Time Ink Dispersion in Absorbent Paper.

[46] Ehsan Fattahi, Christian Waluga, Barbara Wohlmuth, Ulrich Rüde, Michael Manhart, and Rainer Helmig. Lattice Boltzmann methods in porous media simulations: From laminar to turbulent flow. *Computers and Fluids*, 140:247–259, 2016.

[47] Dassault Systemes. Xflow.

[48] ultrafluidx overview. https://altairhyperworks.com/product/ultrafluidx. Accessed: 27-09-2020.

[49] Simscale releases gpu-based solver using lattice boltzmann method. https://www.simscale.com/blog/2018/12/lattice-boltzmann-method-solver/. Accessed: 27-09-2020.

[50] Martin Geier, Martin Schönherr, Andrea Pasquali, and Manfred Krafczyk. The cumulant lattice Boltzmann equation in three dimensions: Theory and validation. *Computers and Mathematics with Applications*, 70(4):507–547, 2015.

[51] D. Kandhai, W. Soll, S. Chen, A. Hoekstra, and P. Sloot. Finite-Difference Lattice-BGK methods on nested grids. *Computer Physics Communications*, 129(1):100–109, 2000.

[52] An upwind discretization scheme for the finite volume lattice Boltzmann method. *Computers and Fluids*, 35(8-9):814–819, 2006.

[53] Weidong Li. High order spectral difference lattice Boltzmann method for incompressible hydrodynamics. *Journal of Computational Physics*, 345:618–636, 2017.

[54] Jan Tobias Horstmann, Thomas Le Garrec, Daniel-Ciprian Mincu, and Emmanuel Lévêque. Hybrid simulation combining two space–time discretization of the discrete-velocity Boltzmann equation. *Journal of Computational Physics*, 349:399–414, 2017.

[55] P. Bhatnagar; E. Gross; M.Krook;. A model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(1), 1954.

[56] Alessandro De Rosis. Non-orthogonal central moments relaxing to a discrete equilibrium: A D2Q9 lattice Boltzmann model. *Epl*, 116(4), 2016.

[57] Christophe Coreixas, Bastien Chopard, and Jonas Latt. Comprehensive comparison of collision models in the lattice Boltzmann framework: Theoretical investigations. *Physical Review E*, 100(3):33305, 2019.

[58] Nicolas DelBosc. *Real-time simulation of indoor air flow using the lattice boltzman method on graphics processing unit.* PhD thesis, 2015.

[59] Renwei Mei, Li-Shi Luo, Pierre Lallemand, and Dominique D'Humières. Consistent initial conditions for lattice Boltzmann simulations. *Computers & Fluids*, 35(8-9):855–862, 2006.

[60] S. S Chikatamarla, S Ansumali, and I. V Karlin. Grad's approximation for missing data in lattice Boltzmann simulations. *Europhysics Letters (EPL)*, 74(2):215–221, 2006.

[61] Jonas Latt, Bastien Chopard, Orestis Malaspinas, Michel Deville, and Andreas Michler. Straight velocity boundaries in the lattice Boltzmann method. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 77(5):1–16, 2008.

[62] On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids*, 9(6):1591–1598, 1997.

[63] 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.

[64] H Tennekes and J L Lumley. *A First Course In Turbulence.* 1972.

[65] Stephen B. Pope. *Turbulent Flows*, volume 1. 2000.

[66] Pierre Sagaut. *Large Eddy Simulation for Incompressible Flows – An Introduction.* Springer, third edition, 2005.

[67] J. Smagorinsky. General Circulation Experiments With the Primitive Equations. *Monthly Weather Review*, 91(3):99–164, 1963.

[68] Massimo Germano, Ugo Piomelli, Parviz Moin, and William H. Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A*, 3(7):1760–1765, 1991.

[69] S. Hou, J. Sterling, S. Chen, and G. D. Doolen. A Lattice Boltzmann Subgrid Model for High Reynolds Number Flows. *Pattern formation and lattice gas automata*, 6:151–166, 1996.

[70] Xian Wang, Yanqin Shangguan, Naoyuki Onodera, Hiromichi Kobayashi, and Takayuki Aoki. Direct numerical simulation and large eddy simulation on a turbulent wall-bounded flow using lattice Boltzmann method and multiple GPUs. *Mathematical Problems in Engineering*, 2014, 2014.

[71] 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.

[72] M. Weickert, G. Teike, O. Schmidt, and M. Sommerfeld. Investigation of the LES WALE turbulence model within the lattice Boltzmann framework. *Computers and Mathematics with Applications*, 2010.

[73] G.R. Tabor and M.H. Baba-Ahmadi. Inlet conditions for large eddy simulation: A review. *Computers & Fluids*, 39(4):553–567, apr 2010.

[74] N. Jarrin, S. Benhamadouche, D. Laurence, and R. Prosser. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *International Journal of Heat and Fluid Flow*, 27(4):585–593, 2006.

[75] B Chopard, Joris Borgdorff, and A G Hoekstra. A framework for multi-scale modelling Subject Areas :. *Phil.Trans.R.Soc.A*, 2014.

[76] A. Mivehchi, J.C. Harris, S.T. Grilli, J.M. Dahl, C.M. O'Reilly, K. Kuznetsov, and C.F. Janssen. A hybrid solver based on efficient BEM-potential and LBM-NS models: Recent BEM developments and applications to naval hydrodynamics. *Proceedings of the International Offshore and Polar Engineering Conference*, pages 713–720, 2017.

[77] Charles Talbot, Elie Bou-Zeid, and Jim Smith. Nested Mesoscale Large-Eddy Simulations with WRF: Performance in Real Test Cases. *Journal of Hydrometeorology*, 13(5):1421–1441, 2012.

[78] Hans Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. preCICE – A fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250–258, 2016.

[79] Aditya C. Velivelli and Kenneth M. Bryden. Domain decomposition based coupling between the lattice Boltzmann method and traditional CFD methods - Part II: Numerical solution to the backward facing step flow. *Advances in Engineering Software*, 82:65–74, 2015.

[80] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[81] Thierry Morel, Florent Duchaine, Anthony Thévenin, Andrea Piacentini, Moritz Kirmse, and Eric Quémerais. Open-PALM coupler. 2019.

[82] Anthony Craig, Sophie Valcke, and Laure Coquart. Development and performance of a new version of the OASIS coupler, OASIS3-MCT-3.0. *Geoscientific Model Development*, 10(9):3297–3308, 2017.

[83] Yu Hang Tang, Shuhei Kudo, Xin Bian, Zhen Li, and George Em Karniadakis. Multiscale Universal Interface: A concurrent framework for coupling heterogeneous solvers. *Journal of Computational Physics*, 297:13–31, 2015.

[84] Derek Groen, Jaroslaw Knap, Philipp Neumann, Diana Suleimenova, Lourens Veen, and Kenneth Leiter. Mastering the scales: A survey on the benefits of multiscale computing software. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 377, 2019.

[85] Gerasimos Chourdakis. A general OpenFOAM adapter for the coupling library preCICE. (December), 2017.

[86] precice github wiki. https://github.com/precice/precice/wiki. Accessed: 24-09-2020.

[87] Jong Yeon Hwang and Kyung Soo Yang. Numerical study of vortical structures around a wall-mounted cubic obstacle in channel flow. *Physics of Fluids*, 16(7):2382–2394, 2004.

[88] Yan Meng and Kazuki Hibi. Turbulent measurments of the flow field around a high-rise building, 1998.

[89] Pietro Asinari, Taku Ohwada, Eliodoro Chiavazzo, and Antonio Fabio Di Rienzo. Link-wise Artificial Compressibility Method. (November 2011), 2011.

[90] A Revell, I Afgan, A Ali, M Camps Santasmasas, T Craft, A de Rossis, Laurence-D Holgate, J, B Iyamabo, and A et al. Mole. Coupled Hybrid RANS-LES Research at TheUniversity of Manchester. *ERCOFTAC Bulletin, European Research Community on Flow, TurbulenceAnd Combustion, 2020, Progress in RANS-based Scale-Resolving Flow Simulation Methods*, (120):pp.67, 2020.

Appendices

Sometimes it pays to stay in bed
on Monday, rather than spending
the rest of the week debugging
Monday's code.

Dan Salomon

Appendix A

Paper III - LUMA: A many-core, Fluid–Structure Interaction solver based on the Lattice-Boltzmann Method

LUMA is the original CPU based lattice Boltzmann code developed by our group at the University of Manchester, in which GASCANS is based on. GASCANS is the lattice Boltzmann software used for all the lattice Boltzmann simulations presented in this thesis.

The following journal paper is published in *SoftwareX*.

The full paper is accessible online at: https://doi.org/10.1016/j.softx.2018.02.004

SoftwareX 7 (2018) 88–94

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

LUMA: A many-core, Fluid–Structure Interaction solver based on the Lattice-Boltzmann Method

Adrian R.G. Harwood [*], Joseph O'Connor, Jonathan Sanchez Muñoz, Marta Camps Santasmasas, Alistair J. Revell

School of Mechanical, Aerospace and Civil Engineering, The University of Manchester, Sackville Street, M1 3BB, United Kingdom

ARTICLE INFO

Article history:
Received 11 January 2018
Received in revised form 13 February 2018
Accepted 21 February 2018

Keywords:
Lattice-Boltzmann Method
Finite-Element Method
Flow simulation
Fluid–structure interaction

ABSTRACT

The Lattice-Boltzmann Method at the University of Manchester (LUMA) project was commissioned to build a collaborative research environment in which researchers of all abilities can study fluid–structure interaction (FSI) problems in engineering applications from aerodynamics to medicine. It is built on the principles of accessibility, simplicity and flexibility. The LUMA software at the core of the project is a capable FSI solver with turbulence modelling and many-core scalability as well as a wealth of input/output and pre- and post-processing facilities. The software has been validated and several major releases benchmarked on supercomputing facilities internationally. The software architecture is modular and arranged logically using a minimal amount of object-orientation to maintain a simple and accessible software.

Code metadata

Current code version	v1.7.3
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-18-00007
Legal Code License	Apache License 2.0
Code versioning system used	Git
Software code languages, tools, and services used	C++, MATLAB, Python, MPI, OpenMP
Compilation requirements, operating environments & dependencies	Windows/Linux/Mac OS, C/C++ Compiler, MPI, HDF5, LAPACK, VTK
Link to developer documentation/manual	https://github.com/aharwood2/LUMA/wiki
Support email for questions	adrian.harwood@manchester.ac.uk

1. Motivation and significance

Computational Fluid Dynamics (CFD) is the science of simulating the physical behaviour of fluids using computers. It is an essential tool for design, analysis and validation. A suitable set of discretised transport equations governing the physics of the fluid are solved at discrete time-steps to produce a time-varying spatial field of physical quantities such as velocity, density and pressure. Similarly, the structural mechanics of deformable bodies can be modelled using Newton's laws of motion and solved over time, according to the level of detail required.

Fluid–Structure Interaction (FSI) is the coupled analysis of CFD with structural mechanics. Existing software implementations of these solvers vary in complexity, accuracy and speed depending on the modelling strategies chosen. Fluid dynamics solvers are generally based on either Eulerian approaches such as the Finite Volume Method (FVM) [1–4], or Lagrangian methods such as Smoothed Particle Hydrodynamics [5]. The Finite Element Method (FEM) is used widely in engineering for structural modelling [6,7]. The development of LUMA continues to be motivated by the wide range of FSI problems for which simulation is essential. In particular, the modelling of flexible filaments are an area of significant importance in the fields of aerodynamic drag reduction [8], flow control [9] and sensing [10].

In research environments, modelling and simulation software should be a capable platform for the development of new features, while also retaining simplicity in order to facilitate modification and debugging. This accelerates the uptake of the software by students with a range of programming experience and reduces barriers for meaningful contribution in, what can be, short project time frames. Full-featured, open-source engineering software [3,11] makes thorough use of object-orientated features for maximum code reuse and flexibility. However, this software can be difficult to customise for the novice user and can be an obstacle to research. LUMA has been developed for willing contributors with less experience of object-oriented programming (OOP) languages by using a logical, but simplified set of OOP features in class design.

* Corresponding author.
E-mail address: adrian.harwood@manchester.ac.uk (A.R.G. Harwood).

https://doi.org/10.1016/j.softx.2018.02.004
2352-7110/

Table 1
Table of features available since LUMA v1.7.

Features since LUMA v1.7	
Git Version Control Complete Wiki with Validation Cases	Doxygen Documentation
LBM Pull Kernel Embedded Grid Refinement [13] Load-Balancing Decomposition	Immersed Boundary Method [22] MPI Many-Core Parallelisation Body Force Calculation [23]
Bounce-Back No-Slip BCs Forced Equilibrium Velocity BCs Regularised Pressure BCs Periodic BCs	Interpolated Bounce-Back BCs [24] Regularised Velocity BCs Symmetry BCs (Specular Reflection)
Body Forcing SRT/BGK Collision Model Time-Averaging Facility	Smagorinksy Turbulence Model KBC Collision Model Restart Facility
Point Cloud Reader HDF5 Output	HDF to VTU Post-Processor Library of Point Cloud Input Files

Inheritance is kept to a single level with functions not related to physics abstracted away into manager classes. Methods and fields are intuitively named and coding standards are imposed to promote clarity over elegance.

2. Software description

Lattice-Boltzmann at The University of Manchester (LUMA) is an initiative which aims to develop novel, physical modelling for complex engineering simulation, underpinned by a flexible, but developer-friendly, many-core accelerated software framework. Development is collaborative , inclusive and centred on a simple version control process with a regular developer release schedule and continuous validation. At the heart of the initiative are applications in aerodynamics, bio-fluids and flow control.

2.1. Flow solver

The choice of flow solver is crucial for achieving desired levels of accuracy while managing complexity. The Lattice-Boltzmann Method (LBM) [12] is an alternative to traditional methods for simulating flow physics and is characterised by its simplicity. Rather than solving the Navier–Stokes as in the majority of CFD software, LBM represents fluid motion at a smaller scale using the Boltzmann equation

$$\left(\frac{\partial}{\partial t} + \vec{e}\cdot\nabla\right) f = \Omega \tag{1}$$

where f is the probability density distribution associated with a group of particles, Ω a local particle collision operator, and $\vec{e}$ a particle velocity. This equation may be discretised in space and time with spatial discretisation based on a uniform lattice of nodal locations which we refer to as a grid. If the velocity space is similarly discretised, such that groups of particles are only allowed to travel along a set of links between spatial nodes, then $f_i \in f$ where f_i is the probability of finding a particle at a given nodal location with velocity $\vec{e}_i$. The discrete, lattice-Boltzmann equation then reduces to

$$f(\vec{x} + \vec{e}_i\delta t, t + \delta t) = \Omega_i(\vec{x}, t). \tag{2}$$

The solution of Eq. (2) proceeds as a two-step process: (1) computation of the particle distributions under the collision operation Ω followed by (2) the convection of these particles to their immediate neighbours along adjoining lattice links. These two steps are referred to as the ‘collision’ and ‘streaming’ operations, respectively, in the remainder of this paper.

Variation in the local grid resolution is essential for computational efficiency; fine-grain calculations are only performed in areas of interest. In LUMA, refinement is implemented by embedding grids of higher resolution within other grids. LUMA offers both manual and automatic methods for defining the location of these grids and supports nesting to allow the construction of a grid hierarchy. A spatial and temporal refinement of factor two is applied across each transition. Time stepping proceeds on each grid at its local temporal scale with synchronisation every two cycles between grid pairs. The algorithm of Rohde et al. [13] is used to communicate populations between adjacent grids due to its simplicity and efficiency.

2.2. Structural solver

The structural solver is based on the Finite Element Method (FEM) and has been specifically designed for high aspect ratio structures undergoing large deformations (e.g. flaps, filaments, cilia) [14]. Co-rotational Euler–Bernoulli beam elements are used to represent the structure while geometric non-linearity due to large deformations is incorporated via a non-linear FEM formulation with Newton–Raphson sub-iterations. Second-order time stepping is achieved via the implicit Newmark time integration scheme. Although flexible objects are modelled exclusively using these types of elements at present, the design of the structural solver does allow other elements to be added to LUMA as required. Developers must define and implement suitable mappings for the communication of displacements/forces between the fluid and the elements chosen.

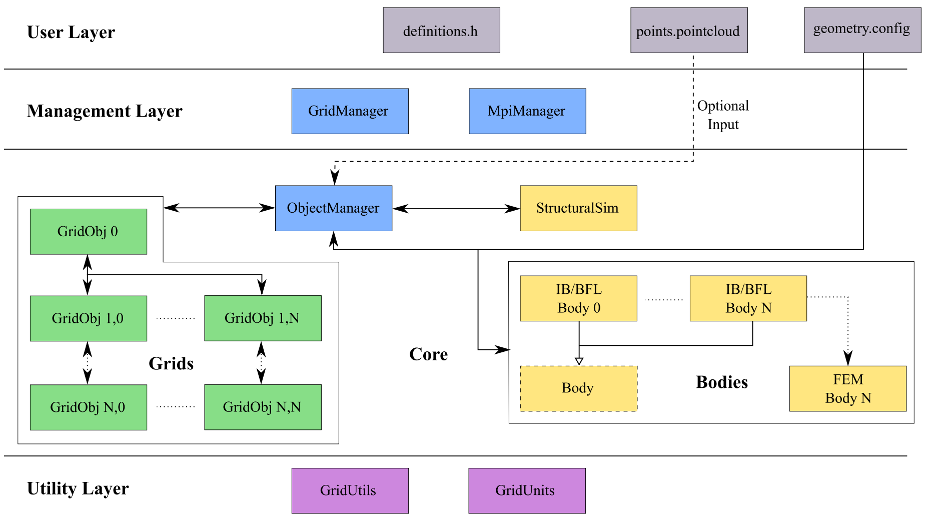
2.3. Fluid–structure coupling

The immersed boundary method (IBM) is used to enforce the no-slip condition on deformable bodies within LUMA [15,16]. The benefit of this approach is that it allows the fluid and structure to be handled separately on their own independent grids – negating the need for regular re-meshing procedures – while also facilitating large structural deformations, which are otherwise challenging with body-fitted grids. Structures are represented as a collection of surface markers with a sphere of influence. Bi-directional force information is passed between markers and fluid points within this sphere each fluid time step. For efficiency purposes, LUMA is only capable of handling cases where the fluid mesh resolution is finer than the structural mesh. This is achieved via a mapping routine between the IBM forces and the FEM structure. To ensure stability across a wide range of input conditions, a strongly coupled sub-iterative scheme is used for the FSI coupling [17]. During the sub-iteration procedure, a fixed relaxation factor is used to update the structural velocities [18] and convergence is achieved when the difference between consecutive iterations reaches a tolerance value.

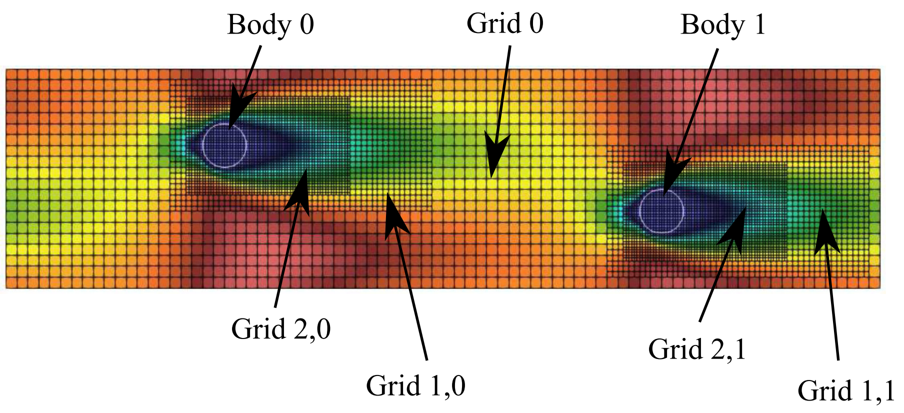
2.4. Software functionalities

LUMA is written in C/C++ and designed for an x64 machine running Linux, MacOS or Windows. The software may be compiled to run in serial or parallel depending on requirements and target platform capabilities. Parallel computing capabilities are implemented through decomposition of the problem into load-balanced blocks with added halo cells (ghost cells). Inter-block communication between halo cells uses the Message Passing Interface (MPI) [19] and multi-threading is implemented for each block using OpenMP.

Input/Output (I/O) facilities include surface mesh construction and point cloud reading and pre-processing tools. Point clouds can be generated from depth-sensing cameras or from CAD geometry, the latter using our STL to point cloud conversion tool. Parallel, binary data I/O is implemented using Hierarchical Data Format 5 (HDF5) [20] and allows the reading of initial conditions for



(a) LUMA architecture showing segregation of classes by function and the system of abstraction which allows modelling to be visible but supporting framework to be hidden from a novice developer. Flow of data is indicated by the solid arrows. There is a single layer of inheritance between the Body class and its derived children. GridObj instances can own one another to allow the construction of a hierarchy.



(b) Illustration of how multiple Body instances and multiple GridObj instances can be used to build complex problems. The grid instances are labelled based on their level in the hierarchy as well as their branch (region of refinement)

Fig. 1. The architecture of the LUMA software (top) and a visual representation of a typical problem using multiple bodies and grids and how these map onto objects in the software environment (bottom). (For interpretation of the references to colour in this figure, the reader is referred to the web version of this article.)

restarting a simulation as well as the writing of datasets for later analysis. Although HDF5 files may be read directly into scientific visualisation applications such as Matlab, and TecPlot, a complementary post-processor, custom-built using the Visualization Tool Kit (VTK) API [21], produces time-slice datasets aimed at enabling the more advanced features of the Paraview visualisation tool. The integration of these features makes LUMA a powerful, capable, integrated package for engineering flow simulation.

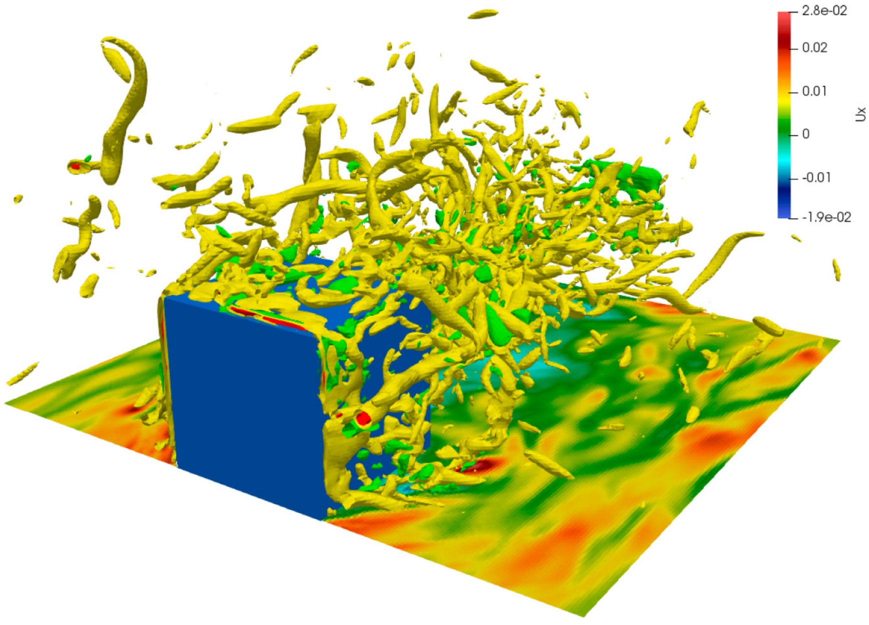
The key features of LUMA are listed in Table 1 with many of these capabilities demonstrated in the test cases contained within the online documentation.

2.5. *Software architecture*

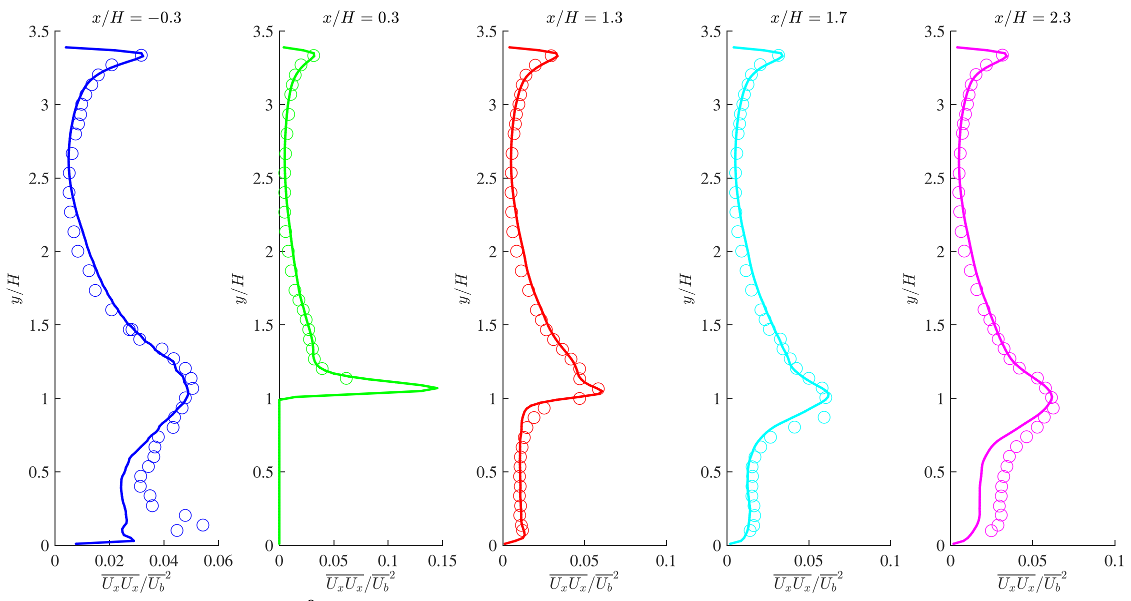
As the principal ethos of the LUMA project is simplicity and accessibility, the software architecture reflects this throughout. The class hierarchy is minimalistic and shallow. Object orientated design is used to package capabilities rather than to over-generalise objects in the hierarchy. The resulting software is then represented completely by the small number of classes depicted in Fig. 1(a). The colour-coding of the diagram indicates how capabilities are packaged by these classes.

There is a clear segregation between the fluid solver components (green) and structural solver components (yellow) and the software management framework supporting them. This structure allows engineers to edit just a single class in which all related models are stored without the need for detailed knowledge of anything else. The fundamental flow physics objects of the simulation are instances of the GridObj class. Objects of this class may be organised hierarchically to provide an intuitive structure representative of the physical domain space. These objects contain all data and methods required to perform LBM on their respective grids. Specifically, they contain 1D arrays of physical parameters at each node in the grid such as density and velocity, as well as methods for executing the different parts of the fluids algorithm on that grid. Data is stored as a structure of arrays to facilitate efficient indexing during loops. A snippet from the class header is provided for reference in the Appendix (Listing 1). Note that multiple GridObj and Body instances are permitted which facilitates simulations which require embedded grid refinement and multiple bodies as illustrated in the example in Fig. 1(b).

For run-time efficiency, most parameters such as decomposition dimensions, feature switches and fundamental physical definitions which characterise a particular problem or deployment



(a) Q-criterion illustration of vorticity in the fully-developed flow. Stream-wise velocity projected onto the floor of the channel.



(b) Stream-wise Reynolds stress $\overline{uu}/\overline{U_b}^2$ at different stream-wise locations at the lateral location $z/H = 0$ compared with experimental data.

Fig. 2. Results from using LUMA to simulate the experimental case of Meinders and Hanjalić [25].

are defined as pre-processor macros in the special header file `definitions.h`. The LUMA core must, hence, be recompiled if these compile-time options are changed. However, this process is handled automatically by the supplied makefiles to maintain ease of use. Body details are specified in the tab-separated `geometry.config` file, which allows for an arbitrary number of bodies in the same flow as well as dynamic body removal without the need to recompile. Output files are written for each grid in the simulation with snap shots appended within the same file. Typical data-flow during execution of LUMA is illustrated by the arrows in Fig. 1(a).

The `MpiManager`, `ObjectManager` and `GridManager` are classes of singleton design and act as self-contained elements with a clear remit. Each class manages many-core decomposition and communication, object processing and FSI facilitation, and flow solver with grid refinement respectively. Finally, the `GridUtils` and `GridUnits` classes are organisational classes of static utility methods including frequently-used vector and matrix operations, logging capabilities and unit conversions.

As the software is designed to run in parallel on multiple processors, structural objects are owned by a single processor in the available cluster which is responsible for performing the structural update. This allows multiple objects to be included efficiently with all structural calculations able to take place concurrently. Synchronisation is then enforced by the `ObjectManager` before the LBM continues.

Performance tests are performed regularly as part of the software development process. Strong and weak parallel efficiency test results are available in the online documentation.

3. Illustrative examples

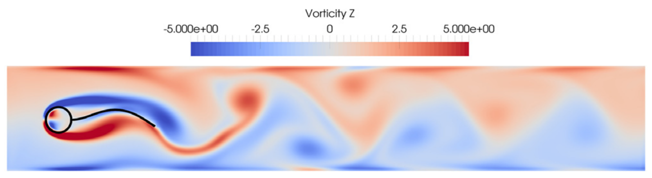
The following example cases demonstrate the key capabilities of LUMA: turbulent flow simulation and fluid–structure interaction. They may be reproduced by users following the details provided in the 'Validation' section of the online documentation.

3.1. Wall-mounted cubes in a turbulent channel

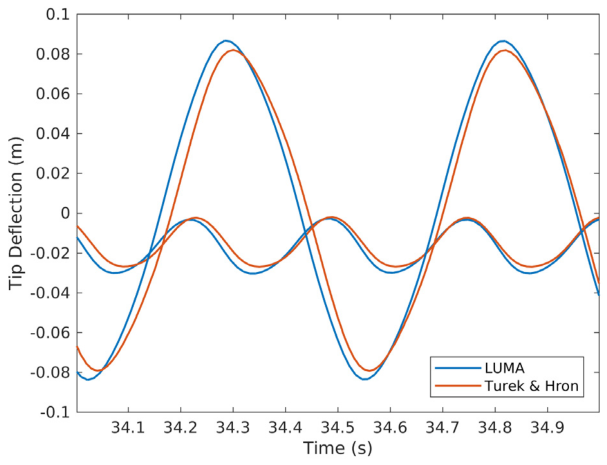
This test case is based in the wind tunnel experiment by Meinders and Hanjalić [25] and models the flow around one of the central cubes in a 25 × 10 array of cubes inside a channel. The flow Reynolds number $Re = 3850$ is based on the height of the cube H and the bulk velocity at the inlet of the channel U_b. Fig. 2(b) shows the stream-wise Reynolds stress $\overline{uu}/\overline{U_b}^2$ over five vertical lines at different positions relative to the front face of the cube. This example uses the Smagorinsky turbulence model [26] with $c_{smag} = 0.3$, the bounce-back rigid-wall boundary condition and the point cloud reader and geometry input file are used to place the cube. The post-processing is achieved using the post-processing tools supplied with LUMA and data visualised using Matlab and Paraview.

3.2. Dynamics of a flexible filament at low Reynolds number

The configuration for this 2D FSI case consists of a rigid cylinder with an attached flexible plate embedded slightly off-centre within a channel flow [27]. A parabolic velocity profile is specified at the inlet which is ramped up over the first two seconds. No-slip conditions are used at the channel walls, and the outlet boundary condition may be any of the velocity or pressure boundary conditions available in LUMA. At the intermediate Reynolds number case ($Re = 100$), the plate exhibits steady periodic flapping. Fig. 3(b) shows good agreement between LUMA and the benchmark tip displacement histories at this Reynolds number.



(a) Illustration of the vortex street behind the cylinder–filament combination. Colours represent vorticity.



(b) Comparison of tip displacement histories against benchmark data.

Fig. 3. Results from using LUMA to simulate the benchmark by Turek and Hron [27]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4. Impact

The current version of the LUMA software is the culmination of a long-term project to provide an accessible training and development platform for lattice-Boltzmann researchers. It has been deployed and benchmarked on the UK National Supercomputer ARCHER and continues to be used on this and other HPC facilities by The University of Manchester and our UK, EU and Chinese collaborators.

LUMA is currently serving as a test bed for contributing new features for the development of a new long-term software flagship project funded by the UK Engineering and Physical Sciences Research Council for the UK Consortium on Mesoscale Engineering Sciences (UKCOMES), an organisation involving over 10 research institutions across the UK.

The accessibility of the LUMA source files as well as their extensive commenting, auto-generating documentation and suite of support tools and scripts, form a capable, coherent platform for training as well as supporting publication-standard numerical studies. The complete LUMA package has enabled researchers at The University of Manchester to use the software and contribute to its development immediately with little formal support.

LUMA is suitable for the simulation of complex fluid–structure interaction and turbulent flow problems. These are some of the most challenging applications of modelling simulation in engineering which require capable software to underpin their investigation. The extended capability of LUMA such as embedded grid refinement and the independent parallelisation of fluid and structural solvers across a many-core architecture are critical enabling technologies for the pursuit of applications of flow control and bio-inspired engineering solutions.

Prior to the creation of LUMA, the number of open-source solvers with this level of capability were limited, typically lacking appropriate modelling, and difficult for a novice user to modify. The LUMA initiative and the accompanying software platform has made a significant difference to internal work flows and encouraged a more collaborative atmosphere for researchers at all levels. The support framework for LUMA is currently under further development to allow coupling to other CFD solvers. In this way, the LBM core can be coupled to traditional FVM/FEM solvers. This novel, hybrid approach to computation allows for more efficient load balancing and power efficient, accelerated calculations.

Finally, a fork of the LUMA LBM core has been used for 'interactive', mobile CFD simulations [28,29]. This is an emerging use mode of simulation and in [29], a GPU-accelerated LUMA core is used to conduct the simulations, which will be included in a future release of LUMA. The interactive fork of LUMA benefits from all the support structure of the trunk code with no modification which illustrates the modularity of the original software.

5. Conclusions

This article has introduced LUMA, a 3D many-core fluid–structure interaction solver with embedded grid refinement. The modular architecture, thorough commenting and clear coding standards have produced an accessible yet capable solver fit for future applications of modelling and simulation in engineering. The software is in use already by UK and international researchers and is the central tool for a number of UK EPSRC-funded projects. Development on the software is active with new features in the pipeline. Future plans include the merging of the GPU-accelerated fork of the LBM core to facilitate deployment on heterogeneous systems. The software is also being adapted to allow coupling with other solvers to enable the use of hybrid methodologies in CFD analysis.

Acknowledgements

This work was supported by Engineering and Physical Science Research Council Impact Accelerator Account (Grant No. EP/K503782/1) and the UK Consortium on Mesoscale Engineering Sciences (Grant No. EP/L00030X/1).

Appendix. Code sample

The fundamental class of the software is the `GridObj` class which is declared as follows.

```
class GridObj
{

// Cstor & Dstor etc... //

/************** Member Data **************/

public :

// Cell typing
IVector<eType> LatTyp;  ///< Flattened 3D array of site labels

// Grid Scalars
double dh;              ///< Dimensionless lattice spacing
int region_number;      ///< Region number
int level;              ///< Level in embedded grid hierarchy
double dt;              ///< Dimensionless time step size
double dm;              ///< Dimensionless mass reference
int t;                  ///< Number of completed iterations
double nu;              ///< Kinematic viscosity (in lattice units)
double omega;           ///< Relaxation frequency
double gravity;         ///< Gravity force
double uref;            ///< Reference velocity

// Local grid sizes
int N_lim;              ///< Local size of grid in X-direction
int M_lim;              ///< Local size of grid in Y-direction
int K_lim;              ///< Local size of grid in Z-direction

private :

/// 1D array of sub-grid pointers (size = L_NUM_REGIONS)
std::vector<GridObj*> subGrid;

// 3D or 4D arrays of nodal properties
IVector<double> f;              ///< Distribution functions
IVector<double> feq;            ///< Equilibrium distribution functions
IVector<double> fNew;           ///< Copy of distribution functions
IVector<double> u;              ///< Macroscopic velocity components
IVector<double> u_n;            ///< Velocity at start of time step (IBM)
IVector<double> force_xyz;      ///< Macroscopic body force components
IVector<double> force_i;        ///< Mesoscopic body force components
IVector<double> rho;            ///< Macroscopic density


/************** Member Methods **************/

public :

// Initialisation functions
void LBM_initGrid();                              // Grid initialiser
void LBM_initSubGrid(GridObj& pGrid);             // Initialise sub-grid
void LBM_initGridToGridMappings(GridObj& pGrid);// Initialise mappings
void LBM_initBoundLab();                          // Initialise labels for walls
void LBM_initRefinedLab(GridObj& pGrid);          // Initialise labels on refined
void LBM_initVelocity();                          // Initialise the velocity field
void LBM_initRho();                               // Initialise the density field
void LBM_addSubGrid(int RegionNumber);            // Construct a sub-grid

// IO methods
void io_restart(eIOFlag IO_flag);                 // Reads/writes restart data
void io_probeOutput();                            // Write values at probes
int io_hdf5(double tval);                         // HDF5 writer

// Master method to perform an LBM iteration using the private methods below
void LBM_multi(int subcycle = 0);


private :

// LBM implementation
void _LBM_stream(int i, int j, int k, int id, eType type_local, int subcycle);                           // Streaming step


void _LBM_coalesce(int i, int j, int k, int id, int v);                             // Get data from sub−grid

void _LBM_explode(int src_x, int src_y, int src_z, int id, int v, );                        // Give data to sub−grid

void _LBM_collide(int id);                              // Collision step
void _LBM_macro(int i, int j, int k, int id, eType type_local);                    // Update velocity and density

void _LBM_forceGrid(int id);                            // Apply body forces
void _LBM_resetForces();                                // Reset body forces to initial
double _LBM_equilibrium(int id, int v);                 // Compute feq value

// Extra models
void _LBM_kbcCollide(int id);                           // KBC version of collision
double _LBM_smag(int id, double omega);                 // Apply Smagorinsky model

// Boundary conditions
bool _LBM_applyBFL(int id, int src_id, int v, int i, int j, int k, int src_x, int src_y, int src_z);    // Apply BFL BC


bool _LBM_applySpecReflect(int i, int j, int k, int id, int v);                             // Apply Specular Reflect BC

void _LBM_regularised(int i, int j, int k, int id, eType type, int subcycle);                               // Apply Regularised BC


};
```

Listing 1: Fundamental design of the GridObj class with constructors/destructors and some short utility methods removed for brevity.

References

[1] Zaghi S. OFF, Open source Finite volume Fluid dynamics code: A free, high-order solver based on parallel, modular, object-oriented Fortran API. Comput Phys Comm 2014;185(7):2151–94.

[2] Mortensen M, Valen-Sendstad K. Oasis: A high-level/high-performance open source Navier-Stokes solver. Comput Phys Comm 2015;188(Suppl. C):177–88.

[3] OpenFOAM: The open source CFD toolbox. http://www.openfoam.com. [Accessed 7 December 2017].

[4] Fluids - CFD Simulation Software | ANSYS. http://www.ansys.com/en-gb/products/fluids. [Accessed 7 December 2017].

[5] Gomez-Gesteira M, Rogers B, Crespo A, Dalrymple R, Narayanaswamy M, Dominguez J. SPHysics - development of a free-surface fluid solver - Part 1: Theory and formulations. Comput Geosci 2012;48(Suppl. C):289–99.

[6] Abaqus Unified FEA. https://www.3ds.com/products-services/simulia/products/abaqus/. [Accessed 7 December 2017].

[7] Structure FEA Analysis | ANSYS. http://www.ansys.com/en-gb/products/structures. [Accessed 7 December 2017].

[8] Favier J, Li C, Kamps L, Revell A, O'Connor J, Brücker C. The PELskin project – part I: Fluid-structure interaction for a row of flexible flaps: A reference study in oscillating channel flow. Meccanica 2017;52(8):1767–80.

[9] Wu J, Qiu YL, Shu C, Zhao N. Flow control of a circular cylinder by using an attached flexible filament. Phys Fluids 2014;26(10):103601.

[10] O'Connor J, Revell A, Mandal P, Day P. Application of a lattice Boltzmann-immersed boundary method for fluid-filament dynamics and flow sensing. J Biomech 2016;49(11):2143–51.

[11] Palabos –CFD, Complex, Physics. http://www.palabos.org. [Accessed 7 December 2017].

[12] Chen S, Doolen GD. Lattice Boltzmann method for fluid flows. Annu Rev Fluid Mech 1998;30(1):329–64.

[13] Rohde M, Kandhai D, Derksen JJ, van den Akker HEA. A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes. Internat J Numer Methods Fluids 2006;51:439–68.

[14] Rosis AD, Falcucci G, Ubertini S, Ubertini F. A coupled lattice Boltzmann-finite element approach for two-dimensional fluidstructure interaction. Comput & Fluids 2013;86(Suppl. C):558–68. http://dx.doi.org/10.1016/j.compfluid.2013.08.004.

[15] Favier J, Revell A, Pinelli A. A lattice Boltzmann-immersed boundary method to simulate the fluid interaction with moving and slender flexible objects. J Comput Phys 2014;261(Suppl. C):145–61. http://dx.doi.org/10.1016/j.jcp.2013.12.052.

[16] Li Z, Favier J, D'Ortona U, Poncet S. An immersed boundary-lattice Boltzmann method for single- and multi-component fluid flows. J Comput Phys 2016;304(Suppl. C):424–40. http://dx.doi.org/10.1016/j.jcp.2015.10.026.

[17] Küttler U, Wall WA. Fixed-point fluid-structure interaction solvers with dynamic relaxation. Comput Mech 2008;43(1):61–72. http://dx.doi.org/10.1007/s00466-008-0255-5.

[18] Tian F-B, Dai H, Luo H, Doyle JF, Rousseau B. Fluidstructure interaction involving large deformations: 3D simulations and applications to biological systems. J Comput Phys 2014;258(Suppl. C):451–69. http://dx.doi.org/10.1016/j.jcp.2013.10.047.

[19] Message passing forum, MPI: A Message-Passing Interface standard, Tech. rep., University of Tennessee, 1994, Knoxville, TN, USA.

[20] HDF5 –The HDF Group. https://www.hdfgroup.org/solutions/hdf5/. [Accessed 7 December 2017].

[21] Schroeder W, Martin K, Lorensen B. The visualization toolkit. 4th ed. Kitware; 2006.

[22] Peskin CS. The immersed boundary method. Acta Numer 2002;11:479517.

[23] Ladd AJC. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 2. Numerical results. J Fluid Mech 1994;271:311339.

[24] Bouzidi M, Firdaouss M, Lallemand P. Momentum transfer of a Boltzmann-lattice fluid with boundaries. Phys Fluids 2001;13(11):3452–9. http://dx.doi.org/10.1063/1.1399290.

[25] Meinders E, Hanjalić K. Vortex structure and heat transfer in turbulent flow over a wall-mounted matrix of cubes. Int J Heat Fluid Flow 1999;20(3):255–67.

[26] Yu H, Girimaji SS, Luo L-S. DNS and LES of decaying isotropic turbulence with and without frame rotation using lattice Boltzmann method. J Comput Phys 2005;209(2):599–616.

[27] Turek S, Hron J. Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow. In: Bungartz H-J, Schäfer M, editors. Fluid-structure interaction: modelling, simulation, optimisation. Berlin, Heidelberg: Springer; 2006. p. 371–85.

[28] Harwood ARG, Revell AJ. Parallelisation of an interactive lattice-Boltzmann method on an Android-powered mobile device. Adv Eng Softw 2017;104(1):38–50.

[29] Harwood ARG, Revell AJ. Interactive flow simulation using Tegra-powered mobile devices. Adv Eng Softw 2018;115(Suppl. C):363–73.

First solve the problem. Then write the code.

John Johnson

Appendix B

Paper IV - GPU-Accelerated Solver for Coupled Approaches to Navier-Stokes GASCANS

The following paper presents the GASCANS lattice Boltzmann solver, which is the software used for all the lattice Boltzmann simulation presented in this thesis.

The GASCANS paper still requires review and more information in different sections including performance, coupling with other solvers and further details on the validation cases. However, the presented version serves as an overview of the structure and capacity of GASCANS.

When ready, we aim to submit the paper to the *Computer Physics Communications* journal. On the other hand, we are studying merging this paper with the NSLB paper (Paper I 4.3.1) in order to focus the current paper on the GASCANS capacity to be coupled to other codes and avoid repeated information.

GPU-Accelerated Solver for Coupled Approaches to Navier-Stokes (*GASCANS*)

Marta Camps Santasmasas[a,*], Adrian R. G. Harwood[b], Ian Hinder[b], Sijiang Fan[c], Benjamin Owen[a], Joseph O'Connor[a], Alistair J. Revell[a]

[a]*School of Engineering, The University of Manchester, Oxford Road, M13 9PL, United Kingdom*
[b]*Research IT, The University of Manchester, Oxford Road, M13 9PL, United Kingdom*
[c]*Address*

Abstract

We present *GASCANS* , an open source fluid simulation software that implements the lattice Boltzmann method (LBM) on multiple graphic processing units (GPUs) using CUDA and C++. *GASCANS* is aimed at turbulent flow and fluid structure interaction. The main novel features in *GASCANS* are that can exchange information with another computational fluid dynamics (CFD) code at run time, coupling both solvers. *GASCANS* also implements a synthetic eddy method to generate instantaneous inlet velocities for turbulent flow simulations. *GASCANS* also allows the introduction of any solid object shape via its point cloud reader. Curved boundaries and moving objects are resolved via an immersed boundary method (IBM) boundary condition. We discuss the general design principles of the code and its features before proceeding to validation test cases.

Keywords: Lattice-Boltzmann Method; GPGPU Computing; Hybrid RANS-LES; Fluid-Structure Interaction

---

PROGRAM SUMMARY

Program Title: GASCANS*: GPU-Accelerated Solver for Coupled Approaches to Navier-Stokes*
CPC Library link to program files: (to be added by Technical Editor)
Developer's repository link: (add link to GASCANS Github when we release it.)
Code Ocean capsule: (to be added by Technical Editor)
Licensing provisions: BSD 3-clause
Programming language: C++ and CUDA
Supplementary material: (Link to the wiki when available)
Nature of problem(approx. 50-250 words): Multi-GPU-accelerated simulation of incompressible turbulent flow around obstacles and moving objects. *GASCANS* can work stand alone or be coupled with other fluid solvers at run time.

*Corresponding author.
E-mail address: marta.campssantasmasas@manchester.ac.uk

Solution method(approx. 50-250 words): BGK lattice Boltzmann method with Smagorinsky large eddy simulation turbulence model and synthetic eddy method (SEM) inlet to provide instantaneous inlet velocity for turbulent flow simulations. *GASCANS* can be coupled with another solver using the preCICE [1] library. Objects are read into the simulations using a point cloud reader and moving objects are implemented using an immersed boundary method.
Additional comments including restrictions and unusual features (approx. 50-250 words): *GASCANS* can be run in both Windows and Linux, with the exception of coupling which only runs on Linux. The immersed boundary method boundary condition is not yet implemented for multi-GPU. Finished simulations can be restarted using the results written on the restart file as initial conditions for the new simulation.

References

[1] H. J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, preCICE - A fully parallel library for multi-physics surface coupling, Computers and Fluids 141 (2016) 250-258.

1. Introduction

Turbulent flows are present in a large range of engineering applications, from cardiovascular modelling to wind engineering, and often interact with moving objects. Computational fluid dynamics is a widely used tool to model fluid flows in engineering applications, however, the accuracy and quality of the information obtained from the results is closely tied to the computational resources and time available to the engineer.

Traditionally, CFD simulations for turbulent flow discretise and solve the Navier-Stokes equations using multi-CPU devices and supercomputers if available. However, in the recent years, lattice Boltzmann methods have been applied to turbulent flows (f e. [1] [2]). One of the reasons for that change is tied to the advent of more powerful and less energy consuming graphics processing unit (GPU) hardware, however, GPU code requires to be massively parallel and local for it to be efficient. The lattice Boltzmann method complies both characteristics. Mawson and Revell [3] review the optimization strategies and performance of a GPU LBM implementation.

There exist a number of available GPU accelerated LB software able to simulate turbulent flows around complex geometries. For example XFlow [4], UltrafluidX [5], TCLB [6] and WaLBerla [7] are able to read and incorporate geometry to the simulation from .stl files, or surface mesh files for WaLBerla. Regarding ease of use and case configuration, both XFlow and UltrafluidX present a graphical user interface (GUI) and do not require to be re-compiled for each different simulated case. However, both XFlow and UltrafluidX are not open source and require a commercial license. TCLB and WaLBerla are open source; TCLB can be configured using .xml files, while WaLBerla is presented as a collection of

C++ libraries and thus needs to be codded to each studied case. There are also open source LB codes aimed at implementing and testing different LB methods and presenting novel programming techniques. For example, Sailfish [8] uses Python code generation, which allows the user to code and test new LB methods without interacting with the underlying GPU architecture; and STLBM [9], a library programmed in C++17 that executes in both GPU and CPU using the same base code. It also incorporates different LB schemes and data structures, Latt et al. [9] present a comprehensive study of the performance of different LB schemes and data structures using STLBM.

The most used velocity and collision schemes are D3Q19 BGK [10] and D3Q27 cumulant [11], with TCLB, STLBM, WaLBerla and Sailfish implementing both schemes amongst other schemes; and UltrafluidX implementing D3Q27 cummulant only. Regarding turbulence modelling, UltrafluidX and XFlow implement large eddy simulation (LES) Smagorinsky models with wall functions, while Sailfish uses LES Smagorinsky without wall functions. One way to reduce the number of cells and thus the computational resources of the simulation is to refine the mesh towards the surface of the geometry, where more flow detail is needed. WaLBerla, XFlow and UltraFluidX offer octree mesh refinement. WaLBerla and Sailfish divide the mesh in blocks that can be distributed following the geometry, thus avoiding to set LB cells in regions of the mesh occupied by the geometry.

The capabilities to solve multi-scale and multi-physics problems with the reviewed software are limited. TCLB can be coupled with a series of predefined discrete element method (DEM) solvers in order to incorporate rigid moving objects into the simulation; WaLBerla includes a DEM solver for the same purpose. However, to the extend of our knowledge, there is no open source GPU LB software able to transfer and receive velocity data to/from another solver at run time.

This paper presents *GASCANS* , a GPU accelerated open-source lattice Boltzmann solver aimed at turbulent flow with complex geometries and moving objects. GASCANS is able to extract and incorporate boundary conditions velocity during run-time and thus be coupled with other solvers to further take advantage of the heterogeneous CPU-GPU architecture of consumer computers; also to reduce the size of the domain simulated by LB and thus reducing the cost of the complete simulation. It also incorporates an LES Smagorinsky turbulence model and allows the inlet velocity to be generated from mean flow data using a synthetic eddy method (SEM). From the software point of view, *GASCANS* is formed by a core library and an application. The application part can be changed to meet the user's needs and the library case can be expanded to include new methods and models using the provided `IFeature` template.

GASCANS fills an niche in the market of open-source GPU accelerated lattice Boltzmann solvers, providing ready to use and adaptable turbulent flow and moving objects modelling aimed at engineers with limited or no access to large computer systems.

2. Software Design

GASCANS consists of two parts – the *library* and the *application*. The former encapsulates the mathematical algorithms, their host and device implementation and defines a public-facing interface called `Interface` and a set of definition classes within `SimData`. The utility of this solution means that *GASCANS* can be used by a wide variety of front-end applications. The application part of *GASCANS* is a console application that leverages the public-facing part of the library to run a simulation. However, the application using the *GASCANS* library could be a custom graphical user interface. In [12], the *GASCANS* library was leveraged from an application class built within a game engine.

Software on GPU devices follows a predictable pattern based on GPU-programming APIs. Namely:

1. Allocate memory on the host
2. Initialise the data on the host
3. Allocate memory on the device
4. Copy data from host to device
5. Compute on the device and update device memory
6. Copy data back from device to host
7. Post-process host data and repeat

GASCANS provides the base class `IFeature`, which provides methods for the GPU-programming steps in the previous list. These methods ensure that any class that inherits from `IFeature` will be compatible with the *GASCANS* struture and multi-GPU implementation. Thus `IFeature` acts as a template to add new models and add-ons to *GASCANS* base code.

GASCANS is designed for single-node, multi-GPU use. This implementation allows the use of multi-GPU APIs within the CUDA framework while not requiring the additional complexity of MPI to managed multi-node communication, which makes *GASCANS* especially appealing to engineers with limited or no access to large computing clusters. Users of *GASCANS* may configure the library to suit their system, running either a single GPU or multiple GPUs. Furthermore, the library interface can be leveraged by users who wish to use the provided application component or just as easily by those wishing to build their own.

2.1. Architecture

Figure 1 illustrates the class hierarchy and aggregation. The core LB classes encapsulates the main LBM behaviours, including the LES turbulence model and the immersed boundary method, along with the state of the *domain grid* – the grid of cells covering the entire domain – and the `Interface` class, which governs the high-level steps described above and exposed public methods for invoking the initialisation, calculation and back-copy steps. When using multi-GPU configurations, the `Grid` defines and maintains the one-to-many relationship between the domain grid and the *problem blocks* – a sub-section of the

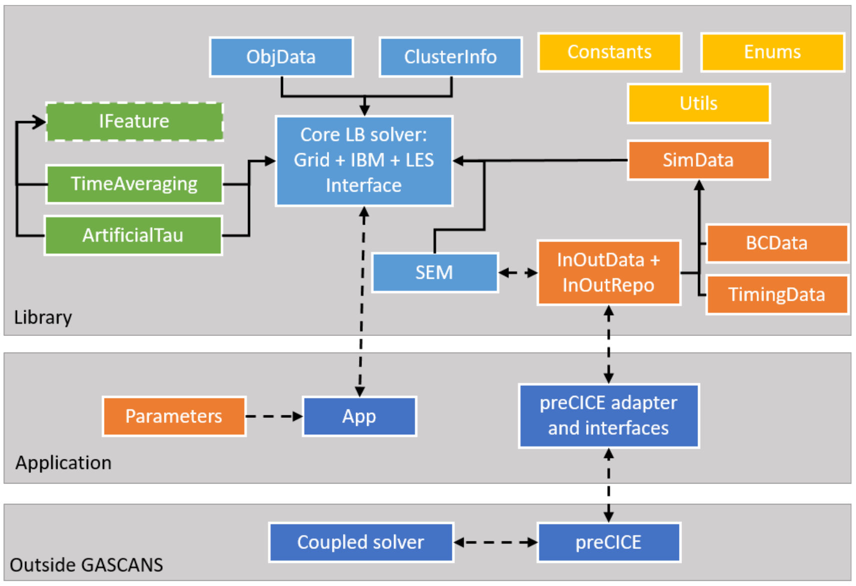


Figure 1: Illustration of the two-part architecture of *GASCANS* and the relationships between components. An open arrow indicates inheritance, a solid arrow indicates aggregation and a dashed arrow indicates a primary communication relationship. The isolated boxes represent definitions and utilities accessible by the core LB solver.

domain grid plus a set of halo cells associated with a given device. The overall state of the multi-GPU configuration is maintained by the `ClusterInfo` class. The user input and configuration is defined by the `Parameters` class, which handles the configuration files, and the `SimData` class, which is an aggregation of structures that allow an application to define a simulation completely depending on the features required.

The classes that allow *GASCANS* to be coupled with an external solver are `InOutData` and `InOutRepo` in the Library part, and `Adapter` and `PreciceInterface` in the Application part. The `InOutRepo` and `InOutData` classes acts as a link between the data stored in the grid and the external world. Each instance of `InOutRepo` stores and handles data at set coordinates within the simulation domain and it is marked as data extracted from *GASCANS* that needs to be send to an external solver, or data from an external solver that needs to be introduced into *GASCANS* grid. `InOutData` aggregates all the `InOutRepo` needed for a simulation (for example each `InOutRepo` stores data at a boundary). `Adapter` and `PreciceInterface` form the interface with the coupling library preCICE [13]; which controls the coupling algorithm and exchange of information with the coupled solver(s).

All other classes within figure 1 simply provide definitions and utilities.

2.2. Multi-GPU Decomposition

Decomposition and definition of the CUDA kernel launch parameters is defined in a different way for 2D and 3D problems to maximise memory access efficiency. Data is stored in memory using the X, Y, Z order. Therefore, to ensure data is moved in a contiguous block for maximum efficiency, the domain is decomposed in a 1D sense. For 2D problems, the domain cells are grouped into 1D problem blocks in the Y-direction. For 3D problems, the data is grouped into 2D problem blocks in the Z-direction. The number of problem blocks is controlled by the user and may or may not match the number of devices. This allows some flexibility in deploying the simulation on a multi-GPU cluster. For example, if a user requests 4 blocks and 4 devices but only 3 devices are available then *GASCANS* will decompose the domain into 4 problem blocks but distribute them over 3 devices, with the last device computing two blocks.

Each problem block is defined to be of a uniform size with the final block accumulating any additional cells if the domain is not exactly divisible. When calculating the local block size, if the user requests more than one block, halos of a width defined in the library definitions source will be added to either end of the blocks. This halo region represents an overlap of cells between adjacent blocks and is used to exchange information between adjacent blocks during synchronisation. Note that the frequency with which synchronisation is performed is directly proportional to the thickness of the halo region. Narrow halos have a reduced memory footprint but require more frequent communication of smaller amounts of data. The communication of data between GPUs uses the device-to-device memory copy API. If supported by the hardware, this allows direct communication of information between device memories without using the host as a relay. The variation in performance of the LBM algorithm with halo thickness has been reported in [14].

The kernel launch parameters are defined once the decomposition and block distribution is known. The CUDA framework defines the launch parameters hierarchically. Threads are grouped into 32-thread *warps* which are arranged over a 3D *thread block* which are arranged over a 3D *grid*. To maximise alignment between the threads and memory, it is prudent to maximise these dimensions in the X-direction. The distribution of threads is performed by an iterative algorithm which uses the user-specified number of warps per block as well as the number of cells in each problem block to define an optimal 3D arrangement which satisfies all the constraints. In 2D the grid and thread block dimensions are equal to 1. The algorithm used is described in Listing 1. Note that if the number of cells in the X-direction of the problem block nX is less than the number of threads desired ($32 \times$ WARPS_PER_BLOCK) then the thread block dimensions [X,Y,Z] are simply equal to [nX, 1, 1]. Otherwise, the algorithm is used to determine the optimal Y and Z dimensions.

Listing 1: Algorithm used to define the kernel launch parameters

```

	/*
	Occupancy is here defined as the number of threads that have a
```

```
    corresponding cell to compute
    */
    var occupancy = 0;

    /*
    Temporary variables to store thread block dimensions (B) and grid
    dimensions (G)
    */
    var tmpBx, tmpBy, tmpGx, tmpGy, tmpBz, tmpGz;

    // Iterate on block y-dimensions and z-dimensions
    tmpBy = 0;
    do
    {
       // Try next integer value of bY
       tmpBy += 1;

       // Reset bZ
       tmpBz = 0;

       do
       {
          // Try next integer value of bZ
          tmpBz += 1;

          /*
          Compute number of threads in X direction in block
          */
          tmpBx = 32 * G_WARPS_PER_BLOCK / ()tmpBy * tmpBz);

          /*
          If not a integer value then not a complete number of warps so
          continue
          */
          if (Abs(tmpBx - Ceiling(tmpBx)) > 0)
             continue;
          else
          {
             tmpBx = Ceiling(tmpBx);
             tmpGx = Ceiling(nX / tmpBx);
             tmpGy = Ceiling(nY / tmpBy);
             tmpGz = Ceiling(nZ / tmpBz);

          /*
          Store config if better occupancy and X is still the largest
          dimension
          */
          var tmpOcc = (nX * nY * nZ) /
          (tmpBx * tmpBy * tmpBz * tmpGx * tmpGy * tmpGz)
          if (tmpOcc > occupancy &&
             tmpBy < tmpBx && tmpBz < tmpBx)
          {
             occupancy = tmpOcc;
             bX = tmpBx;
             bY = tmpBy;
             bZ = tmpBz;
             gX = tmpGx;
```

```
            gY = tmpGy;
            gZ = tmpGz;
         }
      }
   } while (
      tmpBz <= nZ &&
      tmpBz < maxThreadsPermittedZ &&
      tmpBz < tmpBx);
} while (
   tmpBy <= nY &&
   tmpBy <= maxThreadsPermittedY &&
   tmpBy < tmpBx);
```

2.3. Point Cloud Reader

GASCANS is capable of accepting geometry prescribed as a cloud of point in 3D Cartesian space. This is particularly useful for coupling to 3D object scanners for which a point cloud is the standard output format. The point cloud data file compatible with *GASCANS* comprises a list of comma-separated positional information (x,y,z). The user specifies in the geometry configuration file the name of the point cloud data file along with the desired position and scale of the object. At runtime, *GASCANS* reads in the cloud of points and performs a spatial shift and scale transformation on the coordinates, computing its bounding box within the problem domain.

Objects that use the bounce-back boundary condition require lattice cells to be labelled as part of the object. *GASCANS* is then able to apply the boundary condition between adjacently labelled fluid-solid cells. When building a bounce-back object, *GASCANS* will sub-sample the relevant point cloud data using a voxel grid filter. The logic to determine whether a cell contains a point, and should thus be labelled as a solid cell. The behaviour applied to point cloud processing is describe in Listing 2 and illustrated in 2. Note that it is necessary to shrink the calculated length of the object for the purposes of labelling: if the object length was exactly divisible by the cell spacing then the resulting object is one cell larger than it should be.

Listing 2: Logic for deciding whether a point is within a particular cell

```
/*
Assumed definitions:
all_points = the collection of points in the cloud
point.x, point.y, point.z = the point position in original units of the file
length = the desired length in dimensionless units
sx, sy, sz = the spatial position of the start or centre of the bounding box
     of the object
dx = the cell spacing (assume same spacing in all dimensions)
delta = a small number defined as 10^-7
*/

/*
Correct the length of the object to ensure it is not an exact number of cells
*/
var length_mod = length - delta * dx;
```

```

/*
Compute the scaling factor (assumed here scaling with respect to X but could
     be any dimension)
*/
var scale = length_mod / (maxX(all_points) - minX(all_points));

/*
Loop through the points
*/
foreach (point in all_points)
{
/*
Calculate the shift required to star the object at the correct location
*/
var shiftX = sx - scale * minX(all_points);

/*
Calculate the new point
*/
var localX = (point.x * scale) + shiftX;

// Repeat for other dimensions...

/*
Compute the position to the nearest cell and round down to the lower
edge of that cell
i, j, k = the cell indices in the grid
*/
var i, j, k;
i = floor(localX / dx);
j = floor(localY / dx);
k = floor(localZ / dx);

/*
If the rounded point rests on the lower edge up to but not including the
     upper edge of the cell then the cell can be labelled as solid
*/
cell.type[i,j,k] = CellType.Solid;
}
```

2.4. Transfer data from/to external sources

GASCANS is able to read velocity data generated at the application level (Fig. 1) and incorporate it as a velocity boundary condition during run time. It is also able to extract velocity data from user specified coordinates within the domain and use it at the application level and/or transfer it to other applications.

GASCANS uses the `InOutData` and `InOutRepo` classes to incorporate and extract data to/from the GASCANS mesh. GASCANS is agnostic to where the data in `InOutData` comes from. Thus `InOutData` can be used to incorporate data from a different range of sources, for example coupled simulations or files, without modifying the GASCANS source code.

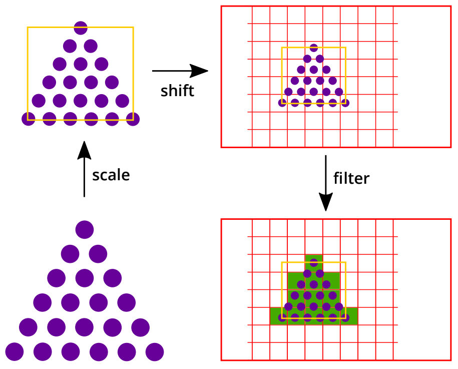


Figure 2: Illustration of the point cloud process. The bounding box of the scaled cloud is shown in yellow. The domain and grid of cells is shown in red. The cells labelled by the process as *solid* are coloured in green.

`InOutData` contains a vector of `InOutRepo`; one `InOutRepo` for each mesh section to extract data from or insert data to. Listing 3 shows the structure of an `InOutRepo` object. The `coordinates_` vector stores the world coordinates of the data to be incorporated or extracted to/from GASCANS. `vectorData_` and `scalarData` store the values of the variables to be transferred together with the name of each variable.

Listing 3: Excerpt from the InOutRepo class

```
// Data to send into and out
// of the underlying LBM simulation
template <typename T>
class InOutRepo
{
public:
//...//

// InOutRepo data needs to be written to LBM mesh
bool bReadyToWriteToLBM = false;

// LBM mesh data needs to be written to InOutRepo
bool bReadyToReadFromLBM = false;

//... Methods to fill in and access the data
// stored in InOutRepo ... //

private:
// CPU storage
std::vector<T> coordinates_;
std::map<std::string, std::vector<T>> vectorData_;
std::map<std::string, std::vector<T>> scalarData_;
int numDataPoints_ = -1;

// GPU storage.
// There is one pointer for each GPU (decive)
std::map<std::string,std::vector<T*>> d_vectorData_;
```

```
    std::map<std::string,std::vector<T*>> d_scalarData_;
    std::vector<T*> d_coordinates_;

    // ... //

    // Offset between world coordiantes
    // and GASCANS coordinates
    T x0_ = 0;
    T y0_ = 0;
    T z0_ = 0;
    }
```

The information stored in `InOutData` can be updated at run time from another solver connected to *GASCANS* via preCICE [13]. preCICE is an external open source coupling library written mainly in C++ [1]. The main functions of preCICE when used to couple *GASCANS* with another code are to control the time loop of the coupling solvers, transfer the data that needs to be exchanged to the receiving solver and interpolate the data from the sending solver mesh to the receiving solver mesh. Fig. 3 shows an scheme of how *GASCANS* runs coupled to another solver via preCICE.

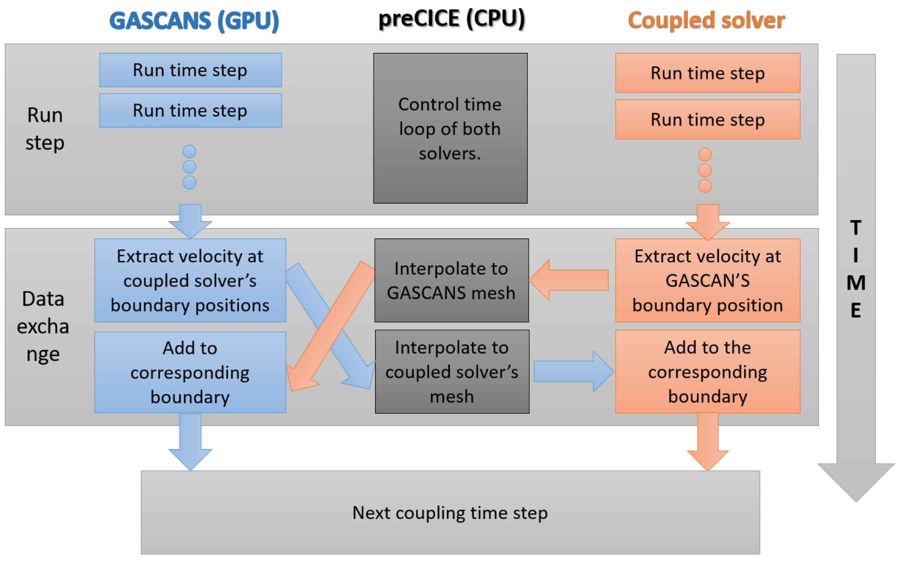

GASCANS (GPU)
preCICE (CPU)
Coupled solver
Run step
Run time step
Run time step
Control time loop of both solvers.
Run time step
Run time step
Data excha nge
Extract velocity at coupled solver's boundary positions
Add to corresponding boundary
Interpolate to GASCANS mesh
Interpolate to coupled solver's mesh
Extract velocity at GASCAN'S boundary position
Add to the corresponding boundary
Next coupling time step
T I M E

Figure 3: Algorithm of the two-way coupling of *GASCANS* with another solver.

GASCANS is connected to preCICE at Application level via the `preCICE adapter` component, which is based on the OpenFOAM preCICE adapter developed by Chourdakis [?]. It contains two main classes:

[1]More information and tutorials about preCICE in `https://www.precice.org/`

- *Adapter class*: Information to configure the coupling on *GASCANS* side, which mainly is:
  - Pointer to the `precice::SolverInterface` object, which links to preCICE.
  - `InOutData` object that stores the data ready to be sent to preCICE or received from preCICE. It contains an `InOutDataRepo` for each mesh section coupled to preCICE.
  - One *PreciceInterface* object for each coupled mesh section.
- *PreciceInterface class*: Links the data contained in `InOutRepo` with the coupled interface in preCICE. Including:
  - Pointer to the `precice::SolverInterface` object, which is the link to preCICE.
  - Mesh information for the coupled cells: name of the coupled mesh, meshID, and name of the file that contains the world coordinates [2] of the cells in the coupled mesh.
  - Names of the variables to read from preCICE into the `InOutRepo` object corresponding to the interface. This is the data that flows from preCICE to GASCANS.
  - Names of the variables to write to preCICE from the `InOutRepo` corresponding to the interface. This is the data that flows from GASCANS to preCICE.

Algorithm 1 shows the *GASCANS* algorithm while coupled with another solver and Algorithm 2 shows the details of how the data is incorporated / extracted to / from the LB main loop.

2.5. Features

GASCANS allows the creation of new features in a modular fashion, new feature classes need simply implement the common `IFeature` interface. This interface provides the methods in Listing 4. The `TimeAveragedQuantities` and `ArtificialTau` classes are considered features and hence implement this interface. `TimeAveragedQuantities` calculates the averaged lattice Boltzmann velocity at run time (section 3.6) and `ArtificialTau` implements an sponge layer at the outlet of the domain (section 3.7).

Listing 4: Methods provided by the common feature interface

```
/// <summary>
/// Called when the number of problem blocks is known
/// </summary>
/// <param name="blocks">number of problem blocks</param>
```

[2]coordinate system shared by pisoFoam and GASCANS

Algorithm 1 Algorithm for GASCANS coupled with another solver emphasizing the coupling steps.

```
1: Step 1: Initialisation
2: for all interfaces, i do
3:     Write the world coordinates of the GASCANS mesh in a file.
4: end for
5: Instantiate and initialise an Adapter object using precice-config.xml and
   preciceConfig.yml.
6: for all interfaces,i do
7:     Add a PreciceInterface object with the information read from preciceConfig.
8:     Add an InOutRepo to the InOutData object.
9: end for
10: Initialise the precice::SolverInterface object.
11: Exchange initial data if needed.
12: Add the offset between world coordinates and GASCANS coordinates to all
    the InOutRepos in InOutData
13:
14: Step 2: Simulation
15: while preCICE is coupling do
16:     runTimeStep(InOutData)
17:     for all interface.writeData, writeData do
18:         Transfer writeData from InOutData.InOutRepo[writeData] to preCICE.
19:         Mark InOutData.InOutRepo[writeData] as ready to write to LBM.
20:     end for
21:     Advance preCICE (It sends the write data and receives the read data)
22:     for all interface.readData, readData do
23:         Transfer readData from preCICE to InOutData.InOutRepo[readData].
24:         Mark InOutData.InOutRepo[readData] as ready to read from LBM.
25:     end for
26: end while
27:
28: Step 3: Finalise
29: Clean up and destroy the preCICE and GASCANS objects.
```

```
    virtual void setSize(int blocks) = 0;

    /// <summary>
    /// Called when the number of cells for the domain is known
    /// </summary>
    /// <param name="nX">number of cells in X</param>
    /// <param name="nY">number of cells in Y</param>
    /// <param name="nZ">number of cells in Z</param>
    virtual void initialise(int nX, int nY, int nZ) = 0;

```

Algorithm 2 Algorithm for the runTimeStep(InOutData)

```
 1: for all InOutRepo in InOutData, repo do
 2:     if repo.isSEM then
 3:         Update SEM velocity
 4:     end if
 5:     if repo.bReadyToWriteToLBM then
 6:         Copy vectorData_[velocity] to d_vectorData_[velocity]
 7:         Introduce d_vectorData_[velocity] to GASCANS mesh
 8:     end if
 9: end for
10: GASCANS time step (stream and collide)
11: for all InOutRepo in InOutData, repo do
12:     if repo.bReadyToReadFromLBM then
13:         Extract velocity from GASCANS mesh to d_vectorData_[velocity]
14:         Copy d_vectorData_[velocity] to vectorData_[velocity]
15:     end if
16: end for
```

```cpp
    /// <summary>
    /// Called to allocate any host pointers
    /// </summary>
    /// <param name="nSize">total number of cells in the domain</param>
    virtual void allocateOnHost(int nSize) = 0;

    /// <summary>
    /// Called to allocate any device pointers
    /// </summary>
    /// <param name="block">the problem block of interest</param>
    /// <param name="pitch">[output] the require pitch of the padded memory based
        on data type</param>
    /// <param name="width">the width of the memory allocation</param>
    /// <param name="height">the height of the memory allocation</param>
    virtual void allocateOnDevice(int block, size_t& pitch, size_t width, size_t
        height) = 0;

    /// <summary>
    /// Called to initialise device memory from host memory
    /// </summary>
    /// <param name="block">the problem block of interest</param>
    /// <param name="deviceOffset">offset into the device pointer memory to start
        the copy</param>
    /// <param name="hostOffset">offset into the host pointer memory to start the
        copy</param>
    /// <param name="pitchElems">pitch in terms of number of elements used in
        combination with the offset to find correct starting byte</param>
    /// <param name="pitchBytes">pitch in terms of the bytes to calculate correct
        stride</param>
    /// <param name="width">width of 2D data</param>
    /// <param name="height">height of 2D data</param>
    /// <param name="stream">stream through which to pass the data</param>
    virtual void initialiseDeviceFromHost(int block, int deviceOffset,
    int hostOffset, int pitchElems, size_t pitchBytes, size_t width,
    size_t height, cudaStream_t& stream) = 0;

    /// <summary>
    /// Called to copy data from device memory back to host memory
    /// </summary>
    /// <param name="block">the problem block of interest</param>
    /// <param name="deviceOffset">offset into the device pointer memory to start
           the copy</param>
    /// <param name="hostOffset">offset into the host pointer memory to start the
           copy</param>
    /// <param name="pitchElems">pitch in terms of number of elements used in
           combination with the offset to find correct starting byte</param>
    /// <param name="pitchBytes">pitch in terms of the bytes to calculate correct
           stride</param>
    /// <param name="width">width of 2D data</param>
    /// <param name="height">height of 2D data</param>
    /// <param name="stream">stream through which to pass the data</param>
    virtual void backCopy(int block, int deviceOffset, int hostOffset,
    int pitchElems, size_t pitchBytes, size_t width, size_t height,
    cudaStream_t& stream) = 0;
```

3. Lattice Boltzmann method and models implemented in *GASCANS*

This section describes the models implemented in *GASCANS* and thus the physical phenomena *GASCANS* is able to model.

3.1. BGK Lattice Boltzmann Method

GASCANS implements a 3DQ19 BGK lattice Boltzmann method (LBM) as its core solver.

The lattice Boltzmann method solves for the particle distribution functions f_α, which represent the probability of a group of fluid particles at a position $\mathbf{x}$ and time t to move at a velocity $\mathbf{c}_i$. The particle velocity space is discretised; the number of discretised particle velocities depend on the lattice model. *GASCANS* implements a 3DQ19 lattice model; its discretised particle velocities are:

$$\mathbf{c}_i = \begin{cases} 0, & i = 0, \\ (\pm1,0,0),(0,\pm1,0),(0,0,\pm1) & i = 1 \quad to \quad 6, \\ (\pm1,\pm1,0),(\pm1,\pm0,\pm1),(0,\pm1,\pm1) & i = 7 \quad to \quad 18. \end{cases} \tag{1}$$

Given distribution functions f_i at a time t, the LBM equation,

$$\begin{aligned} f_i(\mathbf{x}+\mathbf{c}_i, t+\Delta t) = f_i(\mathbf{x},t) - \Delta t/\tau \left[f_i(\mathbf{x},t) - f_i^{(\mathrm{eq})}(\mathbf{x},t) \right] \\ + \Delta t F_i(\mathbf{x},t)\,. \end{aligned} \tag{2}$$

is used to compute their values at a time $t + \Delta t$. τ is the BGK relaxation timescale and the BGK equilibrium distribution $f_i^{(\mathrm{eq})}$ is

$$f_i^{(\mathrm{eq})} = \rho\omega_i \left[1 + \frac{\mathbf{c}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{c}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u}^2}{2c_s^2}\right] \tag{3}$$

where $\omega_0 = 1/3$, $\omega_i = 1/18$ for $i = 1$ to 6 and $\omega_i = 1/36$ for $i = 7$ to 18 are the weight coefficients of the 3DQ19 lattice and $c_s = 1/\sqrt{3}$ is the lattice sound speed. Given a force density $\mathbf{F}$ acting on the fluid, the force contribution F_i in Eq. 2 is defined by

$$F_i = \left(1 - \frac{\Delta t}{2\tau}\right)\omega_i \left[\frac{\mathbf{c}_i - \mathbf{u}}{c_s^2} + \frac{\mathbf{c}_i \cdot \mathbf{u}}{c_s^4}\mathbf{c}_i\right] \cdot \mathbf{F}\,. \tag{4}$$

The macroscopic fluid density and velocities are given by

$$\rho = \sum_i f_i \tag{5}$$

$$\rho\mathbf{u} = \sum_i \mathbf{c_i} f_i + \Delta t/2\,\mathbf{F}\,. \tag{6}$$

All the presented equations that form the lattice Boltzmann units are in lattice units. Lattice units measure the space in number of cells and the time in number of time steps; thus the macroscopic velocity $\mathbf{u}$ represents how many cells cells the fluid covers in one time step. It is also convenient to set $\Delta t = 1$ time step, which allows for each of the discretised particle velocities $_i$ to reach the neighbouring cell in one time step.

It can be shown [15] that the lattice Boltzmann equation reproduces the Navier-Stokes equations for low Mach number flows with the following pressure p and kinematic viscosity ν:

$$p = c_s^2 \rho \frac{\delta x^2}{\delta t^2} \tag{7}$$

$$\nu = \frac{1}{3}\left(\tau - \frac{\Delta t}{2}\right)\frac{\delta x^2}{\delta t} \tag{8}$$

where δx is the lattice boltzmann cell size in the Navier-Stokes units (i e. meters or dimensionless) and δt is the lattice Boltzmann time step in the Navier-Stokes units (i e. seconds or dimensionless).

The algorithm in *GASCANS* proceeds as follows. The first step is to *stream* the distribution functions at time t from appropriate neighbouring lattice sites to compute the “streamed” distribution function f_i^*,

$$f_i^*(\mathbf{x}, t + \Delta t) = f_i(\mathbf{x} - \mathbf{c_i}\Delta t, t)\,, \tag{9}$$

and no other information from time t is used. All further quantities in this section are considered to be evaluated at position $\mathbf{x}$ and time $t + \Delta t$, so these labels are omitted for brevity.

The macroscopic density ρ and velocity $\mathbf{u}$ are then computed from the streamed distribution functions as

$$\rho = \sum_i f_i^* \,, \tag{10}$$

$$\rho\mathbf{u} = \sum_i \mathbf{c_i} f_i^* + \Delta t/2\mathbf{F} \tag{11}$$

Finally, the LBM collision step, Eq. 2, is performed in order to compute the new distribution functions f_i at time $t + \Delta t$,

$$f_i = f_i^* - \Delta t/\tau \left[f_i^* - f_i^{(\mathrm{eq})} \right] + \Delta t F_i \,. \tag{12}$$

In summary, the algorithm is

1. Stream f_i from neighbouring lattice sites according to Eq. 9;
2. Compute the macroscopic density and velocity (Eq. 10-11);
3. Collide the distribution functions as in Eq. 12.

3.2. Boundary Conditions

GASCANS provides two boundary conditions to fix the velocity and density at the specified boundary:

- **Regularised velocity** [16] sets the macroscopic velocity to the user input calculates the corresponding pressure and deduces and corrects the particle distribution functions at the cells.

- **Forced equilibrium** sets the macroscopic velocity to the user input and macroscopic density to one and sets the corresponding particle distribution functions to their equilibrium values (eq. 3)

GASCANS provides three boundary conditions to define walls through which the flow cannot penetrate:

- **Regularised velocity** [16] with the macroscopic velocity set to zero.

- **Half-way bounce-back** [15] which sets a no slip boundary condition (i e. velocity equal zero at the wall). The wall is situated halfway between the centre of the bounce back boundary cell and the centre of the first cell of the domain.

- **Symmetry** [15] sets the boundary as a slip wall, the velocity next to the wall is symmetric to the velocity at the wall, hence the fluid can not go through the wall.

For open boundaries (i e. boundaries that allow the flow to go through them) *GASCANS* provides:

- **Periodic** is the default boundary condition and streams the particles that would exit the domain into the incoming particles for the opposite boundary.
- **Extrapolated** set the missing particle distribution functions to the values of the particle distribution functions on the neighbour cell in the stream direction. Its effect approximately a zero gradient for pressure and velocity.

3.3. Body Forcing

The flow may be accelerated in the direction of the X-axis through specification of a body force value. This value may model a pressure gradient across the domain or a global force terms such as gravity. The forcing is included in the LBM using the approach of Guo et al. [17]. Assuming a force in lattice units of f_x, this is implemented as a two-step correction: during the streaming step, the momentum in the X-direction is augmented by the additional momentum due to the force $\frac{1}{2}\rho f_x$; then, as part of the collision step, the force contribution along the v-th lattice link f_v is added to the population. Mathematically these steps are represented as:

$$\rho u_x \mapsto \rho u_x + \frac{1}{2}\rho f_x \tag{13a}$$

$$f_v = \frac{w_v}{c_s^2}\left(1 - \frac{\omega}{2}\right)\rho f_x \left[c_{x,v}\left(1 + \frac{\vec{c}_{i,v}\vec{u}_i}{c_s^2}\right)\right] \tag{13b}$$

where i represents the Cartesian direction, w is the link weighting, ω is the relaxation frequency, c_s is the lattice sound speed, $\vec{u}$ is the macroscopic velocity, and $\vec{c}$ is the lattice link directions.

3.4. Immersed Boundary Method

GASCANS has the capability to simulate the flow of a fluid around an immersed solid body. The fluid and body velocities must be matched at the boundary between them. The traditional approach to this problem is to modify the fluid grid to conform to the shape of the immersed body, which may be arbitrary, and to exchange force information across the boundary. This is complicated to implement, and when the body deforms or moves, the fluid grid needs to be recomputed and fluid data interpolated to the new cell locations, leading to a significant cost in performance. In contrast, Peskin [18] introduced the Immersed Boundary Method (IBM), in which the fluid is evolved throughout the entire domain on a simple regular grid, and the presence of the solid body is represented via a forcing term in the fluid evolution equations equal to the force between the body and the fluid such that their velocities are equal at the boundary of the body. The IBM method in *GASCANS* follows that in Ref. [19], though the order of the steps from Algorithm 1 has been modified for GPU efficiency reasons. We give a brief presentation of the algorithm implemented in *GASCANS* here, but refer the reader to Refs. [18] and [19] for derivations and detailed discussions. In the present version of *GASCANS* , the solid body is limited to being rigid, and the extension to flexible bodies is planned for the future.

Given the shape of an immersed body, *marker nodes* are laid out along the boundary. The node positions are independent of the fluid lattice, and instead conform to the geometry of the body. The spacing of the marker nodes is chosen to be roughly commensurate with the fluid lattice spacing.

In order to compute the IBM force on the fluid from the boundary of the body, the fluid density ρ and velocity $\mathbf{u}$ must be *interpolated* from the fluid lattice to the position of each marker node, and the computed force then *spread* back onto the fluid lattice as a force density.

The interpolation and spreading operators make use of the following discrete Dirac delta kernel[20]:

$$\tilde{\delta}(\mathbf{x}) = (\Delta x)^{-d} \prod_{i=1}^{d} \tilde{\delta}(x_i) \quad \text{where} \tag{14}$$

$$\tilde{\delta}(r) = \begin{cases} 1/3(1 + \sqrt{-3r^2 + 1} & |r| \leq 0.5 \\ 1/6(5 - 3|r| - \sqrt{-3(1 - |r|)^2 + 1} & 0.5 \leq |r| \leq 1.5 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

A field ϕ is interpolated from the fluid lattice with points $\mathbf{x}_j$ via

$$\mathcal{I}[\phi(\mathbf{x})]_s = \sum_{j \in D_s} \phi(\mathbf{x}_j) \tilde{\delta}(\mathbf{x}_j - \mathbf{x}_s)(\Delta x)^d \tag{16}$$

Conversely, a field Φ is spread from a marker node into the fluid lattice via

$$\mathcal{S}[\Phi(\mathbf{x})]_j = \sum_{s \in D_j} \Phi(\mathbf{x}_j) \tilde{\delta}(\mathbf{x}_s - \mathbf{x}_j) \epsilon_s \Delta s_1 \ldots \Delta s_d \tag{17}$$

where $\Delta s_1 \ldots \Delta s_d$ represents the volume associated with the point, and the computation is required only over the support of $\tilde{\delta}$. The factor ϵ_s ensures that $\mathcal{I}$ and $\mathcal{S}$ are reciprocal[21], and is computed by solving the linear system

$$A\epsilon = 1 \tag{18}$$

where the $N \times N$ matrix A is defined via

$$A_{nm} = \Delta s \sum_{\Omega} \tilde{\delta}(x - X_n) \tilde{\delta}(x - X_m) \tag{19}$$

and represents the degree of coupling between different marker node positions in the stencil of $\tilde{\delta}$, where N is the number of marker nodes.

We now describe how the LBM scheme described in Sec. 3.1 is modified to implement the IBM. The evaluation of the macroscopic velocity $\mathbf{u}$ in Eq. 11 (step 2 of the algorithm given at the end of Sec. 3.1) requires $\mathbf{F}(\mathbf{x}, t+\Delta t)$ to be computed. This proceeds as follows. The velocity is split into *predicted* and *force correction* terms,

$$\rho\mathbf{u} = \rho\mathbf{u}^* + \Delta t/2\mathbf{F} \tag{20}$$

with

$$\rho\mathbf{u}^* = \sum_i \mathbf{c_i} f_i^* \tag{21}$$

which can be immediately computed from the known f_i. This predicted velocity is the velocity that the fluid would have if the boundary force at time $t+\Delta t$ was zero.

ρ and $\mathbf{u}^*$ are then interpolated from the fluid grid to the marker node,

$$\rho_s = \mathcal{I}[\rho]_s \tag{22}$$

$$\mathbf{u}_s^* = \mathcal{I}[\mathbf{u}^*]_s \tag{23}$$

where the subscript s labels a marker node.

The essence of the IBM is to compute the boundary force $\mathbf{F}$ such that the fluid velocity $\mathbf{u}$ is equal to the boundary velocity $\mathbf{u}_\mathrm{B}$ (no-slip condition) at the positions of the marker nodes. Rearranging Eq. 20 and setting $\mathbf{u} = \mathbf{u}_\mathrm{B}$, we find

$$\mathbf{F}_s = 2/\Delta t \rho_s \left[\mathbf{u}_\mathrm{B} - \mathbf{u}_s^*\right] , \tag{24}$$

where again the subscript s indicates that the quantity is evaluated at the position of a marker node.

The force $\mathbf{F}_s$ on the marker node is then translated into a distributed force density field $\mathbf{F}$ via the spreading operator,

$$\mathbf{F} = \mathcal{S}[\mathbf{F}_S] , \tag{25}$$

where $\mathbf{F}$ will have support only in the neighbourhood of the boundary.

The velocity field $\mathbf{u}$ can now be computed as the sum of the predicted velocity and the contribution due to the calculated body force $\mathbf{F}$ using Eq. 20.

In the method described above, the interpolation, force-correction and spreading operations are associated with a specific marker node. Hence, it makes sense to parallelise the algorithm over marker nodes, with one GPU thread processing each node. This allows the computation to scale to larger numbers of GPU cores as the number of marker nodes increases, as it does with increased resolution, since we aim to have similar marker and lattice spacings.

We have developed an `IBM` CUDA module for *GASCANS* which interacts with the rest of *GASCANS* primarily via a single function call, `doIBM`. All IBM-related setup, including specification of the bodies, memory allocation, etc, is handled within this module. `doIBM` interpolates the fluid velocity and density arrays onto the marker nodes, solves for ϵ_s using the standard `CUSOLVER` library, computes the IBM force on each node, then spreads this force back into the fluid force density array. Hence, `IBM` is very loosely coupled with the rest of GASCANS, enhancing maintainability, and allowing for the possibility of reuse of this module.

The main LBM update step is implemented in GASCANS in a single kernel. This kernel is parallelised over the fluid cells and for each cell, performs the following operations in sequence:

1. Load required data from global memory into a local cache;
2. Stream populations f_i from neighbouring cells into the current cell;
3. Compute macroscopic density ρ and velocity $\mathbf{u}$ from the streamed f_i and any applied forces (e.g. gravity);
4. LBM collision to modify f_i according to the collision prescription, including any applied forces

The computation of the IBM force needs to be performed between steps 3 and 4. This introduces a complication, because the LBM algorithm is parallelised over fluid cells, whereas the IBM algorithm is parallelised over marker nodes. This required the introduction of a separate IBM kernel, and splitting the LBM kernel into two when run with IBM to allow the IBM algorithm to be run at the correct point. This impacts performance, as the cache needs to be reloaded during the second LBM kernel, hence the original unsplit kernel implementation is used when GASCANS is run without IBM.

In CUDA, it is efficient (see Ref. [22], *9.2.1 Coalesced Access to Global Memory*) for adjacent threads to access adjacent global memory locations simultaneously. This implies that a *structure of arrays* memory layout is beneficial, and this is what has been chosen for the IBM body data in *GASCANS* .

In this initial version, the IBM implementation is limited to running on a single GPU. Extension to multiple GPUs is planned for the future.

3.5. Turbulence Modelling

Turbulent flows act in a large range of time and space scales; to accurately model turbulence using the basic lattice Bolztmann equation (eq. 2) requires a domain as big as the bigger scales with cell size as small as the viscous dissipation scale. This makes modelling turbulent flow computationally expensive. Large eddy simulation (LES) turbulence models solve for the medium to large turbulent scales and model the small scales.

3.5.1. Large Eddy Simulation (LES) Smagorinsky

GASCANS implements the LES Smagorinsky turbulence model as described in [23]. Koda and Lien [23] model the effect of the turbulent scales smaller than the grid size by adding an extra term to the relaxation time τ_{tot}:

$$\tau_{tot} = \frac{1}{2\rho}\left(\sqrt{\rho^2\tau^2 + \sqrt{2}18\rho C_s \Delta^2 Q^{1/2}} - \tau\rho\right) + \tau \tag{26}$$

where τ is the original relaxation time in eq. 2, ρ is the macroscopic density, C_s is the Smagorinsky constant, usually set as $C_s = 0.01$, and $\Delta = 1$ is the cell size.

3.5.2. Synthetic Eddy Method (SEM)

Large eddy simulations requires time dependent instantaneous values of velocity at the inlet in order to work properly. However, it is common to only have mean flow data. The Synthetic eddy method generates instantaneous velocity values from the following mean flow data: mean velocity U_i, Reynolds stresses $u_i u_j$ and turbulence dissipation rate ϵ.

GASCANS implements the SEM model developed by Skillen et al. [24], which is executed by the CPU part of *GASCANS* .

The main novelty added in *GASCANS* is that the model is optimised to the constant grid size used in lattice Boltzmann simulations and also able to run in parallel using OpenMP. Moreover, the user can set a SEM Courant number, which will determine how often the instantaneous velocities are updated. The constant grid size optimisation together with the OpenMP parallelisation offer around 40 times speed up compared to implementing the base SEM model directly in *GASCANS* . The SEM Courant number feature can be activated if the computational time needs to be further reduced.

3.6. Time Averaging

The large range of scales of motion overlapped in turbulent flow make it difficult to analyse the flow using only the instantaneous velocity values u_i. One of the most common analysis is to represent the instantaneous velocity u_i as the sum of a mean component $U_i = \overline{u_i}$ and a fluctuating component $u'_i = u_i - \overline{u_i}$ with $\overline{u'_i} = 0$. Some characteristics of turbulent flows can be described using the time averaged velocity U_i and the time averaged Reynolds stresses $\overline{u'_i u'_j}$.

GASCANS implements time averaging for each component of the velocity U_i and the time averaging of the products of the components $\overline{u_i u_j}$

$$U_i = \sum_t u_i(t) \tag{27}$$

$$\overline{u_i u_j} = \sum_t u_i(t) u_j(t) \qquad i, j = 0:2 \tag{28}$$

The time averaged Reynolds stresses can be calculated as:

$$\overline{u'_i u'_j} = \overline{u_i u_j} - U_i U_j \tag{29}$$

3.7. Sponge Layer

The majority of lattice Boltzmann boundary conditions reflect pressure waves due to the weakly compressible nature of the lattice Boltzmann method. These pressure waves can destabilize the simulation, especially in the vicinity of outlet boundaries (or Neuman boundaries) in turbulent flows. Sponge layers are a common tool to stabilize the simulation near outlets.

GASCANS implements a viscosity sponge layer for the yz wall at $x = N$, where N is the number of cells in the x direction. Within the sponge layer the viscosity of the fluid is increased as it approaches the outlet via:

$$\nu_{total} = \nu \left[1 + m \left(\frac{i - i_0}{N - i_0} \right)^a \right] \tag{30}$$

where ν_{total} is the viscosity applied within the sponge layer, ν is the viscosity of the fluid in lattice Boltzmann units, m is a multiplier, i is the current cell x index, i_0 is the x cell index where the sponge layer starts and a determines the order of the polynomial. For example $m = 1$ and $a = 3$ double the viscosity at the outlet going from $\nu_{total} = \nu$ at the beginning of the sponge layer to $\nu_{total} = 2\nu$ at the outlet boundary following a cubic function.

4. Validation

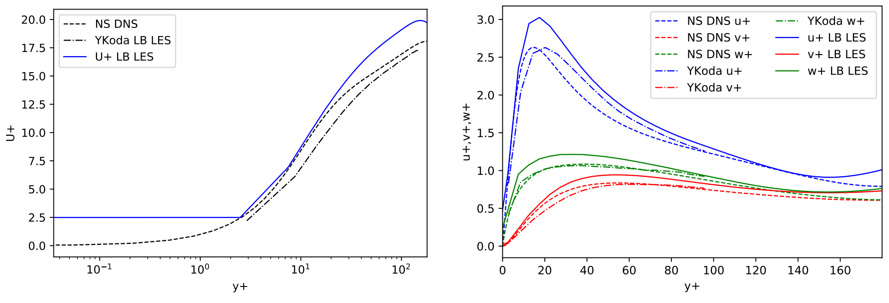
4.1. Turbulent Channel Flow

This test case validates the implementation of the LES Smagorinsky model in *GASCANS* . The case is based in the numerical case by [23] and models a periodic turbulent channel flow with $Re_\tau = 180$ and dimensions $12H \times 2H \times 4H$ in x, y and z respectively, meshed using cubic cells of size $\delta x = 1/31$ and time step $\delta t = 0.001$. H is half the height of the channel. The top and bottom walls are set to no slip using bounce back boundary conditions and the boundaries in the streamwise and spanwise directions are periodic. This case uses the Smagorinsky LES turbulence model (section 3.5.1) with a Smagorinsky constant $C_{sa} = 0.01$ and using the solved density ρ to calculate the total relaxation time (section 3.5.1).

Fig. 4 shows the *GASCANS* results compared to the Navier-Stokes DNS results by Kim et al. [25] and the lattice Boltzmann LES results by [23]. *GASCANS* results show good agreement with the results in the literature for the streamwise mean velocity in wall units $U^+ = U/U_\tau$ and the fluctuating velocities in wall units u^+, v^+ and w^+. The peak location of u^+ is displaced for both Y.Koda [23] and *GASCANS* ; *GASCANS* presents an overprediction of u^+, v^+ and w^+ which is consistent with the overprediction of the mean velocity U^+. The bulk velocity Reynolds Re_b number for the *GASCANS* channel flow is 5% below the Re_b for a $Re_\tau = 180$ channel flow, which can explain the overprediction of the mean and fluctuating velocities. Fig. 4 also shows the maximum U and minimum u^+, v^+, w^+ values displaced from the centre of the channel, which is also an indication of the mismatch in the Re numbers.

4.2. Turbulent channel flow with synthetic eddy method inlet

This test case is a 3D turbulent channel flow with $Re_\tau = 395$ and dimensions $10H \times 2H \times 3.16H$ in x (streamwise), y (vertical) and z (spanwise) directions



(a) (b)

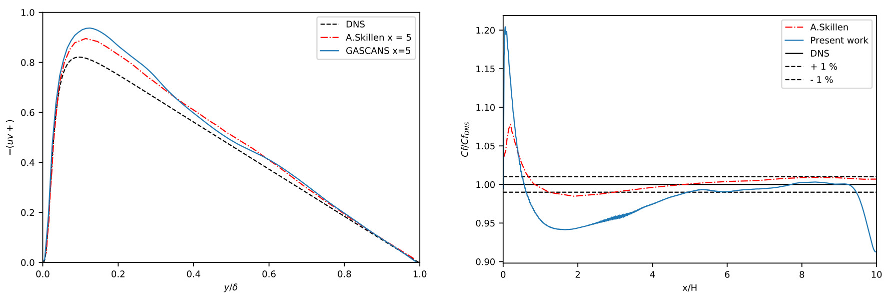
Figure 4: Mean streamwise velocity in wall units (left) and fluctuating velocity components (right): *GASCANS* results (continuous), NS DNS (cite paper), (dashed) and Y.Koda LB LES [23] (dash dot). Mean streamwise velocity U^+ (black), fluctuating velocity u^+ (blue), v^+ (red), w^+ (green).

respectively, meshed using cubic cells of size $\delta x = 0.01$ and time step $\delta t = 0.0001$. H is half the channel height. The top and bottom walls are set to no slip using bounce back boundary conditions and the boundaries in the z direction are periodic. The outlet is set as an extrapolated boundary and a viscosity sponge layer is added from $x = 9$ onwards to stabilize the result at the outlet. The flow enters the domain through the inlet, which sets a regularised velocity boundary condition. The inlet velocity is generated using the synthetic eddy method (see section 3.5 and [24]) with the mean flow data from the Kozuka et al. [26] DNS simulation. The SEM is configured with a maximum eddy size $\sigma_{max} = 0.5$ and a minimum eddy size $\sigma_{min} = 4\delta x$. No turbulence model is used for this test case.

Fig. 5 shows the vertical profile of the shear Reynolds stress at $x = 5$ and the wall friction coefficient along the streamwise direction obtained by *GASCANS* with no turbulence modelling compared with the ones obtained by Skillen et al. [24]; which models a channel flow of similar characteristics but slightly higher resolution using a LES turbulence model. Both the shear Reynolds stress (Fig. 5 (a)) and the wall friction coefficient (Fig. 5 (b)) agree with the results from Skillen et al. [24] and with the DNS data from Kozuka et al. [26]. The main differences are in the friction coefficient (Fig. 5 (b)). *GASCANS* presents a more extreme minimum and maximum values but both results are within 1% of the DNS data from $x = 5$ onwards. The decrease in the friction coefficient near *GASCANS* outlet is due to the effect of the sponge layer and does not affect the accuracy of the SEM.

4.3. Ahmed Body

This section validates the ability of *GASCANS* to use geometries imported from point cloud files. The chosen test case is 3D laminar flow around the Ahmed body; its position in the simulation domain and shape are shown in Fig. 6.



(a) (b)

Figure 5: Reynolds shear stress in wall units at $x = 5$ (left), where δ is half the channel height; and wall friction coefficient non-dimensionalised with the DNS wall friction coefficient from [26] (right). *GASCANS* results (continuous blue), Navier-Stokes periodic DNS by Kozuka et al. [26] (dashed black) and Skillen et al. [24] SEM Navier-Stokes LES (dot dashed red).

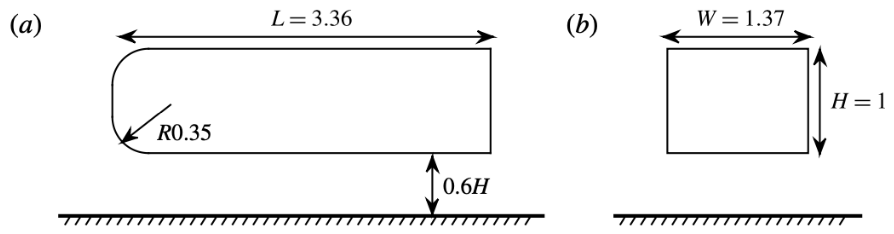


Figure 6: Configuration of the Ahmed body: (a)side view, (b)rear view.[27]

This case models a laminar flow with $Re_H = 310$, where H is the height of the Ahmed body. The domain size is $26.88H \times 6.72H \times 6.72H$ and it is meshed with a constant cell size $\delta x = H/27$ and time step $\delta t = 0.0002$.

As shown in Figure 7, despite of side view or top view, the GASCANS results are quite close to the results on Evstafyeva et al. [27]. The flow after Ahmed body consists of two recirculating regions which lies horizontally on the top. And we can see that the flow is steady and symmetric.

4.4. Flow around a rigid cylinder and filament

We have used our implementation of the immersed boundary method to simulate the 2D flow of a fluid around a rigid cylinder and filament, corresponding to case CFD1 in [28]. The steady-state drag and lift on the bodies are plotted in Fig. 8 as a function of lattice cell spacing, and compared with the reference result from [28] plotted as a dashed line. We see that the *GASCANS* results converge to a value similar to the published reference results.

4.5. 2D moving plate

We have simulated the flow of a fluid around an impulsively started 2-dimensional infinitesimally thin finite flat plate. At time $t = 0$, the fluid and

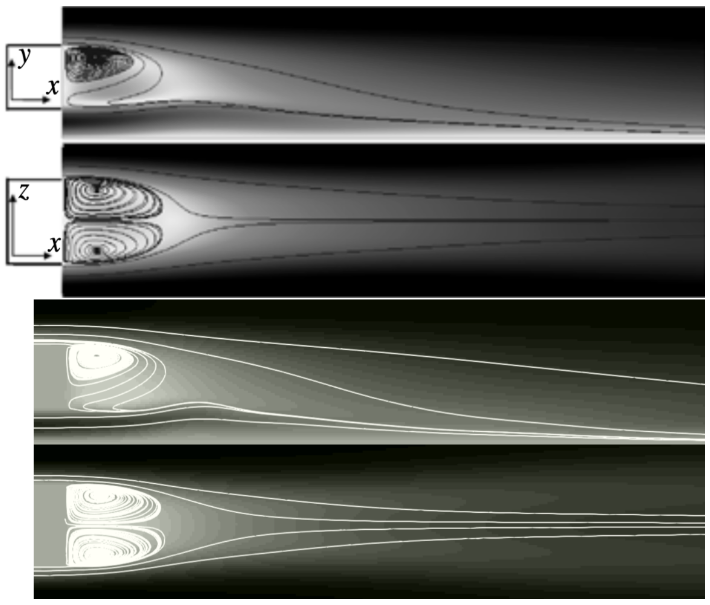


Figure 7: The side view and top view of the flow after Ahmed body via 2D slice with streamlines. (The top two are from Evstafyeva et al. [27], while bottom two results are simulated by *GASCANS*)

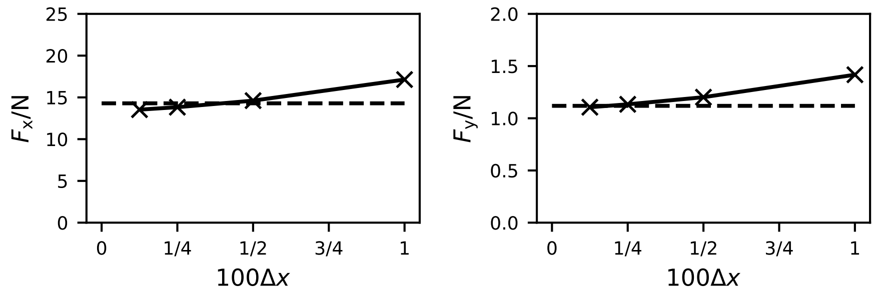
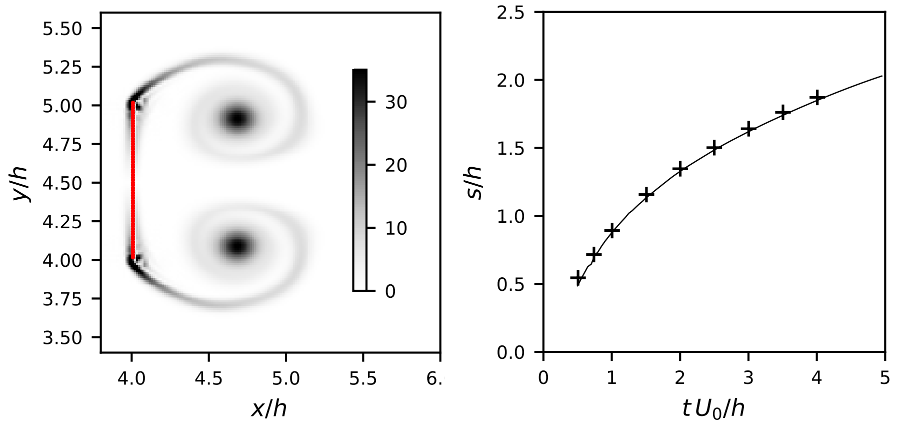


Figure 8: Drag (F_x) and lift (F_y) forces on a rigid cylinder and filament (CFD1 in [28]) computed by *GASCANS* for different lattice spacings Δx. The reference result from [28] is shown as a dashed line.

plate are at rest, and the plate of height h is instantaneously accelerated to a velocity U_0. The velocity is prescribed, rather than being determined by the force exerted by the fluid on the plate. The Reynolds number $\mathrm{Re} = U_0 h/\nu$ is chosen as 1000, where ν is the kinematic viscosity of the fluid. The fluid domain dimensions are $12h \times 9h$, the lattice spacing is $\Delta x = 12h/720$, and the time step is $\Delta t = 5 \times 10^{-4} h/U_0$. This case has been studied in Ref. [29]. In our treatment, extrapolation boundary conditions are applied at all boundaries, and there are 60 IBM markers across the plate.



(a) Vorticity at $tU_0/h = 2$. The plate, represented by the IBM markers at $x = 4h$, is moving to the left;

(b) Non-dimensional separation bubble length s/h behind the plate as a function of non-dimensional time tU_0/h from *GASCANS* (solid line) compared with results from the literature (pluses)

Figure 9: Simulation of an impulsively started moving plate with *GASCANS*

Fig. 9a shows the absolute value of the vorticity around the moving plate, and the two vortices formed behind the plate. The result agrees visually with that reported in Fig. 2.(a) of Ref. [30], in which the same configuration was studied.

Fig. 9b shows the non-dimensional length s/h of the reversed-flow region behind the plate as a function of the non-dimensional time tU_0/h. The solid line corresponds to the present results, and the pluses are the values obtained in Ref. [29]; we see good agreement.

4.6. Coupling with OpenFOAM

This test is based on the numerical study by Hwang and Yang [31] and used to validate the two-way coupling facilities implemented in *GASCANS* . The case models laminar flow ($Re = 150$) around a wall mounted cube in a channel. The cube has size 1 and is placed at 3 units from the inlet in the streamwise direction and centred in the spanwise direction. The original domain is split in two sub-domains (see Fig. 10), one solved using *GASCANS* and one solved using the pisoFoam solver of the Navier-Stokes CFD OpenFoam [32]. The *GASCANS* sub-domain models the flow from the inlet until 7 in the streamwise direction x; the OpenFOAM sub-domain models the flow from $x = 5$ to $x = 10$.

The *GASCANS* sub-domain measures $7\times2\times8$ units in x, y and z respectively and is meshed using cubic cells of size $\delta x_{LB} = 0.025$ and a time step $\delta t_{LB} = 0.0003125$. The top and bottom boundaries are set to no-slip using bounce-back

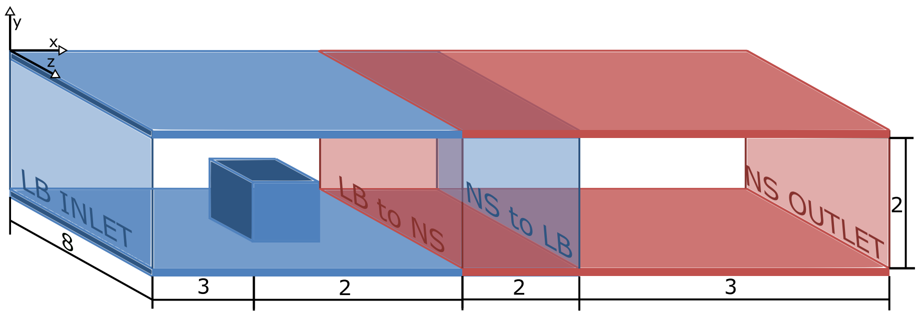


Figure 10: Sketch of the coupled case domain. LB denotes the *GASCANS* sub-domain, NS the OpenFoam sub-domain, LB to NS is the boundary where the data is interpolated from *GASCANS* to OF and the OF left boundary, NS to LB is the boundary where the data is interpolated from OF to *GASCANS* and the *GASCANS* right boundary.

boundary conditions, the inlet (LB inlet in Fig. 10) implements a regularised boundary condition (cite regularised paper) that sets the inlet velocity to a laminar parabolic profile, the sides in the z direction are set to periodic and the right boundary (NS to LB in Fig. 10) implements a forced equilibrium boundary condition that sets the velocity to the values interpolated from the OpenFOAM sub-domain and the the density to 1.

The OpenFOAM sub-domain starts at $x = 5$ and measures $5 \times 2 \times 8$ units in x, y and z respectively. It is meshed using NS a structured mesh refined towards the bottom and the cube walls with a minimum cell height of $\delta x_{NS} = 0.006$ and a time step $\delta t_{NS} = 0.005$. The top and bottom boundaries are set to no-slip, the inlet (LB to NS in Fig. 10) is set to the velocity interpolated from the *GASCANS* subdomain and the pressure gradient is set to 0, the sides in the z direction are set to periodic, the outlet (NS outlet in Fig. 10) velocity gradient is set to zero and the outlet pressure is set to 0. The OpenFOAM inlet requires the volumetric flow rate at the inlet to be corrected to $\dot{m} = A * U_b$, where A is the area of the inlet and U_b is the bulk velocity of the parabolic velocity profile. If the flow is not corrected, OpenFOAM tries to adjust it by modifying the pressure at the outlet, which generates instabilities at the coupling and invalidates the results.

The two sub-domains are coupled using a two-way explicit algorithm in which both sub-domains are run simultaneously. The results of the Navier-Stokes sub-domain are interpolated to the NS to LB boundary in the LB sub-domain; the results of the lattice Boltzmann subdomain are interpolated to the LB to NS boundary in the Navier-Stokes sub-domain (see Fig. 10). The information is exchanged when both solvers reach the coupling time window.

This section only presents the main results of this test case. A more in depth study of the accuracy, performance and effect of the overlap region can be found in [33].

Fig. 11 shows the results of running the test case until a steady state is reached. The streamlines are continuous in both the OpenFoam and *GASCANS* sub-domains, except for a small discrepancy at the *GASCANS* right boundary that can also be observed in the pressure values. This discrepancy is mainly due to the forced equilibrium boundary condition applied to *GASCANS* ' right boundary since it sets the *GASCANS* density to a constant value and it disregards the off-equilibrium information in the OpenFOAM velocity.

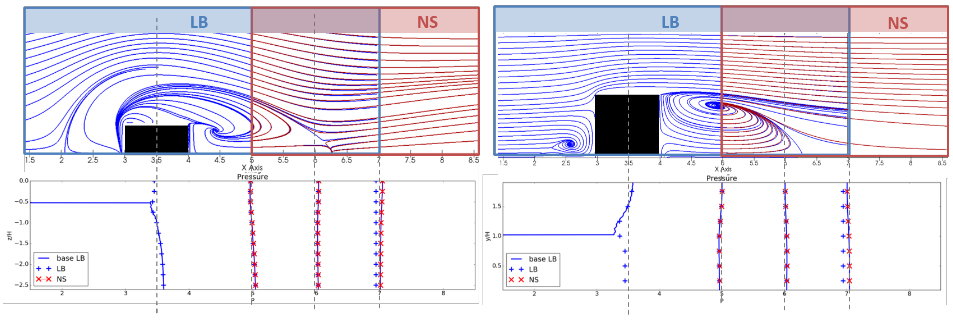


Figure 11: Streamlines on the horizontal plane y=0.04 (top-left) and the central vertical plane z = 0 (top-right) for the coupled *GASCANS* to OpenFOAM simulation, LB streamlines (blue), NS streamlines (red). Bottom plots: pressure at 4 lines in each plane compared with the results of a single LB model.

5. Performance

The Lattice Boltzmann Method is very well matched to the performance characteristics of GPUs, due to the high degree of local computation in a single cell relative to data transfers from neighbouring cells. In this section, we briefly report the results of performance benchmarks we have run on *GASCANS* . We run the benchmarks on a single node of the GPU cluster Bede [34]. The node has 4 Nvidia Tesla V100 GPUs with 32 GB of GPU memory and NVLink 2.0, as well as 2x POWER9 CPUs running at 2.7GHz (32 cores total and 4 hardware threads per core).

The benchmark setup consists of a 3D cubical grid consisting of $384 \times 384 \times 384 N_{\mathrm{GPUs}}$ cells running the core LBM solver in single precision for 1000 timesteps, where N_{GPUs} is the number of GPUs over which the problem is run. This constitutes a weak scaling test, as the problem size per GPU is independent of the number of GPUs. The algorithm is not data-dependent, hence a zero-velocity flow is sufficient to measure performance. We use periodic boundary conditions on all faces of the grid.

The benchmark results are presented in Table 1 and Fig. 12 in terms of the speed of the code in million lattice updates per second (MLUPS).

We see that the single-GPU performance is 2030 MLUPS, and for larger problems that require more GPU memory, the code can be scaled to 4 GPUs with 87% of ideal performance.

n_{GPUs}	Speed [MLUPS]	Scaling efficiency
1	2030	100
2	3746	92
3	5310	87
4	7048	87

Table 1: Weak scaling performance of *GASCANS* . The evolution speed in million lattice updates per second (MLUPS) is given for a fixed problem size per GPU against the number of GPUs, as well as the resulting fraction of ideal scaling performance obtained.

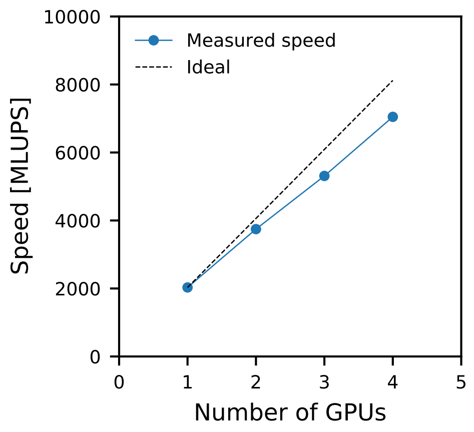


Figure 12: Weak scaling performance of *GASCANS* . The evolution speed in million lattice updates per second (MLUPS) is plotted for a fixed problem size per GPU against the number of GPUs, as well as the line representing ideal weak scaling.

Table 2 shows *GASCANS* performance compared with other GPU LB solvers with LB schemes similar to *GASCANS* ; executed using one Nvidia V100 GPU. To the extend of our knowledge and as shown in Table 2, there is no agreed upon test case to evaluate the performance of a GPU LB code, which could explain in part the performance differences between the codes. GPU perfomance is very sensitive to inhomogeneities in the treatment of each cell (i e. branch divergence) [35], the floating point precision [36], the LB scheme, the data structure and the storage and treatment of the particle distribution functions during streaming [9].

Oliveira et al. [36] present the test case and LBM scheme closest to *GASCANS* but performs at 1.95 times *GASCANS* speed. This speed difference is probably due to the treatment of the boundary conditions. Oliveira et al. [36] uses a 32 bits bitmap to store: type of cell, the normal direction of the boundary cell, the boundary condition to apply and the index of an array with its macroscopic values. *GASCANS* needs to determine if a cell is next to the boundary and which is its normal at run time, thus increasing the computing time.

Reference	Speed [MLUPS]	Test case	LBM scheme
Oliveira et al. [36]	3959.8 (s)	Periodic domain	D3Q19 BGK
STLBM [9]	1422.6 (d)	3D lid driven cavity	D3Q19 BGK
UltrafluidX [5]	800 (s)	Empty wind tunnel	D3Q27 cumulant
GASCANS	2030 (s)	Periodic domain	D3Q19 BGK

Table 2: Comparison of *GASCANS* performance with recent works using Nvidia V100 GPU(s). This table includes the MLUPS in (s) single precision or (d) double precision for each code , the test case used to evaluate the performance and the LB scheme and collision model.

6. Conclusions

This paper presented the multi-GPU lattice Boltzmann code *GASCANS* . *GASCANS* offers fast and easy to configure simulations for turbulent flow and moving objects. Moreover, *GASCANS* is able to assimilate velocity data from external sources at run time and implements an interface with the coupling library preCICE [13]. *GASCANS* combined with preCICE can run coupled simulations with other solvers, in which each of the participating solvers exchange their boundary data during run-time.

GASCANS fills an niche in the market of open-source GPU accelerated lattice Boltzmann solvers, providing ready to use and adaptable turbulent flow and moving objects modelling aimed at engineers with limited or no access to large computer systems. *GASCANS* performance is comparable with other LB GPU codes with similar LBM schemes, but has room for improvement. One suggestion to improve *GASCANS* performance would be to implement a more efficient treatment of the boundary cells.

Regarding, turbulent flows, the synthetic eddy method inlet together with the *GASCANS* core BGK lattice Boltzmann and its Smagorinsky LES turbulence model yield results of an accuracy comparable to the ones obtained using classic Navier-Stokes methods. Moreover, the point cloud reader allows to easily introduce any geometry into the domain and obtain accurate results, as demonstrated with the Ahmed body test case (section 4.3).

We validated the inmmersed boundary method (IBM) boundary condition with flow around a rigid cylinder and filament from [28] and the 2D moving plate from [29]. In both cases *GASCANS* shows good agreement with the base results. The implementation of IBM allows *GASCANS* to model curved surfaces and moving object and it is an important step towards modelling fluid structure interaction.

Finally, future developments projected for *GASCANS* include testing IBM for 3 dimensional flows and implementing coupling with a solid solve for FSI simulations; also introducing momentum exchange to obtain forces on point cloud objects. Finally, further optimisation and integration of the code features including implementing IBM for multi-GPU.

References

[1] Y. Feng, P. Boivin, J. Jacob, P. Sagaut, Hybrid recursive regularized lattice Boltzmann simulation of humid air with application to meteorological flows, Physical Review E 100 (2) (2019) 023304. doi:10.1103/PhysRevE.100.023304.
URL `https://link.aps.org/doi/10.1103/PhysRevE.100.023304`

[2] S. Lenz, M. Schönherr, M. Geier, M. Krafczyk, A. Pasquali, A. Christen, M. Giometto, Towards real-time simulation of turbulent air flow over a resolved urban canopy using the cumulant lattice Boltzmann method on a GPGPU, Journal of Wind Engineering and Industrial Aerodynamics 189 (October 2018) (2019) 151–162.

[3] M. J. Mawson, A. J. Revell, Memory transfer optimization for a lattice Boltzmann solver on Kepler architecture nVidia GPUs, Computer Physics Communications 185 (10) (2014) 2566–2574.
URL `http://dx.doi.org/10.1016/j.cpc.2014.06.003`

[4] Dassault Systemes, Xflow.
URL `https://www.3ds.com/products-services/simulia/products/xflow/`

[5] C. A. Niedermeier, C. F. Janssen, T. Indinger, Massively-parallel multi-GPU simulations for fast and accurate automotive aerodynamics, Proceedings of the 6th European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th European Conference on Computational Fluid Dynamics, ECFD 2018 (June) (2020) 2005–2012.

[6] Łaniewski-Wołłk, J. Rokicki, Adjoint Lattice Boltzmann for topology optimization on multi-GPU architecture, Computers and Mathematics with Applications 71 (3) (2016) 833–848.

[7] M. Bauer, S. Eibl, C. Godenschwager, N. Kohl, M. Kuron, C. Rettinger, F. Schornbaum, C. Schwarzmeier, D. Thönnes, H. Köstler, U. Rüde, WALBERLA: A block-structured high-performance framework for multiphysics simulations, Computers and Mathematics with Applications 81 (2021) 478–501.

[8] M. Januszewski, M. Kostur, Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method, Computer Physics Communications 185 (9) (2014) 2350–2368.
URL `http://dx.doi.org/10.1016/j.cpc.2014.04.018`

[9] J. Latt, C. Coreixas, J. Beny, Cross-platform programming model for many-core lattice Boltzmann simulations (2020).
URL `http://arxiv.org/abs/2010.11751`

[10] P. Bhatnagar; E. Gross; M.Krook;, A model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems, Physical Review 94 (1) (1954).

[11] M. Geier, M. Schönherr, A. Pasquali, M. Krafczyk, The cumulant lattice Boltzmann equation in three dimensions: Theory and validation, Computers and Mathematics with Applications 70 (4) (2015) 507–547.
URL `http://dx.doi.org/10.1016/j.camwa.2015.05.001`

[12] A. R. G. Harwood, A. J. Revell, Interactive flow simulation using Tegra-powered mobile devices, Advances in Engineering Software 115 (Supplement C) (2018) 363 – 373.

[13] H. J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann, preCICE – A fully parallel library for multi-physics surface coupling, Computers and Fluids 141 (2016) 250–258.
URL `http://dx.doi.org/10.1016/j.compfluid.2016.04.003`

[14] A. R. G. Harwood, P. Wenisch, A. J. Revell, A Real-Time Modelling and Simulation Platform for Virtual Engineering Design and Analysis, in: Proceedings of 6th European Conference on Computational Mechanics (ECCM 6) and 7th European Conference on Computational Fluid Dynamics (ECFD 7), 11-15 June 2018, Glasgow, UK, ECCOMAS, 2018.

[15] T. Krüger, The Lattice Boltzmann Method Principles and Practice, no. March 2015, 2017.

[16] J. Latt, B. Chopard, O. Malaspinas, M. Deville, A. Michler, Straight velocity boundaries in the lattice Boltzmann method, Physical Review E - Statistical, Nonlinear, and Soft Matter Physics 77 (5) (2008) 1–16.

[17] Z. Guo, C. Zheng, B. Shi, Discrete lattice effects on the forcing term in the lattice Boltzmann method, Physical Review E 65 (2002) 046308.

[18] C. S. Peskin, Flow patterns around heart valves: A numerical method, Journal of Computational Physics 10 (2) (1972) 252 – 271. doi:https://doi.org/10.1016/0021-9991(72)90065-4.
URL `http://www.sciencedirect.com/science/article/pii/0021999172900654`

[19] Z. Li, J. Favier, U. D'Ortona, S. Poncet, An immersed boundary-lattice boltzmann method for single- and multi-component fluid flows, Journal of Computational Physics 304 (2016) 424 – 440. doi:https://doi.org/10.1016/j.jcp.2015.10.026.
URL `http://www.sciencedirect.com/science/article/pii/S0021999115006907`

[20] A. M. Roma, C. S. Peskin, M. J. Berger, An adaptive version of the immersed boundary method, Journal of Computational Physics 153 (2)

(1999) 509 – 534. doi:https://doi.org/10.1006/jcph.1999.6293.
URL http://www.sciencedirect.com/science/article/pii/S0021999199962939

[21] A. Pinelli, I. Naqavi, U. Piomelli, J. Favier, Immersed-boundary methods for general finite-difference and finite-volume navier–stokes solvers, Journal of Computational Physics 229 (24) (2010) 9073 – 9091. doi:https://doi.org/10.1016/j.jcp.2010.08.021.
URL http://www.sciencedirect.com/science/article/pii/S0021999110004687

[22] NVIDIA, CUDA C++ Best Practices Guide.
URL https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/

[23] Y. Koda, F. S. Lien, The lattice Boltzmann method implemented on the GPU to simulate the turbulent flow over a square cylinder confined in a channel, Flow, Turbulence and Combustion 94 (3) (2015) 495–512.

[24] A. Skillen, A. Revell, T. Craft, Accuracy and efficiency improvements in synthetic eddy methods., International Journal of Heat and Fluid Flow 62 (2016) 386–394.

[25] J. Kim, P. Moin, R. Moser, Turbulence statistics in fully developed channel flow at low reynolds number, Journal of Fluid Mechanics 177 (1987) 133–166.

[26] K. Kozuka, Y. Seki, H. Kawamura, Direct numerical simulation of turbulent heat transfer with a high spatial resolution, Proc. of the 7rd International Symposium on Engineering Turbulence Modelling and Mesurements -ETMM7 1 (September 2016) (2008) 163–168.

[27] O. Evstafyeva, A. S. Morgans, L. Dalla Longa, Simulation and feedback control of the ahmed body flow exhibiting symmetry breaking behaviour, Journal of Fluid Mechanics 817 (2017) –.

[28] S. Turek, J. Hron, Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow, in: H.-J. Bungartz, M. Schäfer (Eds.), Fluid-Structure Interaction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 371–385.

[29] P. Koumoutsakos, D. Shiels, Simulations of the viscous flow normal to an impulsively started and uniformly accelerated flat plate, Journal of Fluid Mechanics 328 (1996) 177–227. doi:10.1017/S0022112096008695.

[30] J. Favier, A. Revell, A. Pinelli, A lattice boltzmann–immersed boundary method to simulate the fluid interaction with moving and slender flexible objects, Journal of Computational Physics 261 (2014) 145 – 161. doi:https://doi.org/10.1016/j.jcp.2013.12.052.
URL http://www.sciencedirect.com/science/article/pii/S0021999113008607

[31] J. Y. Hwang, K. S. Yang, Numerical study of vortical structures around a wall-mounted cubic obstacle in channel flow, Physics of Fluids 16 (7) (2004) 2382–2394.

[32] The OpenFOAM Foundation, Openfoam.
URL `https://openfoam.org/version/4-0/`

[33] M. Camps Santasmasas, A. Revell, B. Parslew, Two-way coupled Navier-Stokes / lattice Boltzmann solver to reduce the resources used by CFD simulations of flow around bluff objects., In preparation.

[34] The Bede supercomputer, Durham University, operated by N8 CIR, `https://n8cir.org.uk/supporting-research/facilities/bede/`, accessed: 11-03-2021.

[35] N. DelBosc, Real-time simulation of indoor air flow using the lattice boltzman method on graphics processing unit, Ph.D. thesis (2015).

[36] W. B. J. de Oliveira, A. Lugarini, A. Franco, Performance Analysis of the Lattice Boltzmann Method, XL CILAMCE iBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING (November) (2019).