High Performance Computing with Python

Final Report

NAME

matricular number
mail

July 3, 2020

Contents

Abbreviations

BTE	Boltzman Transport Equation
CI	Continuous Integration
LBM	Lattice Boltzman Method

1

Introduction

[1]

The Lattice Boltzman Method (LBM) is a numerical parallelizable and efficient scheme for simulating fluid flows. In addition, the LBM can be extended with boundary conditions. The key property of the LBM is that it is a discrete kinetic theory approach featuring a mescoscale description of the microstructure of the fluid instead of discretising macroscopic continuum equations. Other key advantages of the LBM include: efficient implementation by parallelization.

All code is available at `https://github.com/infomon/lattice_boltzman_parallel_solver` under BSD license. We give the instructions how to reproduce the results of the experiments conducted in this report in the *README*.

Structure of report

The remainder of the report is organized as follows:

- **Chapter 2** describes the LBM. More specifically, we describe how we discretize the *Boltzman transport equation* resulting in the LBM. We also show how macroscopic quantities, e.g. density and velocity, can be calculated from the microscopic simulation. In addition, we describe several boundary conditions that can be applied in the LBM.
- **Chapter 3** describes how the LBM is implemented using *Python* as programming language. We also show how we parallelized the implementation and how we ensured software quality by unit testing.
- **Chapter 4** conducts extensive experiments showing the applicability and correctness of the implementation of the solver for the LBM.
- **Chapter 5** concludes this report.

2

Lattice boltzman method

2.1 Overview

In this chapter we describe the Lattice Boltzman Method (LBM). The basic idea of the LBM is to statistically consider particles and consider them as a whole unit.

2.2 Boltzman Transport Equation (BTE)

The boltzman transport equation defines the fundamental differential equation of kinematic gas theory.

Streaming

Collisions

Discretization of the BTE

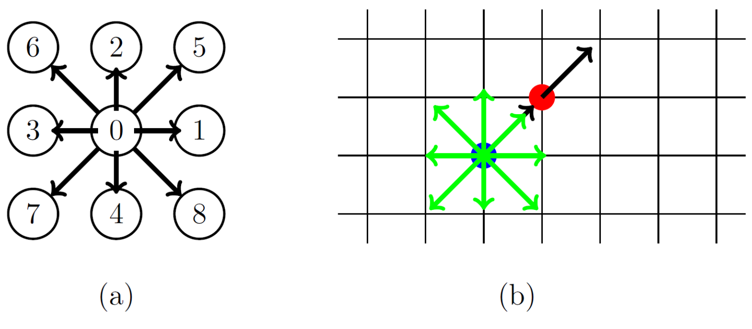

Figure 2.1: example figure

2.3 Lattice

2.4 Moment update

In the LBM, the density ρ and velocity $\mathbf{u}$ are defined by the zeroth and first moments of the probability distribution function f_i, respectively:

$$\begin{aligned}\rho(\mathbf{x},t) &= \int f(\mathbf{x},\mathbf{u},t)\ d^3\mathbf{u},\\ \mathbf{u}(\mathbf{x},t) &= \frac{1}{\rho(\mathbf{x})}\int f(\mathbf{x},\mathbf{u},t)\cdot \mathbf{c}(\mathbf{u})\ d^3\mathbf{u}.\end{aligned}$$

Discretizing those equation we get

$$\begin{aligned}\rho(\mathbf{x}) &= \sum_i f_i,\\ \mathbf{u}(\mathbf{x}) &= \sum_i f_i\mathbf{c}_i.\end{aligned}$$

2.5 Boundary conditions

One key advantage of the LBM is its easy implementation of boundary conditions. To apply boundary conditions the distributions function f_i is modified at each boundary lattice point $\mathbf{x}_b$ in each time step. That is $f_i(\mathbf{x}_b + c_i\Delta t, t+\Delta t) = f_i^*(\mathbf{x}_b, t)$, where $\mathbf{x}_b$ denotes the boundary nodes, f_i^* the distribution function before the streaming and f_i the distribution function after the streaming.

One question that arises is where the boundary nodes $\mathbf{x}_b$ are defined. We distinguish between so called *wet nodes* and *dry nodes* due to different domains, i.e. computational and physical domain. In the former, the computation and physical domain is the same (i.e. the boundaries are placed on the lattice nodes) but this comes with a increased difficulty for the implementation. In the latter the physical domain is half a cell away from the computational domain (i.e. the boundaries are located between the lattice nodes) retaining second order accuracy as long as the boundary is placed exactly in the middle of the lattice nodes.

Below we describe several boundary conditions. Note, that they can be arbitrarily combined as long as they do not contradict themselves.

Periodic

For a periodic boundary condition the flow leaving a boundary re-enters the domain on the opposite side of the domain. Note, that this boundary condition does not simulate reality. This is that $f_i(\mathbf{x}_1, t) = f_i(\mathbf{x}_N, t)$, where $\mathbf{x}_1$ and $\mathbf{x}_N$ are the first and last node in the physical domain, respectively. Visually, we can imagine that we have a cylindrical shape. The periodic boundary condition is implicitly implemented by the streaming function.

Bounce-back

The bounce-back boundary condition applies a no-slip condition at the boundary. It simulates the interaction between the fluid with a non-moving wall without slip. It can also be applied to a stationary obstacle such as a plate.

Moving wall

Inlet

Outlet

Periodic with pressure variation

3

Implementation

In this chapter we will describe how we implement the algorithm using *Python* as programming language.

3.1 Overview

Algo. 1 shows the pseudocode of the iteration loop of the LBM. As input we can specify the geometry of the physical domain, the boundary conditions (see section 2.5 for more details) as well as the initial conditions.

First we initialize the density ρ and velocity $\mathbf{u}$ and compute the initial value of the probability density function $f_i^{eq} = f_i$.

Then we iterate in a loop over several steps as long as the stopping criterion (e.g. maximum time steps) is not satisfied. Note, that there is some flexibility when to apply which step. The following order of steps corresponds to the order in the implementation of the LBM. We first compute the equilibrium function f_i^{eq} given the current density ρ and velocity $\mathbf{u}$. In the collision step we simulate the effects of collisions between particles (see section 2.2 for more details). After that, we simulate the streaming of f_i, i.e. we simulate the movement of particles to the nearest neighbour lattice nodes using the D2Q9 discretization. Then we apply potential boundary conditions on the probability density function f_i. Lastly, we compute the density ρ and velocity $\mathbf{u}$ (see section 2.4 for details on the formulas on how to compute the macroscopic quantities).

After running the LBM we can obtain the density ρ and velocity $\mathbf{u}$ as macroscopic quantities.

```
  Input: Geometry and parameters $l$, $h$, $U$, $\nu$,...; boundary
         conditions; initial conditions
  Output: Final density $\rho$ and velocity $\mathbf{u}$
1 initialize $\rho$ and $\mathbf{u}$
2 compute $f_i$ and $f_i^{eq}$
3 while stopping criterion is not satisfied do
4     compute equilibrium function $\rho,\ \mathbf{u} \rightarrow f_i^{eq}$
5     collision step $f_i^* = f_i(\mathbf{x}, t) - \frac{\Delta t}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t))$ (see section
       2.2)
6     streaming $f_i(\mathbf{x} + c_i\Delta t, t + \Delta t) = f_i^*(\mathbf{x}, t)$ (see section 2.2)
7     apply boundary conditions $f_i(\mathbf{x}_b + c_i\Delta t, t + \Delta t) = f_i^*(\mathbf{x}_b, t)$ (see
       section 2.5)
8     moment update $f_i \rightarrow \rho,\ \mathbf{u}$ (see section 2.4)
9 end
```

Algorithm 1: Pseudocode of the iteration loop of the LBM.

3.2 Parallelization

3.3 Software quality

Static typing

Python is a dynamically typed language. That is that the Python interpreter does type checking only in runtime and the type of a variable is allowed to change. The opposite of dynamic typing is static typing. It is introduced by *PEP 484* in Python. In static typing, the types of the variables are checked before runtime and the change of types is generally not allowed. Note, as an exception type casting is a way to change the type of a variable in many languages.

Dynamic typing allows for rapid prototyping and thus it enables fast software development. On the other side static typing can help to catch errors due to type errors, document the code and help to build a cleaner software architecture. The last point in particular ensures that the programmer thinks about the types of the variables and uses the correct types. Thus, in any larger project typing is critical to build and maintain clean code.

Unit testing

One key component of every software project is extensive testing of the software. To this end, we implement several unit tests in order to validate the expected behavior of the implemented functions. We integrate the unit tests into Continuous Integration (CI) using *Travis CI* as build server so that the implementation and its potential unintentional modifications are validated for each commit to the repository.

4

Numerical results

To demonstrate the LBM we conduct several experiments with different combinations of boundary conditions.

4.1 Shear wave decay

To validate the implementation, we compare the simulated solutions with analytical solutions.

We choose the following simulation parameters for our experiments about the planar couette flow:

- lattice grid shape $= 50 \times 50$
- $\omega = 1.0$
- Sinusoidal density in x-direction $\rho(\mathbf{x}, 0) = \rho_0 + \epsilon_\rho \sin(\frac{2\pi x}{l_x})$
  - $\rho_0(\mathbf{x}) = 0.5$
  - $\epsilon_\rho = 0.08$
  - $\mathbf{u}_{initial}(\mathbf{x}) = 0.0$
- Sinusoidal velocity in y-direction $\mathbf{u}_x(\mathbf{x}, 0) = \epsilon_\mathbf{u} \sin(\frac{2\pi y}{l_y})$
  - $\rho_{initial}(\mathbf{x}) = 0.0$
  - $\epsilon_\mathbf{u} = 0.08$
- time steps $= 2000$

4.2 Planar couette flow

The planar couette flow is a steady, laminar flow between two infinitely long, parallel plates with a fixed distance. One of those plates moves tangentially at a velocity of U relative to the other plate, which itself is stationary. The flow is caused by the viscous drag force acting on the fluid.

We choose the following simulation parameters for our experiments about the planar couette flow:

- lattice grid shape = 20×30
- $\omega = 1.0$
- $U = 0.05$
- $\rho_{initial}(\mathbf{x}) = 1.0$
- $\mathbf{u}_{initial}(\mathbf{x}) = 0.0$
- time steps = 4000

4.3 Planar poiseuille flow

The planar poiseuille flow is a steady flow between two non-moving plates. The flow is caused by a constant pressure gradient $\frac{dp}{dx}$ in the axial direction, x, parallel to two infinitely long parallel plates, seperated by a distance h.

We choose the following simulation parameters for our experiments about the planar poiseuille flow:

- lattice grid shape = 200×30
- $\omega = 1.5$
- $p_{out} = \frac{1}{3}$
- $\Delta p = 0.001111$
- $\rho_{initial}(\mathbf{x}) = 1.0$
- $\mathbf{u}_{initial}(\mathbf{x}) = 0.0$
- time steps = 5000

Some error can occure due to:

- Inaccuracy introduced by the bounce-back boundary condition

4.4 Von Kármán's vortex street

4.5 Scaling tests

5

Conclusions

Bibliography

[1] Krüger Timm, H Kusumaatmaja, A Kuzmin, O Shardt, G Silva, and E Viggen. *The lattice Boltzmann method: principles and practice.* Springer: Berlin, Germany, 2016.