Lattice Boltzmann Simulation With Heat Transfer

CFD-Lab
Group H

Summer Term 2018

Introduction

The code consists of the implementation of the Lattice Boltzmann Method for 3D simulations of weakly compressible flows. LBM has its origins in statistical mechanics. In this approach, one solves the Boltzmann equation for the particle distribution function. This function describes the probability of finding fluid molecules in a specific part of space, and having a specific velocity. Then, the macroscopic variables such as velocity and pressure are obtained by evaluating the hydrodynamic moments of the particle distribution function. This is done by integrating the distribution over the velocity space.

The Boltzmann equation considers a collision operator, which is a second order differential operator representing the effect of intermolecular collisions within the fluid. This collision operator has been linearized with the BGK approximation.

The D3Q19 model for the velocity vectors in the 3D space.

The code offers the possibility to run two different scenarios. The scenarios are: Driven Cavity and Natural Convection.

The Driven Cavity implementation followed the procedure in the Worksheet 2 from previous terms in the CFD-Lab course.

The implementation of Heat Transfer in the simulation for the Natural Convection scenario followed the procedure proposed by Chih-Hao et al. in “Thermal boundary conditions for thermal lattice Boltzmann simulations”.

In this approach one considers additionally from the particle density distribution function, a particle energy distribution function, which has the same D3Q19 velocity scheme. This distribution follows the same streaming an collision steps as the particle density distribution function. The computation of the temperature can be obtained in a similar form as the density, with discrete analogs of integral expressions of the particle energy distribution function.

The geometry of the domain and the information about boundary types is stored in a Flag vector. The initialization of the Flag vector for each scenario is completed by a specialized function. - The Driven Cavity scenario has one moving wall in the upper z-face, free-slip conditions in the y-faces and no-slip conditions on the x-faces and the bottom z-face. - The velocity boundary conditions are free-slip in the y-faces and no-slip on the rest. The temperature boundary conditions are Dirichlet to the x-faces and adiabatic for the rest.

In both cases the velocity boundary conditions for the y-faces are set to free-slip to be able to compare with 2D simulations on the x-z plane.

The two scenarios which can be parametrized by modification of .dat files. The addition of Heat Transfer introduced a new dimensionless relaxation time for the particle energy distribution function, which controls the rates of approaching the equilibrium of this distribution. To differentiate between the relaxation times we denote as 'tau_f' the one for density distribution and 'tau_g' for energy distribution.

The sign of the gravity value in the .dat files is positive because it contains the direction of the buoyancy force applied to the particles. The buoyancy terms are modeled with the Boussinesq approximation, assuming linear dependency on the temperature given by the thermal expansion coefficient noted by 'beta'.

All the model implementation was developed from the assumption for the grid and stepsizes both being equal to one. The length chosen in the .dat files for the domain will correspond to the number of cells used.

Code Usage

- Building the code with the Makefile creates the lbsim executable.
- There are two possible scenarios:
  - Driven Cavity:
    - One can modify its parameters in cavity.dat.
    - To run: ./lbsim cavity.dat
  - Natural Convection:
    - One can modify its parameters in convection.dat.
    - To run: ./lbsim convection.dat

Output Folders

- Each simulation is creating an output folder for .vtk files.
- The name of the folder is the name of the scenario.
- The output folder do NOT have to be cleaned, removed or manually created before any simulation.

Results:

The code written has been used to simulate two typical CFD scenarios, the driven cavity and the natural convection. For the driven cavity scenation, a side by side comparison between Lattice Boltzmann and Navier stokes is shown (Figures 1 and 2). Although both methods lead to expected behaviour of fluids, the LBM result appears to be simulating a much more viscous liquid. This is due to the two methods using a very different parameter format: we hope that in

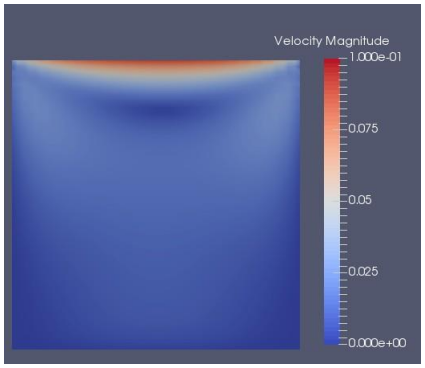

Figure 1: Driven Cavity, LBM: Velocity at t=tend

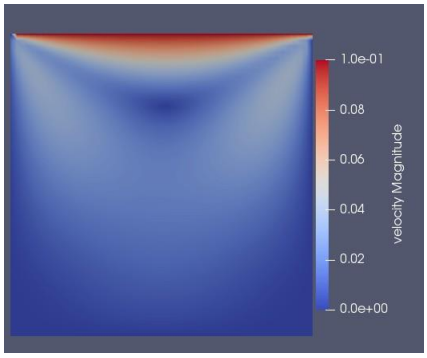

Figure 2: Natural Convection, NS: velocity at t=tend

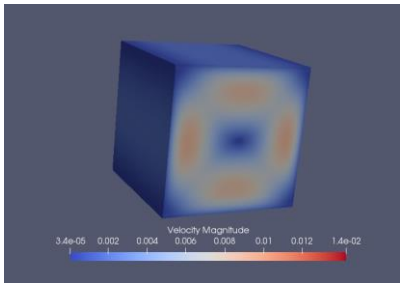

Figure 3: Natural Convection: Velocity at t=80

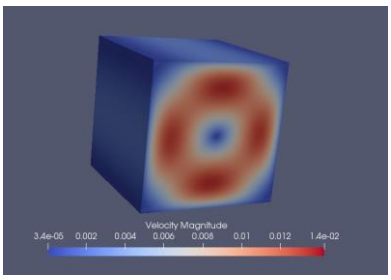

Figure 4: Natural Convection: velocity at end of simulation

the final presentation we will have understood better the mapping between the parameter sets, however for the time being this is a work in progress. Figures 3 and 4 show the velocity behaviour of the natural cavity scenarion in 3D, in all three dimensions. The temperature behaviour is shown in Figures 5 and 6, however this time only a slice of the cube, in the direction of interest. It can be seen that the results look stable and very similar to Navier Stokes results from Worksheet 2. Finally, figures 5 and 6 display the velocity behaviour of the fluid, only on a slice of the cube: a circular movement is what we expect, due to the warm liquid rising to the top and cold fluid sinking to the bottom.

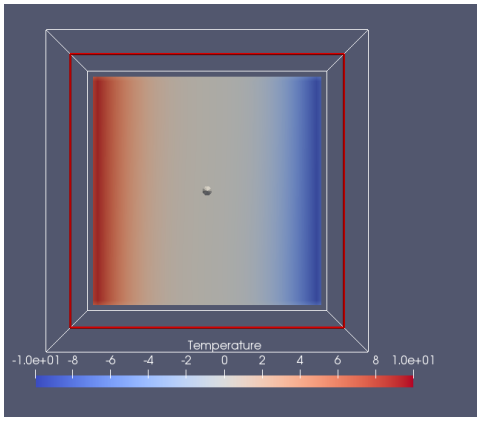

Figure 5: Natural Convection, inner slice: temp at t=50

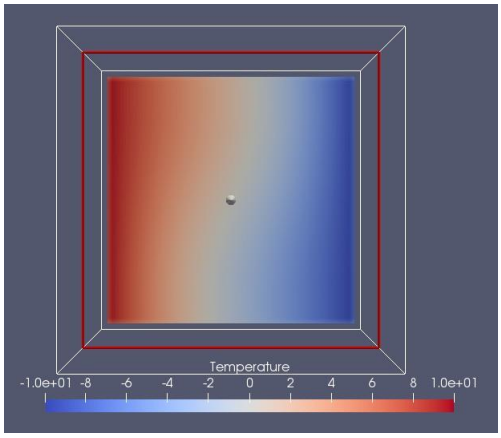

Figure 6: Natural Convection, inner slice: temp at t=tend (equilibrium)

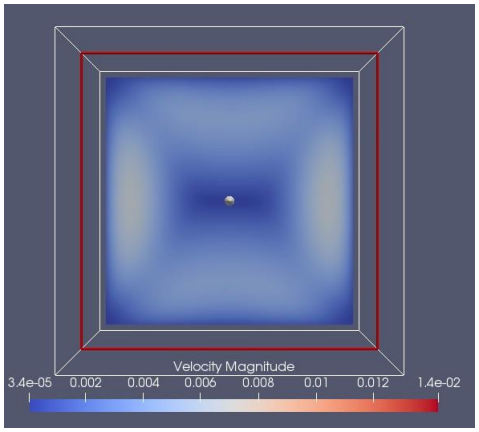

Figure 7: Natural Convection: Velocity at t=40

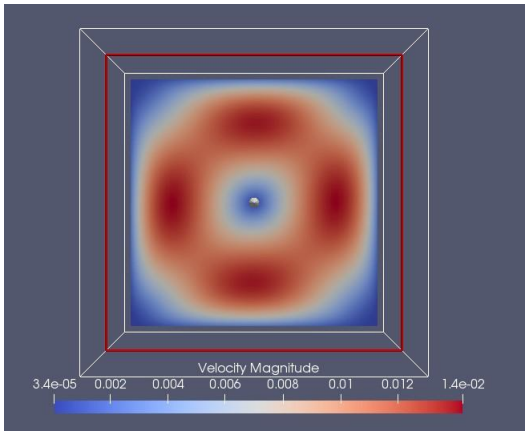

Figure 8: Natural Convection: velocity at end of simulation

Algorithm:

```
initializeVariables(...)
readParameters(...);
allocateMemory(...);
initializeFields(...);
treatBoundary(...);
while( t <= t_end){
        doStreaming(...);
        swapField_f();
        swapField_g();
        doCollision(...);
        treatBoundary(...);
        if(t%tpp ==0){
            writeVtkOutput(...);
}
free(...);
```