Lattice Boltzmann Exercise Report

Exercise 2: Taylor-Green Vortex with LBGK

The vortex remains stable until about time step 50'000. After that it starts to decay as can be seen in Figure 1.

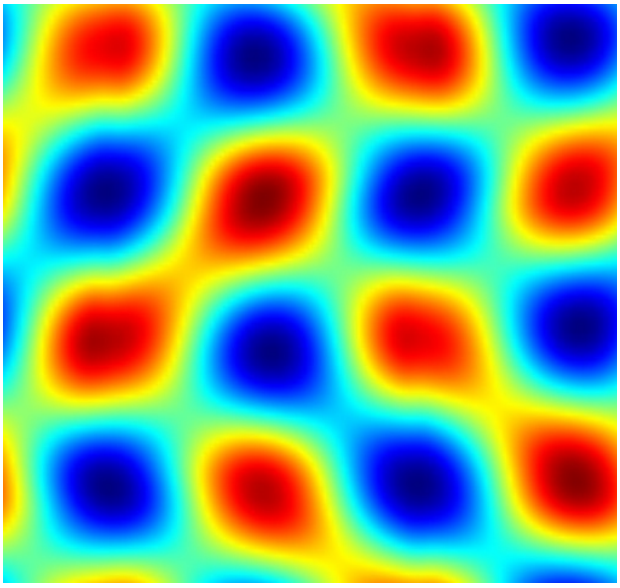

Figure 1: Instability of the Vortex (Re = 3000, Vmax = 0.05, L = 128 , t ~ 60000)

The expected second order convergence holds as can be seen in Figure 2. The lower t (the simulation time) is, the better the behaviour of the convergence is for smaller values of L.

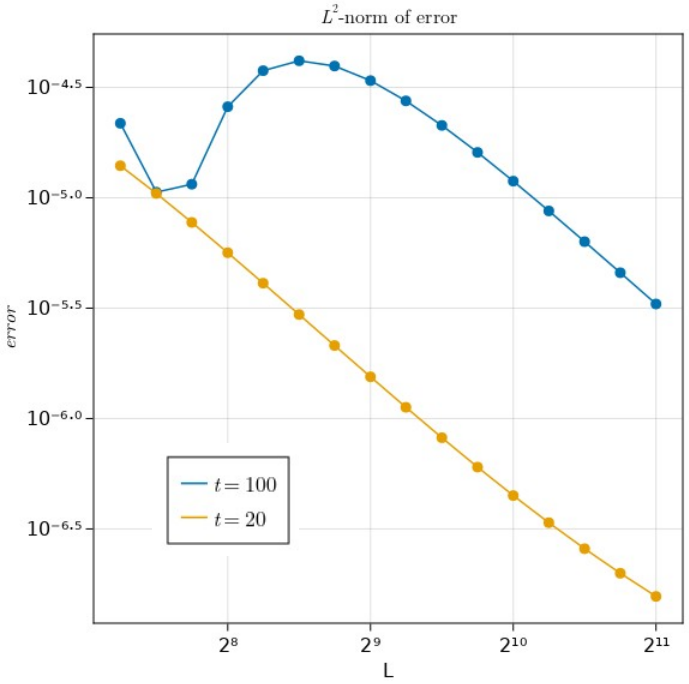

Figure 2: Convergence (Vmax = 0.05, Re = 30000)

How to reproduce:

```
$ tar -xvf LB2D.tar.gz
$ cd LB2D
$ mkdir build
$ cd build/
$ cmake ..
$ ./LB2D taylor_green
```

or

```
$ ./LB2D convergence_lbgk
```

The first mode runs a visual simulation while the latter runs a grid refinement study.

To tweak[1] simulation parameters edit the `taylor_green()` function in `runner.hpp`

I tested the the code using clang-13.0.0 on linux using at most C++17 features. The following cmake `find_package` scripts are assumed to be available:

```
- GLEW, GLUT, OpenGL, OpenMP
- X11 (Linux only)
```

1 Infos about the simulation parameters can be found in the appendix.

Exercise 3: Periodic Shear Layer using KBC

Comparing Figure 5 and 6, we see that KBC-D is unstable (spurious vortices appear along the bottom-right and top-left shocks) at Re=300’000 and L=128, whereas at L=512 it remains stable.

Re=3’000 poses no problem to either resolution.

LBGK simulations at Re=300’000 and L=512 diverge. So the benefits of KBC seem obvious to me.

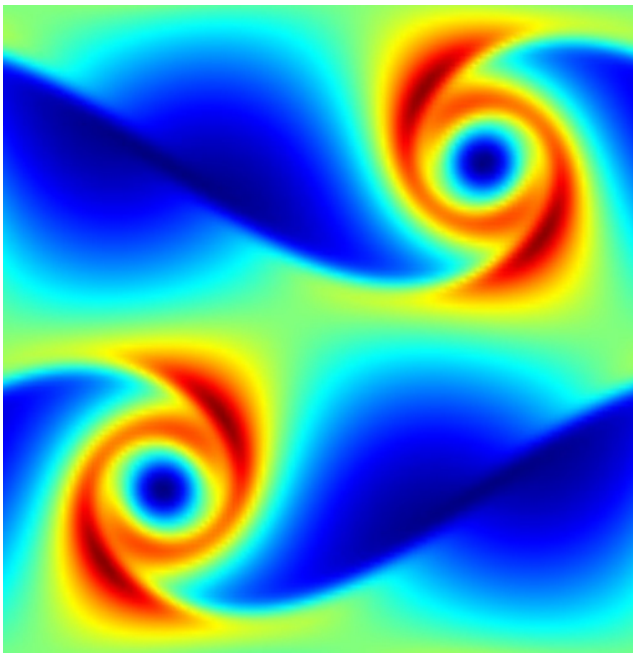

Figure 3: Re = 3’000, L = 128

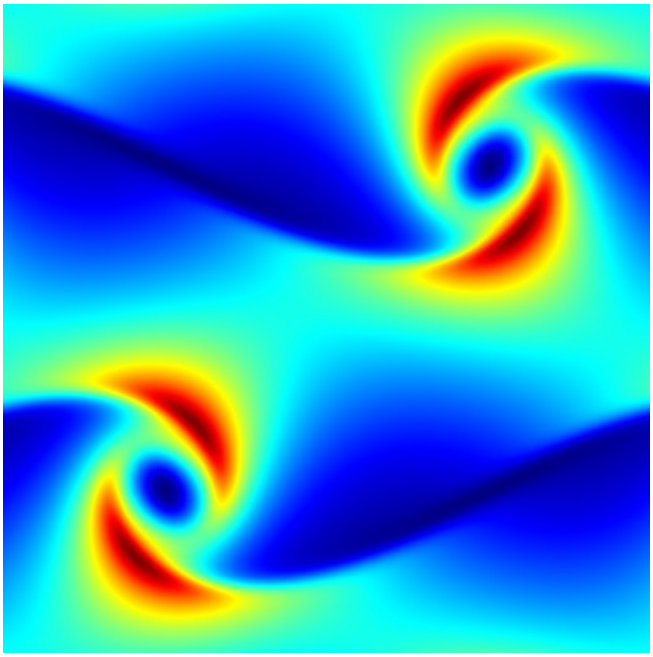

Figure 4: Re = 3’000, L = 512

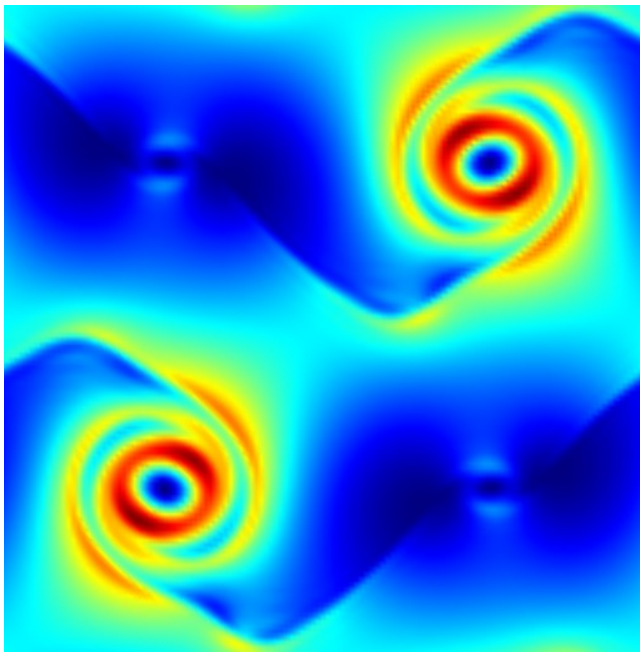

Figure 5: Re = 300'000, L = 128

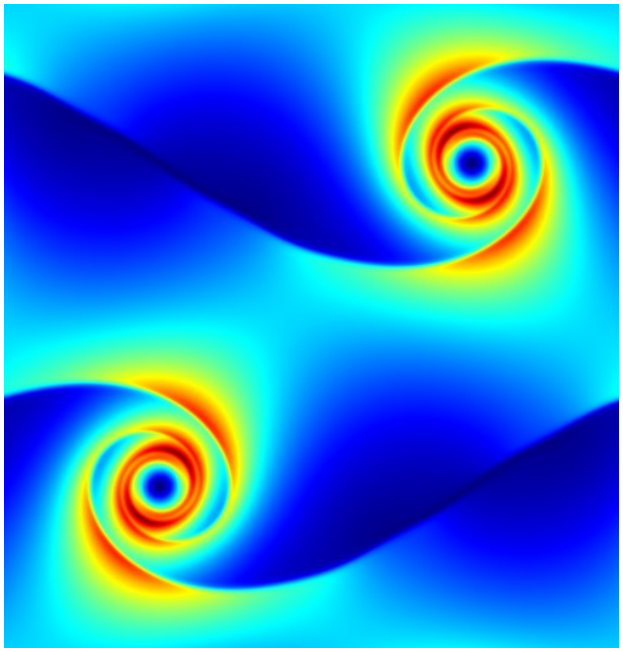

Figure 6: Re = 300'000, L = 512

Setting gamma = 1/beta produces simulations which look promising at first, but then suddenly diverge after some time.

How to reproduce:

Perform the steps outlined in exercise 2, then run:

```
$ ./LB2D shear_layer
```

To tweak simulation parameters edit the `shear_layer()` function in `runner.hpp`.

Exercise 4: Flow around a cylinder

As mentioned in the exercise notes, the proposed outflow boundary conditions only work as long as they are far enough downstream. I assume that this is due to instability occurring when relatively large vortices enter the outflow cells.

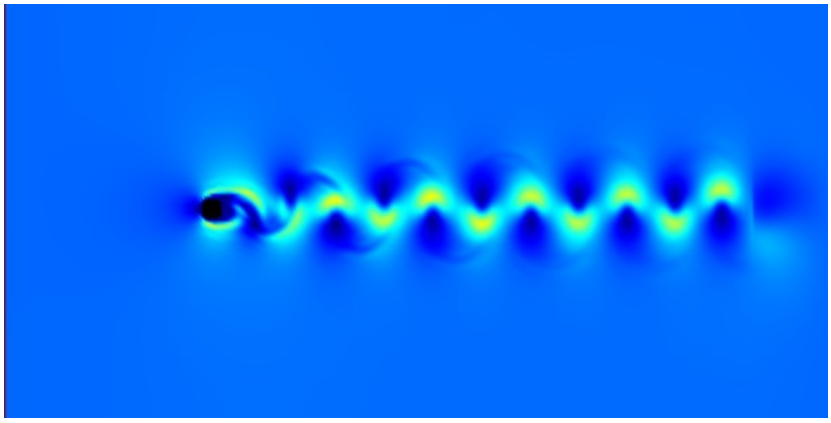

Figure 7: Kármán vortex street (D = 7.5, Re=100’000)

When just using the default cylinder with the given dimensions, this issue does not arise. However, user-placed walls can lead to turbulence (see Figure 8) occuring too close to the boundary which will lead to divergence. To fix this, I set beta to 0.5 in proximity of the outflow boundary. While this leads to some artifacts downstream the stability issue is mostly mitigated.

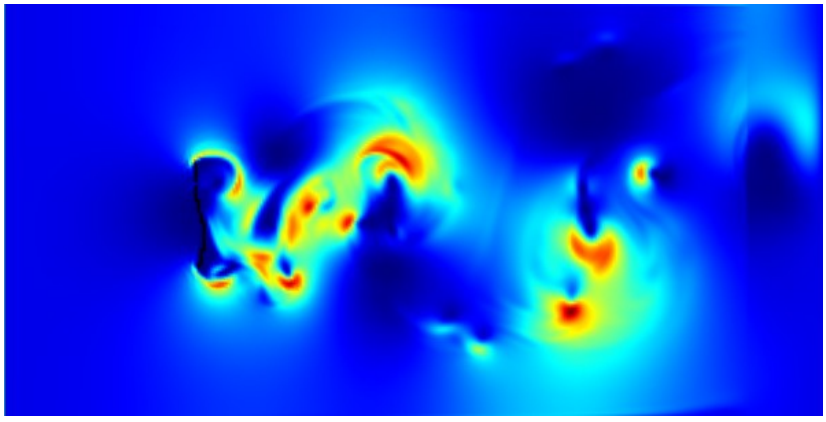

Figure 8: Turbulence occuring after a custom wall (L = 300, Re = 100'000)

Analysis of the Drag Force

To analyze the drag “felt” by the cylinder I implemented the momentum exchange method presented in the exercise notes.

Then I averaged the measured drag over 8000 time steps and plotted it w.r.t. to velocity, yielding Figure 9.

There seems to be a nice quadratic relationship between until some critical mach number around 0.95

As can be seen in Figure 10, the vortices start to clump up for these velocities

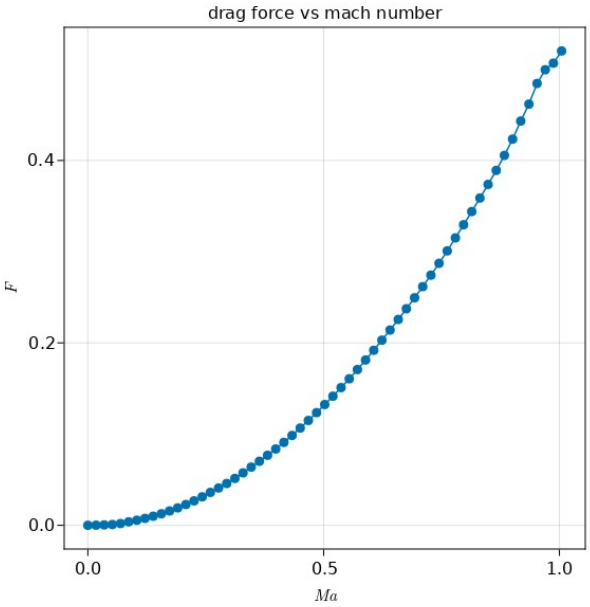

Figure 9: drag force, varying Ma at constant Re = 100000, L = 300

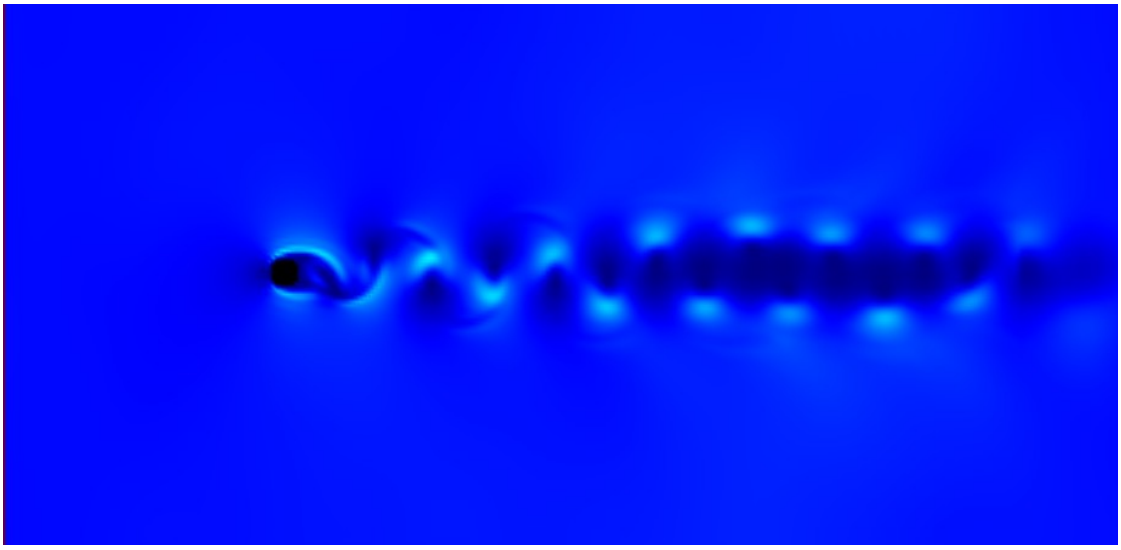

Figure 10: Vortices "clumping up" at high velocities

How to reproduce:

Perform the steps outlined in exercise 2, then run:

```
$ ./LB2D cylinder_flow
```

To tweak simulation parameters edit the `cylinder_flow()` function in `runner.hpp`

Appendix: How to use the new simulation options.

The `lb::simulation` constructor:

Now takes a new optional last element collisionType. It can be either `lb::CollisionType::LBGK`, `lb::CollisionType::KBC` or `lb::CollisionType::None`. The latter mostly exists for debugging purposes.

Simulation Members to be used after construction:

- **`bool sim.periodic`:**
If on, regular periodic boundary conditions are used, otherwise the setup for exercise 4 is enabled.

- **`bool sim.regularized`:**
Overrides the calculation of gamma in the KBC collision operator with 1/beta.

- **`bool sim.calculate_wall_force:`**
Used for the drag vs mach plot. Increments the simulation's speed by 0.01 every 10000 steps, measuring the average force and printing it out at the end.

- **`void initialize(T initializer)`**:
Used to initialize simulations. Expects a lambda with signature `(const lb::simulation& sim, int i, int j)` returning a tuple `(scalar_t u, scalar_t v, scalar_t rho, bool wall)`, describing the initial conditions at a given location and whether a wall should be placed or not.

- **`void calculate_beta(T initializer)`**:
Recalculates the viscosity and beta after a change to velocity.

Exercise 1: 1D-LBM

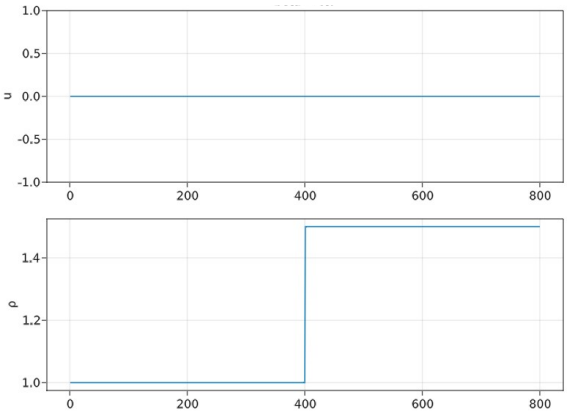

Figure 12: Initial Conditions

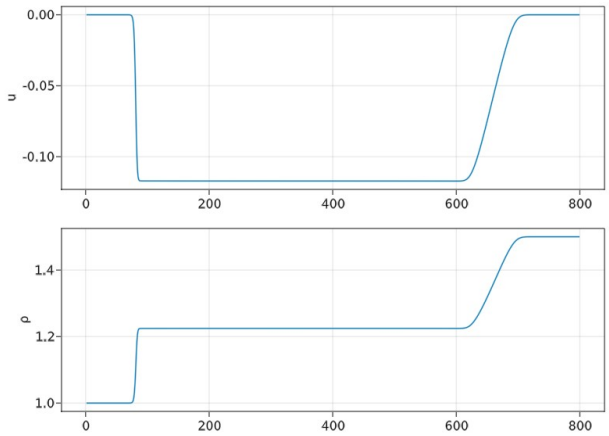

Figure 11: beta = 0.7

I only got around to implementing the shock tube part of the exercise:

We can see numerical instabilities for low viscosity (Figure 13) and smooth solutions for higher viscosity (Figure 12).

To reproduce this simulation run:

```
julia ex1/1d_lbm_shock_tube.jl
```

Julia will complain about missing libraries, but will give instructions on how to install them.

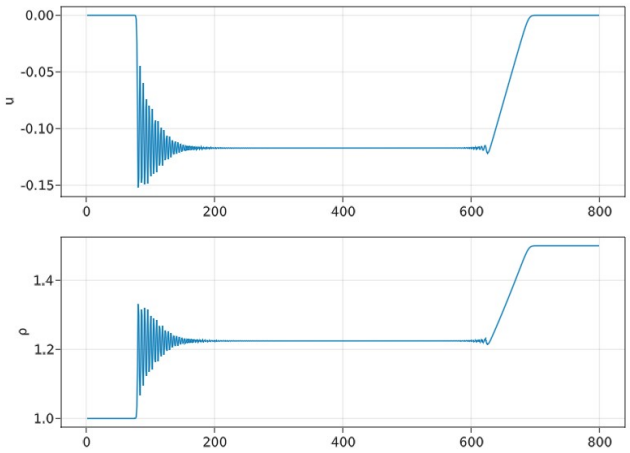

Figure 13: beta = 0.99