Master Thesis Experiment Report IV : Optimisation and Bounceback

Jérémie Guy

September 26, 2024

Abstract

Code optimisation and fine-tuning

1 Context

In this experiment, we will focus on optimizing the code and fine-tuning the bounceback behavior. In particular, we will look at the streaming step and bounceback step of the main loop of the simulation. The details of both have been covered in previous reports, so we will not go into too much detail in this one.

2 Problem

The streaming up until now follows the standard by going through each node, and for each direction calculates the next location to which the PDFs have to go. We will instead use the Numpy roll function to do so.

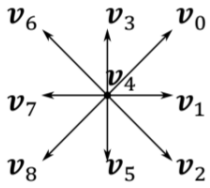


Figure 1: Directional velocity in a D2Q9 lattice, **Source :** [1]

3 Experiments

3.1 Experiment 1

3.1.1 Description

We laid the roll for every direction: we determined which axis is relevant for any direction and manually assigned them (based on the velocity vectors illustrated in Figure 1) to bring the PDFs to their next cell destination. Moreover, for the diagonal directions, we used two rolls in a row (one for horizontal movement and one for vertical movement). The details are shown in the next Python code section.

```
# streaming step
# previous
for x in range(nx):
    for y in range(ny):
        for i in range(9):
```

```
            if flags[x,y] == 0 or flags[x,y]==1:
                next_x = x + v[i,0]
                next_y = y + v[i,1]

                if next_x < 0:
                    next_x = nx-1
                if next_x >= nx:
                    next_x = 0

                if next_y < 0:
                    next_y = ny-1
                if next_y >= ny:
                    next_y = 0

                fin[i,next_x,next_y] = fout[i,x,y]
```

```
# streaming step
# optimized
# i = 0
fin[0,:,:] = roll(roll(fout[0,:,:],1,axis=0),1,axis=1)
# i = 1
fin[1,:,:] = roll(fout[1,:,:],1,axis=0)
# i = 2
fin[2,:,:] = roll(roll(fout[2,:,:],1,axis=0),-1,axis=1)
# i = 3
fin[3,:,:] = roll(fout[3,:,:],1,axis=1)
# i = 4
fin[4,:,:] = fout[4,:,:]
# i = 5
fin[5,:,:] = roll(fout[5,:,:],-1,axis=1)
# i = 6
fin[6,:,:] = roll(roll(fout[6,:,:],-1,axis=0),1,axis=1)
# i = 7
fin[7,:,:] = roll(fout[7,:,:],-1,axis=0)
# i = 8
fin[8,:,:] = roll(roll(fout[8,:,:],-1,axis=0),-1,axis=1)
```

Additionally, we made a quick change to the initial velocity: previously the initial velocity was the expected curve injected into the left-side border. We changed it so that it would scale with the inlet, regardless of its size and placement on the left-side border of the lattice.

Finally, we changed the bounceback step behavior making it more efficient. Previously, the bounceback step went through every coordinate, and for the nodes that were defined as bounceback using a flag array, it changed for every direction the input PDFs of a cell into its output in the same direction (standard node-based bounceback behavior). Instead, we used the flags as a mask that allowed us to bypass the double *for* loops checking every coordinate, making the code significantly faster. Details are shown below.

```
# Bounce-back condition
# previous
for x in range(nx):
    for y in range(ny):
        if flags[x,y] == 1:
            for i in range(9):
                fout[i,x,y] = fin[8-i,x,y]
```

```python
# Bounce-back condition
# optimized
for i in range(9):
    fout[i, flags] = fin[8-i, flags]
```

The changes have been tested on a small 15x11 system. The code runs over 500 iterations. The system structure is detailed in Figure 2.

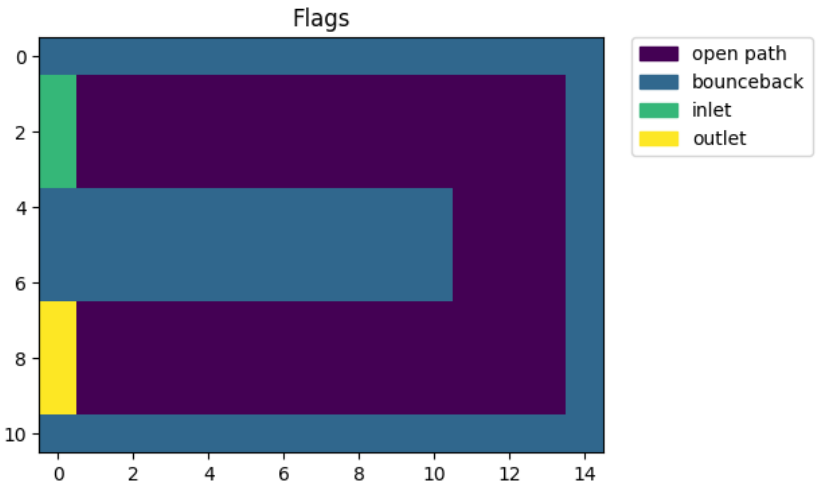
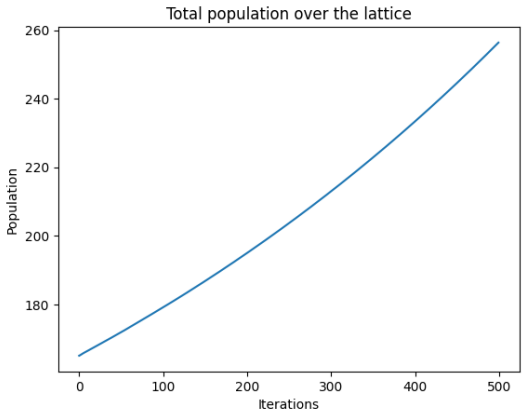


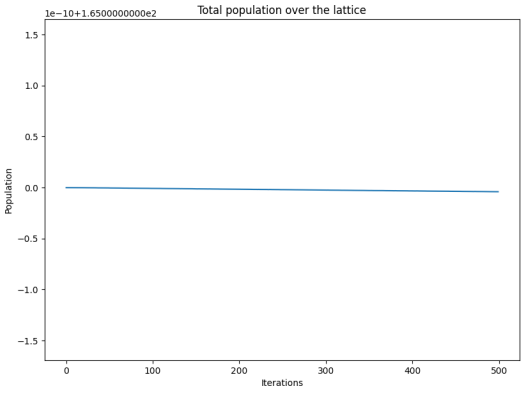
Figure 2: System Structure

3.1.2 Results

The modifications mentioned above resulted in a normal flow state. However, to check the results, we looked at the total sum of the population of the lattice throughout the iterations. The results are shown in Figure 3a. As expected, there is a slow and steady increase in population induced by the Zu-He border condition. As a matter of precaution, we also tested the same system, but without any inlet or outlet. The results shown in Figure 3b show that the results are almost constant: there is a very slight decrease of the PDF values at magnitude e-10, which can be considered negligible.



(a) Population sum with inlet & outlet



(b) Population sum with no inlet & outlet

Figure 3: Population Sums

3.2 Experiment 2

3.2.1 Description

During the streaming step, both the previous code and the new rolls possess a cyclic behavior: what goes out of the system from one side, comes back on the opposite side. The counter that effect, we first started by initializing the bounceback nodes differently. For every bounceback node, we set the PDFs values of the directions facing away from the open path of the system to 0 : this means that no values should cycle (only zeros). We tried as well to compenstate the roll cyclic behavior for specific directions, given the system detailed in figure 2. In this case that means, for directions 6,7 and 8 (as detailed in Figure 1), the PDFs going out of the inlet and outlet site are expected to go around and end in the bounceback nodes on the right border of the system. To counter that we set the incoming PDFs of said border for the three relevant directions to 0. Details are given below.

Initial bounceback conditions :

```
def setBBnodeDirToZero():
    # top border
    fin[[1,2,4,5,7,8],:,0] = 0
    # right border
    fin[[3,4,5,6,7,8],nx-1,:] = 0
    #  bottom border
    fin[[0,1,3,4,6,7],:,ny-1] = 0

    # obstacle Top except rightmost
    fin[[0,1,3,4,6,7],0:10,4] = 0
    # obstacle bottom except rightmost
    fin[[1,2,4,5,7,8],0:10,6] = 0
    # obstacle right border except corners
    fin[[0,1,2,3,4,5],10,5] = 0
    # obstacle top right corner
    fin[[0,1,4,3],10,4] = 0
    # obstacle bottom right corner
    fin[[1,2,4,5],10,6] = 0
    # obstacle inside
    fin[:,0:10,5] = 0
```

Roll compensation during streaming step :

```
# compensate roll
fin[[6,7,8],nx-1,:] = 0
```

3.2.2 Results

The results are unfortunately not satisfactory. For every case we described above, the systems PDFs grow exponentially due to some numerical instability. We will need to further assess if its relevant to continue to compensate the cyclic behavior and initial conditions or if it doesn't influence the system badly enough.

4 Conclusion

5 References

[1] Jonas Lätt. "La méthode de Boltzmann sur réseau pour la simulation des fluides". In: ().