Lattice Boltzmann Report

Ben Morgan 25th April

# Part 1

Part one of the project implements a local version of the taylor green function which calculates the velocity and density at a particular lattice slice. This is then used by the taylor\_green.cpp file to simulate a vortex which can then be plotted by the taylor\_green\_postprocess.py file.

The code I wrote for this section is shown below:

Text

Description automatically generatedMy implementation of the code first calculates ky, kx, and the decay time. The integer indices are then adjusted to yield their respective lattice coordinates. The previous values can then be used to calculate the velocity in the x and y directions at the lattice slice. They are then also used to calculate the density at the lattice slice.

This code generated a csv file which was then plotted resulting in the following plot:

Graphical user interface

Description automatically generated

This plot gives some insight into the vortex that was generated. The top left pane shows the density of the vortex with respect to the x and y coordinates. The bottom left shows the vorticity. The top right pane shows the velocity profile across the vortex. The bottom right plot shows how the error in density and velocity changes over time.

# Part 2

In part 2 the fluid props and equilibrium functions were written. My code for fluid props is shown below:

Text

Description automatically generatedThe dot products of velocity.velocity and cj.velocity were calculated. These were then used to calculate the equilibrium distribution at each value of j.

My code for the local version of fluid props is shown below:

Text

Description automatically generatedMy code loops through the f list and sums the values it stored for each value of j to calculate the density. It then loops through the list again and calculates ux and uy.

There is also a global version of fluid props which I edited. The code I added is shown below:

Text

Description automatically generatedIt loops through the lattice x and y coordinates and gets the field and distribution indexes. It then checks if the site is an obstacle. If it is fluid, it calculates fluid props. If it is an obstacle, it zeros out the fluid properties.

The taylor-green cpp file can then be ran and post processed generating a new set of vortex plots which are shown below.

Graphical user interface

Description automatically generated

The new code does not change the outputs of the plots and they are all exactly the same as in the original implementation.

# Part 3

In part 3 collision and streaming were implemented. The code I wrote for collision is shown below:

Text

Description automatically generatedText

Description automatically generated

My code loops through all the lattice sites and checks if the site is in fluid. If the site is fluid fluid\_props() is ran to calculate the fluid properties. Equilibrium is then run to calculate the equilibrium distribution. The equilibrium distribution is then iterated by j value and the main lattice is updated.

My code for streaming is shown below:

// ------ YOUR CODE HERE ------  
 //looping through all lattice sites  
 for(unsigned int x = 0; x < sim.Nx; ++x)  
 {  
 for(unsigned int y = 0; y < sim.Ny; ++y)  
 {  
 size\_t id = field\_index(sim,x,y); // Index for field array for (x,y).  
 //checks to see if the point is inside an obstacle (false means fluid true means obstacle)  
 if(sim.mask[id] == false)  
 {  
 //looping through all the values of j  
 for(int j = 0; j < sim.Nv; ++j) {  
 //find new x and y coordinates  
 unsigned int new\_x = (sim.Nx + x + sim.cx[j])%sim.Nx;  
 unsigned int new\_y = (sim.Ny + y + sim.cy[j])%sim.Ny;  
  
 //updates the new site in the lattice with the new population of particles  
 fnew[distrib\_index(sim,new\_x,new\_y,j)] = f[distrib\_index(sim,x,y,j)];  
  
 }  
 }  
 }  
  
 }  
}

It iterates through the lattice and checks if the site is fluid. If this is the case feq is iterated through by j and the new x,y coordinates of the site that is streamed to are calculated. The new site is then updated.

The taylor\_green.cpp file was then ran again simulating a new vortex. This was plotted and resulted in a plot which was once again identical to the plot in part 1. The plot is shown below:

Graphical user interface

Description automatically generated

# Part 4

In Part 4 streaming\_reflect was updated and Poiseuille.cpp was edited. The Poiseuille plotting function was also changed to create more plots.

The new version of streaming\_reflect is shown below:

Text

Description automatically generatedText

Description automatically generated

The function now also accounts for reflections if the site which is streamed to is fluid the function acts in the same way but if it is a mask site the reflection is computed. The sites which are reflected to are then updated.

The code I added to Poiseuille.cpp is shown below:

Text

Description automatically generated

The code iterates through all the lattice sites and sets the fluid properties rho ux and uy to their initial values. It then also sets the mask values to true where y = 0 and where y = Ny-1. This creates two obstacles which span the length of the top and bottom of the lattice.

When the file was ran it simulated fluid flow between the two obstacles. A plot of this fluid flow is shown below.

Timeline

Description automatically generated with low confidence

The first plot shows the density of the fluid between the walls. Density is uniform throughout the tube. The second plot shows the velocity of the fluid. Velocity is greatest at the centre of the tube and smallest by the walls. This makes sense qualitatively as you can imagine the fluid being slowed by friction with the walls meaning the further from the walls the fluid is the higher its velocity. The final plot represents vorticity. The magnitude of vorticity is greatest at the walls and smallest at the centre of the tube. The values are negative at the bottom wall and positive at the top wall.

The change of velocity with respect to the y coordinate was investigated and was found to fit to a parabolic relationship. This was plotted below:

Chart

Description automatically generated

The parabola has a maximum at the centre of the tube and moves down to near zero at the edges. The points 0 and 49 are not plotted because these are inside the obstacle and thus there is no fluid to be considered.

The near zero velocity at the walls alludes to the classic laminar flow example where dust is not blown off the blades of a fan because of the low fluid speed at the soid-fluid interface.

# Part 5

In part 5 fluid flow is simulated around an obstacle. The turbulence.cpp file was edited, and the changes made are shown below:

Text

Description automatically generatedText

Description automatically generated

The fluid properties at t=0 were initialised with a degree of randomness. A matrix of seeded pseudo-random values of the same shape as the lattice is generated. The program then iterates through the lattice and assigns the starting fluid properties. The x velocity and rho are set to u0 and rho0 but the initial y velocity is calculated using the random matrix. An obstacle is then created by setting certain values in the mask array to true. If a mask array site is not in the obstacle it is set to false.

When this program is run the turbulence of a fluid flowing around an object is simulated.

A python plotting script was written and is shown below:

A screenshot of a computer

Description automatically generated with medium confidence

It plots a graph at each time step showing how the flow evolves over time. The simulation was run for vortex shedding and these plots are shown below:

Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated Chart

Description automatically generated

You can see vorteces forming behind the obstacle as the fluid flows around it. The simulation was also ran using the parameters for unseperated flow the final plot is shown below.

Chart

Description automatically generated

The obstacle distorts the fluid flow but the fluid quickly returns to normal furthur down the plot. The flag colour map was used in this plot to emphasise the small changes in the vortex

The simulation was also ran with the foppl parameters the resulting final plot is shown below. Chart, surface chart

Description automatically generated

The distortion in the fluid extends furthur but the fluid still quickly returns to normal.

Finally I included another plot of the vortex shedding simulation but with another colour map because I think it really nicely demonstrates the shapes which are formed by the vorteces.

Chart, scatter chart

Description automatically generated