Creating a Lattice Boltzmann Fluid Solver in TensorFlow

Vikram Singh

July 2019

1 The Lattice Boltzmann Method

The Navier-Stokes equations for viscous flow are very complicated, and impossible to solve analytically in many situations. Thus in order to analyse fluid flow in irregular systems, we have to use numerical solvers. One such method is the Lattice Boltzmann Method [1]. This involves simulating the fluid as populations of particles moving on a discrete lattice with discrete velocities, and simulating collisions at each lattice point. The full method is detailed in [2].

2 Potential advantages of TensorFlow

Many of the operations performed by a Lattice Boltzmann solver are done across large arrays. TensorFlow is used extensively in Machine Learning because of its efficient implementation of operations like this, that are less expensive in time and space than those implemented in Numpy. This efficiency could be used to improve the speed of a Lattice Boltzmann solver. In particular, the calculation of macroscopic variables, and the collision with an obstacle, can be expressed as convolutions of an array, which is implemented very well in TensorFlow. I did some experiments to investigate the potential improvements that TensorFlow could give.

3 Analysis

To analyze the benefits of TensorFlow in a solver, I performed some tests using solvers I wrote using both TensorFlow and only Numpy. My code for these solvers is available at (`https://github.com/vikram8128/PythonLatticeBoltzmann`).

3.1 Methodology

My tests consisted of 6 different simulations, created in both TensorFlow and Numpy versions. These were:

1. Flow around a cylinder with low viscosity (420×180)
2. Flow around a cylinder with high viscosity (420×180)
3. Flow around an airfoil (420×180)
4. Flow through a narrowing pipe (420×180)

5. Flow through a bending pipe (1000×452)
6. flow in a lid driven cavity (180×180)

The simulations ran for 20000 timesteps, producing .png output every 100 timesteps. Each of these simulations was run on an Intel Skylake server, with access to 32 CPU cores.

3.2 Results

The simulations were run 3 times each, and the mean real times are shown in Table 1.

Table 1: The wallclock times in seconds taken by each simulation

Simulation	1	2	3	4	5	6
Numpy	316	311	319	1565	4043	615
TensorFlow	198	198	198	199	708	122

In these results, I assumed a generous 10% uncertainty in the time (observed error was around 2%). From this, I created the plot in figure 1.

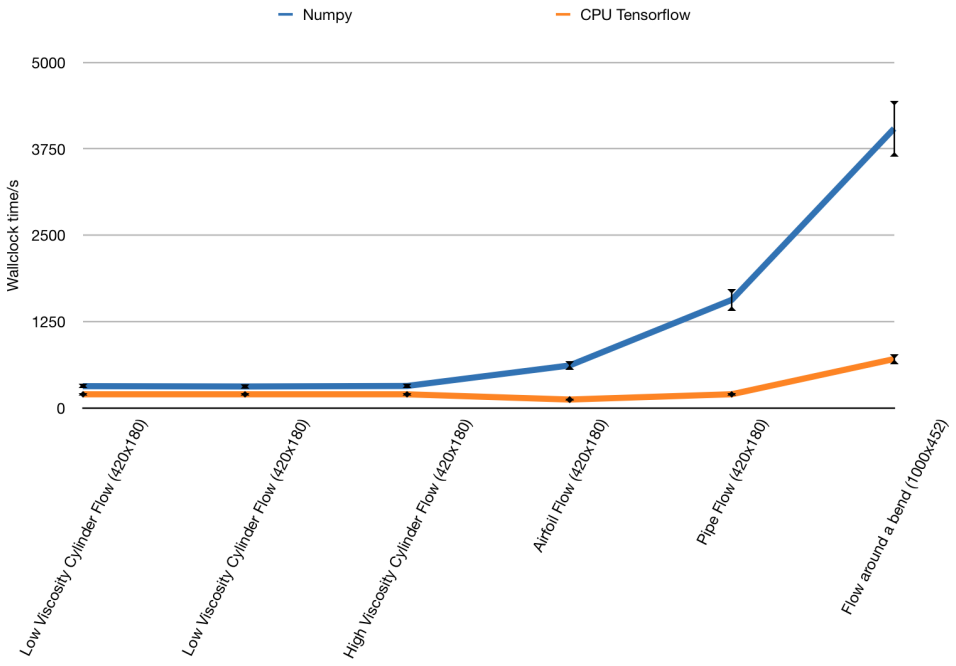

Figure 1: Runtimes of simulations

In addition, I calculated the rate of cell updates (in Megahertz) for each of the simulations, which is shown in table 2.

Table 2: Cell updates per second (MHz) for each of the simulations

Simulation	1	2	3	4	5	6
Numpy	4.78	4.86	4.74	0.966	2.24	1.05
TensorFlow	7.64	7.64	7.64	7.60	12.8	5.31

Again assuming 10% uncertainty in this data, we get the plot of cell update frequency in figure 2.

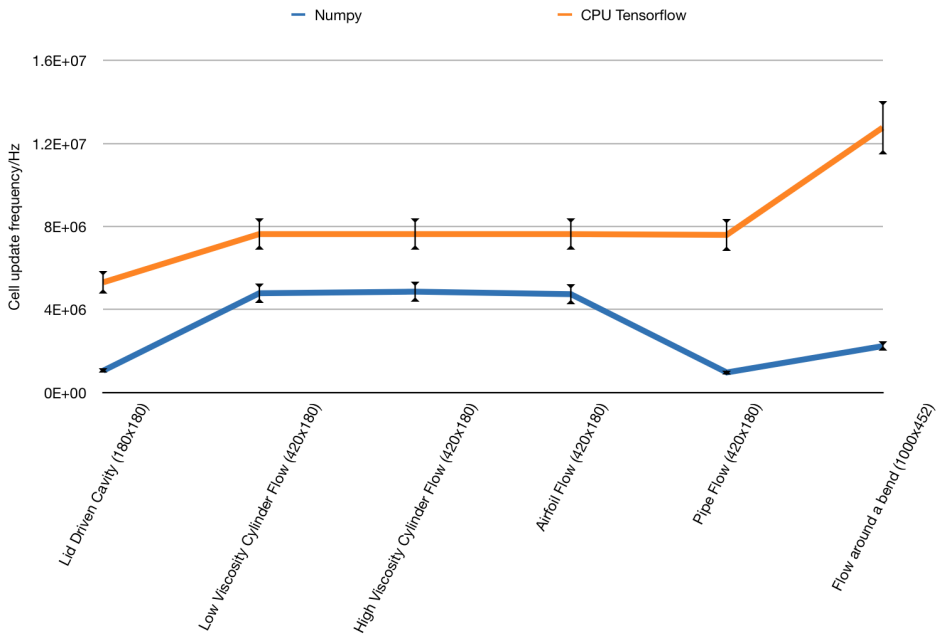

Figure 2: Cell update frequencies of the simulations

3.3 Conclusions

We observe that TensorFlow consistently outperforms Numpy, but the improvement factor is not constant. In the simulations of flow in a narrowing pipe, flow in a lid driven cavity and flow in a bending pipe, TensorFlow is faster by a factor of about 5 times, as opposed to about 1.5 in the other three simulations. These simulations have much larger areas of obstacle, and in addition, the bending pipe simulation has a much larger lattice. We can conclude that TensorFlow scales much better than Numpy with an increased lattice size, and an increased area of obstacle. The improved parallelisation that is implemented in TensorFlow gives a large improvement in speed of the Solver, which scales well with size. This improvement could be quite useful for industrial fluid dynamics simulations.

4 Next Steps

While this project has provided some useful results, there are some areas that it has not explored. I did not explore how the performance of the models translates to 3D models, where the factor of improvement that TensorFlow provides could be different to the 2D simulations that I examined. Additionally, I only used CPUs to run my TensorFlow simulations. Adapting them slightly to run on GPUs may give a

further speed improvement. However, GPU TensorFlow does not support the (*roll*) operation that is used extensively, so this would have to be implemented manually. Both expanding the solvers to 3D, and running the TensorFlow solvers on GPUs would be potentially quite informative, and these require further investigation.

References

[1] Wikipedia article on the Lattice Boltzmann method (`en.wikipedia.org/wiki/Lattice_Boltzmann_methods`)

[2] Alexander J. Wagner (2008) *A Practical Introduction to the Lattice Boltzmann Method*, North Dakota State University

[3] TensorFlow Documentation (`www.tensorflow.org/api_docs/python/`)