Week 6 Progress Report

Abdulmajed Dakkak

July 18th, 2008

Monday

Got the FCHC program to work and explained how to split a CUDA program to Professor Francis and Katie.

Code

- FCHC (.c, xcode.tar.gz)

Tuesday

Fixed an embarrassing bug in the FCHC code (all the indecies were off by one). Dr. Francis told me that there is no reason to pursue FCHC anymore since it branches too much to be placed on a `SIMD` machine. It seemed like the project has ended and I had to pick a new one, but after doing some more research I found a different way of doing 3D fluid simulation using Cellular automata — the Lattice Boltzmann Method. The 3D version, namely D3Q19, is used frequently to model free surfaces. The collision could also be written using no `if` statements, which makes it ideal for `CUDA`.

The only disadvantage of LBM is that while it can be thought of as an extension of Lattice Gases and uses CA as its basis, its detached for the CA.

Code

- FCHC.c (fixed)
- D2Q9.c

Wednesday

Started implementing `D2Q9`, and by the middle of the day I had something working, but blew up to infinity after a few time steps. I spent the rest of the day debugging the code.

Thursday

Read a bit about adaptive time stepping and wrote some code to perform it, but quickly realized that the paper I was looking at did not define what mass is, and I did not know. The paper defined mass based on density, and density based on mass, and while density makes sense to define, mass was not. Just before the visit to Wolfram, I did get the `d2q9` code to work. The problem with going to infinity was because the equations will blow up if any of the velocity vectors in the cell exceed $\frac{1}{3}$. After setting the initial vectors to $\frac{1}{3}$ and disallowing any vectors to grow over $\frac{1}{3}$, the system works.

The visit to Wolfram was helpful, and he gave me good advice. It seems that only one things was of interest to him, however, and that is CUDA. I described what CUDA is to him, and what work I have been doing. Wolfram said that the best experiment to do is to use FHP and have a system where there are no random choices and only a cylinder as an obstacle. It seems like he wants to see whether turbulence occurs because of the obstacle. He said that there are three scientific camps each believing they know the cause of the turbulence (which is not predicted by the Navier-Stokes equations):

1. The molecular dynamics at the atomics level are noticeable at the fluids level and that is the cause of the turbulence.
2. The initial conditions are the cause of the turbulence. This is the Chaos camp that believes that small variations of the initial conditions have a grave impact on the final outcome.
3. The cellular automata view, which he advocates, is that randomness can grow out of a deterministic system (I did not understand this point as I did for the other two).

He says that while some work has been done in the early 90s to show that his theory works (citing an example where scientists found a periodicity of 2 hours in the turbulence of liquid nitrogen), the work of lattice gases has faded.

While it is clear that he values lattice gases over lattice Boltzmann, he did offer a compelling argument to use LGCA and that is that whatever outcome is produced is due to molecular interaction — no scientist will blame error on numerical inaccuracies. He mentioned that turbulence was for many years

regarded as an numerical error in computation fluid dynamics, but he showed that it wasn't.

He did mention that he worked on a 3D version of LGCA and that it is simple to derive a boolean expression for it, but I have yet to find a paper describing anything but FCHC for 3D LGCA.

After getting the advice from Wolfram, and two copies of the periodic table poster, I went home to sleep. After 2 hours of attempting to sleep, I figured out how to place obstacles in my `d2q9` program and how to perform collisions. I quickly wrote the program down, and it worked.

Code

- D2Q9.c

Friday

Starting placing the `d2q9` in CUDA. While this process is trivial, there are many nuances one has to deal with. First, one has to identify which part of the program must be parallelize — the collision and propagation in my case. Second, one must flatten the array — 3D in my case. Finally, one must copy the data from the CPU to the GPU correctly and have optimization in mind from the beginning — placing unchanging data in constant cached memory. I spent 2 hours to get the function parallelize, the arrays flattened, and the constant data into cache. While the program does not work correctly, it does compile. A few hours were then spent teaching Katie how to parallelize the Mandelbrot set.

Code

- d2q9-cuda(tar.gz)