Week 4 Progress Report

Abdulmajed Dakkak

July 4th, 2008

Saturday/Sunday

Looked over the Schaber method, and figured out how to Schaberize simpleGL. Wrote a rule 90 1D cellular automata in openGL to experiment with C's boolean operators. The 1D cellular automata was then ported to CUDA, although the graphics are off, since I do not know anything about *vertex buffer objects*. Finally, I implemented the FHP model.

Read a little bit about the FCHC model.

Code

- FHP
- Open GL Rule 90
- rule90_cuda.tar.gz (Cuda's Rule 90 CA, Kernel)

Monday

Schaberized the `simpleGL` example included in the CUDA projects, and worked with Chase and Will to migrate fluidsGL from Unix Sockets to Winsocks.

Code

- Schaberized SimpleGL(client, server, kernel)

Tuesday

Decided that I do not care much about Windows, and started my effort to more the `fhp` code from openGL to CUDA. Converted my representation of the lattice from a 2D object to a 1D flat array. It turns out that this might not have been a good choice.

Code

- FHP CUDA (fhp.cu, fhp_kernel.cu, types.h)[**Does not work**]

Wednesday

Joined Will and Chase, again, in researching for ways of making the Schaber method work in Windows. This meant going through the Windows documentation, forums, and some Winsock books trying to find why Winsock does not implement `MSG_WAITALL`.

Formed a pact to stay all night (if we have to) to get fluidGL into Sysgz. In the end we had to and stayed. Around 9AM the next morning the next night we got something that should work in the CAVE.

Thursday

CUDA/FHP

Made the `fhp` program work on the CUDA chip. The problem was with the following piece of code

```
dim3 block(32, 32, 1);
dim3 grid(LATTICE_WIDTH / block.x, LATTICE_HEIGHT / block.y, 1);
collision<<<grid, block>>>(gpu_c1, gpu_c2, gpu_c3, gpu_c4, gpu_c5, gpu_c6,
                           gpu_k1, gpu_k2, gpu_k3, gpu_k4, gpu_k5, gpu_k6,
                           gpu_boundary);
propagate<<<grid, block>>>(gpu_c1, gpu_c2, gpu_c3, gpu_c4, gpu_c5, gpu_c6,
                           gpu_k1, gpu_k2, gpu_k3, gpu_k4, gpu_k5, gpu_k6);
```

From my understanding of CUDA, a few grids (`LATTICE_WIDTH / block.x` to be precises) containing `LATTICE_HEIGHT / block.y` blocks which in turn contain `block.x * block.y` or `32*32` threads would be allocated on the GPU. If that fails, then CUDA will figure everything out and everyone would be

happy. This is not the case, however, and the results were not seen in the form of an error in this function. Rather, when I tried to copy the results after computation.using `cudaMemcpy`

```
cudaMemcpy(c1, gpu_c1, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
cudaMemcpy(c2, gpu_c2, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
cudaMemcpy(c3, gpu_c3, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
cudaMemcpy(c4, gpu_c4, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
cudaMemcpy(c5, gpu_c5, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
cudaMemcpy(c6, gpu_c6, ARRAY_LENGTH, cudaMemcpyDeviceToHost);
```

I was getting the same array as I had before; i.e. `gpu_c[1-6] = c[1-6]`. This meant that I only saw the initial positions of particles, and nothing changed. I knew, however, that something must have changed, because, at the very least, the positions should have propagated.

After a few hours (`4` to be more precise) I figured out that the problem can be fixed by setting the block size to `16*16` rather than `32*32`.

```
dim3 block(32, 32, 1);
```

This was found on the forums, among many other places. This is the first time I also saw the immaturity of CUDA. Take, for example, the following piece of code:

```
int x = powf(2.0, 3.0);
```

In regular mathematics we are taught that $2^3 = 8$, but in CUDA's opinion the answer is 7. This phenomenon continues for other odd powers of 2.

Schaberization

I also stuck around trough the night with Chase and Will helping them in making the Phleet and CUDA talk. Will and Chase did the majority of the work, and I only helped a bit with debugging. Incidentally, the problem they were encountering, namely with some dimensions working while others not, might be related to the problem with odd powers of 2 found in CUDA. Their programs fail, have interesting behavior, or are slow if you use odd powers of two (although we are limited by what we could try).

Code

- FHP CUDA (fhp2.cu, fhp_kernel2.cu, types2.h)

Friday

After the long night (and subsequents sleep), I noticed that the toroidal boundary is not working in `fhp-cuda`. Thus one of my aims today is to fix that issue. Possibly the most important thing, however, is to optimize the code. This means that I have to learn more about CUDA's memory architecture and use Vertex Buffer Objects (or `vbo`); something I dreaded from the beginning.

I also noticed that CUDA's profiler which I downloaded and tried two weeks ago (ans is invaluable for the optimization) now gives a segfault. I will talk to Johnathan on Monday to see if this is related to the update done last week.