PHYS405

Advanced Computational Physics
Parallel Computing

Assignment # 9b
Due: Friday, December 11, 2009

Purpose: Learn the CUDA language.

Note: Please identify all your work.

Background: This part of the assignment asks you to adapt to CUDA a serial code that generates a bifurcation diagram for the logistic map. The logistic map is a map of real line to itself given by

$$x_{i+1} = a - x_i^2.$$

This mapping is ubiquitous in many problems of practical interest and is arguably the simplest example of a (discrete) complex dynamical system (indeed, you'll note its similarity to the equation generating the complex Mandelbrot set).

The variable a is a parameter that is held constant while x is iterated from some initial condition x_0. We are interested in the long term or asymptotic behavior as x_0 is iterated for various values of a. A plot of the asymptotic values of x verses a is called a *bifurcation diagram.*

The reason for this terminology is as follows. The asymptotic behavior often varies smoothly with a. For example, for some a x_0 may tend to some fixed point x^* with the value of x^* varying smoothly with a. However, for another a x_0 could end up in a period two orbit, oscillating between two values x_1^* and x_2^*. The values of these two points may also vary smoothly with a, but there is some transition value $\tilde{a}$ where we jump from the fixed point to the period two orbit. This non-smooth process is called a *bifurcation.* The bifurcation diagram then shows all of these bifurcations on a single plot since we scan over all values of a.

The serial code loops over a and iterates a random initial condition `THRESH` number

of times. This is to let transients "die out" and approach the asymptotic behavior. If an iterate leaves the interval $[-2, 2]$ during this time it will eventually escape to ∞, so the trajectory is thrown out and another random initial condition is tried (it is known that positive measure attracting sets exist for the a values in the program so this loop will eventually terminate).

If a trajectory stays bounded after `THRESH` iterates the next `MAXITER` iterates are tracked. The x-axis is binned into `xRES` number of bins and the `binit` routine is called to find which bin the current point in the trajectory is in. This repeats until `xRES` number of initial conditions have been iterated and binned. The bins are then normalized to a maximum value of one and are then output to the screen. The values in the bins are essentially the density of iterates around various points and plotting them shows the bifurcation structure of the map.

Assignment: First run the serial code and gnuplot script so you can see what it is you're supposed to produce.

```
%>gcc -o logistic logistic.c -lm
%>./logistic > log.dat
%> gnuplot -persist log.p
```

Then adapt the serial code to run on CUDA using the skeleton file `log_skel.cu` included on the assignment page. Note the differences from the serial code. Functions called from a kernel are prefixed with `__device__` and `host` functions *cannot* be called from device functions. The random number generator `rand()` is a host function, so I added my own random number generator for the kernel to use. Finally, the original `binit` sorting algorithm was recursive, but device functions *do not* support recursion, so it has been rewritten without recursion (the while loop functions as the recursion step).

Parallelize over a, so that each thread computes the future orbit for a single value of a. Thus the block and grid need only be one dimensional (note that this allows a maximum of $2^9 \times 2^{16} = 2^{25} \sim 3 \times 10^7$ values of a, which should be sufficient. The kernel function should replace the entire main loop of the serial code. This includes iterating for a value of a, binning the trajectory, and normalizing the bin. The normalized bins should be returned

to the main program for output. Finally, time the CUDA code.

Note that you may keep the various `#define` statements intact so that these parameters need not be explicitly passed to functions.

Extra Credit: The above implementation iterates a random initial condition followed by another if the first escapes the region. For the parameter range given every initial condition either escapes to ∞ or tends to a unique stable bounded attractor (a fixed point, periodic orbit, or "chaotic" Cantor set). In principle a map $x_{i+1} = f(x_i)$ could have more than one coexisting attracting set, so that different initial conditions can tend to distinct bounded asymptotic behaviors, or (Lebesgue almost) every initial condition may excape to ∞.

Modify the CUDA program using an extra dimension of block/threads to assign initial condtions distributed throughout the interval $[-2, 2]$ amongst these threads. Have the various threads bin the bounded trajectories together. Solutions that escape the interval should not be binned.

Test this code on the map $x_{i+1} = a - (a - x_i^2)^2$ and compare against the original code. Are the results different? (Note, this example is the second iterate of the logistic map, so period two orbits of the original become distinct period one orbits of the second iterate map.)