Low level parallel programming Assignment 4

by Max Falk Nilsson

I am in group 16 i did this lab by myself.

Intro

To chose mode between pthreads, omp, cuda and vector type “-mode” then the mode you want to choose (PTHREAD, OMP, VECTOR, CUDA). Type “-thread NUMBER” to add the number of threads to be run with PTHREAD, OMP and VECTOR. CUDA will always run with the same amount of threads (can be changed in the code). To add parallel heat calculations type “-heat” if you have omitted “-heat” the serial version will be run. Type “-col” to use parallel collision detection with 4 threads. “--timing-mode” will run the program without the window and in “timing mode”.

Computer
Processor: AMD Phenom(tm) 8450 Triple-Core Processor.
RAM: 3GB DIMM DDR2 1639 MHz.
Graphics card: GF114 [GeForce GTX 560 Ti].

A.
Two dimensional arrays are trick with cuda and they require the memory to be mapped in a special way. An easier way to have 2D arrays in cuda is to have a fake or a flattened array to simulate a 2D array. Which was the solution I used. This means that there is no need to allocate memory on the GPU just allocate a 2x larger normal array on the Host.

If you are referring to the three updating steps i would say that cuda handles them pretty well they are not embarrassingly parallel but close. You could also improve my implementation by using shared memory and maybe even kernels starting kernels if your hardware supports that.

B.
My partitioning of the problem is a little bit different than the 3 described. I use 4 kernels where one is doing some work on the heatmap and the scaled map. The give me these average times of executing in milliseconds:

update_heat_map	update_heat_map_intes	update_heat_map_norm	update_blurr_map
0,160952381	0,03619961905	5,081787429	11,44069638

update_heat_map_norm is the function which both floors the heatmap values and calculates the scaled values. update_blurr_map updates the blur map.
We can see that both of theses kernels take a long time to execute. There are many reasons for this: we are doing more calculations in these kernels then the two other faster kernels. In the update_heat_map_norm we are also going through cells serially in for every heatmap position.
update_blurr_map is the same thing we are doing more work and here we also have two loops which are run serially for each thread.

C. With serial calculation of desired positions, collisions and heat mapping the run time was approximately: 687 seconds.
With serial calculation of desired positions and collisions but with parallel heat mapping the executions time was: 82 seconds.
Gives us an speedup of approximately 8x.

D. No shared memory was used only local and global. Shared memory could have improved the speed but for lack of time this was not done.