TDT4200 Problem Set 6
CUDA

Graded (counts 8% of final grade)

The assignment is due by Thursday 15/11-2012, at 20:00.

If you cannot meet this deadline, please contact the instructor.

Answers should be submitted on It's Learning.

Your answer should consist of exactly two files: `nbody_cuda.cu` for the CUDA assignment and [username]`.pdf` for the remaining tasks. Do not create any kind of archive. Do not submit other files.

The code should compile and run on the computer lab machines.

Please make sure the code is readable with sensible comments.

1 Theory (20 %)

1.1 Miscellaneous theory (15 %)

a. Name 3 types of overhead for CUDA programs.

b. What is the difference between `mem_fence()` and `barrier()` in OpenCL?

c. Why doesn't CUDA support global synchronization?

d. What is the difference between data and task parallelism?

e. Name at least 3 differences between CUDA and OpenCL (other than terminology).

f. What do OpenCL workitems and workgroups correspond to in CUDA?

g. Name 4 important OpenCL language restrictions, and describe the challenges with them.

1.2 Reduction (15 %)

The following questions are related to the reduction example presented in class.

a. What is algorithm cascading? How was it used in the reduction example in class?

b. Instruction overhead was a likely bottleneck in the reduction example presented in class. What kind of instruction overhead was this?

c. What mechanism was used to completly unroll the loops (for optimization)?

2 CUDA Programming (80 %)

The topic for this task is to implement a 2-dimensional n-body simulation in CUDA.

An n-body simulation is a numerical approximation of the evolution of a system of bodies in which each body continuously interacts with every other body. An example of this is a system of planets interacting with each other through gravitational forces.

For further details, we refer to the current recitation slides, as well as Problem Set 4 and the slides from Recitation 6. As with Problem Set 4, the simulation should be 2-dimensional.

2.1 Implementation (70 %)

The provided archive contains a file `nbody_cuda.cu` with skeleton code for a CUDA implementation, as well as a serial CPU implementation `nbody.c`. The skeleton code contains places where you should fill in missing code. Your task is to finish this program.

There are 7 subtasks to perform:

1. Allocate device memory, and transfer data to the device. (5%)
2. Set up correct kernel calls. (5%)
3. Transfer the results back to the host. (5%)
4. Complete the `update_velocities()` function. It should compute the new velocities for all the planets after 1 time step. It should call `calculate_velocity_change_block`. (15%)
5. Complete the `calculate_velocity_change_block` function. It should compute the *change* in velocity for a planet, due to the interations of a group of other planets. It should call `calculate_velocity_change_planet()`. (15%)
6. Complete `calculate_velocity_change_planet()`. It should compte the *change* in velocity due to one other planet. It should use Newtons law of gravition. (15%)
7. Complete `update_positions()`. It should update the positions of the planets, based on the velocities. (10%)

Even if you choose to do this task on your own machine, please make sure the code also works on the computer lab machines. Your CUDA version might be different.

2.2 Report (10 %)

a. Run the program with n bodies for different values of n (256, 1024, 4096) (use the provided files planets256.txt, planets1024.txt, planets4096.txt), with

1000 timesteps. Report the runtime of the CUDA version, make sure to time calculations and memory copying separately. Calculate the speedup compared to the provided CPU version. (5%)

b. Experiment with different block sizes. Run and time your program with at least 3 different block sizes of your choosing. Explain the differences in running time you observe (if any). (5%)