CSCI-UA.0480-003
Parallel Computing
Lab 3

Before you start:

- To calculate the time of each program in this lab use the Linux command *time*. Report the **REAL** part (nor usr or sys).
- After you login to your CIMS account, you need to ssh to one of the following: cuda1, cuda2, cuda3, or cuda4
- The source code, containing both device and host code, has extension .cu
- You compile with **nvcc** progname.cu
- Don't forget to #include <cuda.h>
- A very useful API is **cudaGetDeviceProperties**() check it up.

Write a reduction program that finds the maximum of an array of M integers. You will need to write a host function that fills the array with random integers between 1 and 100000. The host part must get M from the command line. That is, if your program is called seq, then a command line: *seq 1000000* will form an array of one million elements.

Coding:

A. Write the sequential version of the program in C. Note that the sequential version will scan the array sequentially from start to end. Call it **seq.c**.
B. Write a CUDA version of the program that does not take thread divergence into account. Call it **cudabasic.cu**.
C. Update the version in B to take thread divergence into account. Call it **cudadiv.cu**.
D. Update the program in C to make use of shared memory to reduce global memory bandwidth. Call it **cudashared.cu**. Optimize as much as you can in that level (i.e. try to prefetching, different thread granularity, etc. I do not mean that all the optimizations will work, you need to try).

Note: The shortcut of stopping the search once you find an element of 100000 is not permitted. Both the C version and the CUDA versions must scan the whole array.

Experiments:

Do some testing runs with different values of M to find out:

- At what value of M is cudabasic better than seq version? Try to find the smallest M where this condition is satisfied. → call this M1
- At what value of M is cudadiv better than seq? Try to find the smallest M where this condition is satisfied. → call this M2
- At what value of M is cudashared better than seq? Try to find the smallest M where this condition is satisfied. → call this M3

Reporting Results:

Draw a bar graph that compares the execution time of each of the above 4 versions at the different problem sizes M1, M2, and M3.
That is, x-axis contains M1, M2, and M3. The y-axis contain the *real* time measured.
At each M, draw 4 bars: seq, cudabasic, cudadiv, and cudashared.

Analyzing the results:

Below the graph, answer the following questions:

1. What is the relationship between M and the fact of the GPU version be better than the sequential version? Justify.
2. Did you find that taking care of branch divergence makes the GPU version (i.e. cudadiv) better than the sequential version in lower M than the basic GPU implementation (i.e. is M2 higher or lower than M1)? Justify.
3. Similarly, was M3 higher or lower than M1? Justify.
4. Beside branch-divergence and shared memory usage, what other optimizations have you used? State them in a bulleted list, with a justification of why you think these optimizations are effective for the problem at hand. If you haven't used any other optimizations, they say so and justify why you thought that the other optimizations wouldn't be effective here.

Note: The justification parts in the above questions have much higher grade than just stating the relationship.

What to submit (in single zip file called yourlasname.firstname.zip)?

- 4 source code files (seq.c, cudabasic.cu, cudadiv.cu, and cudashared.cu)

- 1 pdf file for your conclusions of #3
- email the lab to the grader by the deadline.

Have Fun!