LSU EE 7700-2 Homework 1 Due: 28 February 2011

To complete this assignment follow the setup instructions from the course Web page. The setup instructions bring you to the point where you can compile the cpu-only examples but for this assignment that won't be necessary. Next follow the Programming Homework Work Flow instructions to check out this assignment. These instructions describe an assignment from an earlier semester. For this semester, the main routine is in file `stream-2.cc` *and the executable (provided to gdb) is named* `stream-2`*. To solve this assignment both* `stream-2.cc` *and* `stream-2-kernel.cu` *will have to be edited.*

This code runs a simple stream program using CUDA, the same program used in classroom examples. It will run the code for a variety of block and grid sizes, though not necessarily enough to answer all the problems here. (You will have to modify the code to solve the problems.) See the comments in `stream-2.cc` *for details on how the code works.*

Problem 1: Compile and run the code unmodified. When the code is run the available GPUs will be listed. Find a machine with two GPUs and use that for this assignment. One should be of compute capability 1.3 and the other should be $2.x$. Indicate:

- The name of the machine you are running on.
- The names of the GPUs.
- The manufacturer's claimed memory bandwidth for each GPU.

To answer the last item above look for the manufacturer's specifications. That information is not provided in the program output.

Problem 2: As we know, performance will be below its peak potential if there are an insufficient number of threads in a multiprocessor. Call the minimum number of threads needed to reach peak performance $T(n_{\rm B})$, where $n_{\rm B}$ is the number of blocks per multiprocessor. (To clarify, performance will be below peak if the number of threads is less than $T(B)$ and the performance will not be higher than peak if the number of threads is greater than $T(B)$.)

Use the code for this assignment to determine whether $T(n_{\rm B})$ does not depend on $n_{\rm B}$ (the number of blocks), whether $T(n_{\rm B})$ is smaller (a good thing) when $n_{\rm B}$ is smaller or whether $T(n_{\rm B})$ is smaller when $n_{\rm B}$ is larger.

(a) Modify `stream-2.cc` as needed to run the grid and block sizes needed to answer the question.

(b) Answer the question above about $n_{\rm B}$. Indicate the configurations you ran and the results and comment on your confidence in the answer given the data collected and experiments performed.

Problem 3: The code contains three kernels, `dots_loopless`, `dots_stride_large`, and `dots_stride_small`. The original code just launches `dots_stride_large`, in this problem `dots_loopless` will be launched, and `dots_stride_small` is for the next problem.

As its name suggests `dots_loopless` does not contain a loop. It can be run if the total number of threads is equal to the number of array elements (by default 2^{20}). However, the code as written will never use it.

(a) Modify routine `dots_launch` so that `dots_loopless` is run if the number of array elements per thread is one, otherwise `dots_stride_large` is run.

(b) Run experiments to determine if performance is any better running `dots_loopless` than it is running `dots_stride_large` when there is one iteration per thread. Describe the experiments (block sizes, etc) and results.

(c) Provide a possible reason for the results in the last part.

Problem 4: Modify routine `dots_stride_small` so that the array elements accessed by a block are contiguous. For example, if there are 1000 array elements and 10 blocks then block 0 should access 0-99, block 1 should access 100-199, etc. **Be sure that the code still runs efficiently.**

Modify `dots_launch` so that it calls `dots_stride_small`. The `stream-2.cc` routine will be print an error message if the code executes incorrectly, look out for these. CUDA will give an error message if `idx` is out of range.