Hints for debugging GPU programs

- The CUDA calls return error status. Remember to check the error status and report any errors after every call – this will help you localize where an error occurred. The kernel call itself does not return error status; you nonetheless should check for errors on this too (as shown in the skeleton code for the labs).
- Set the Windows environment variable CUDA_LAUNCH_BLOCKING to 1. To do this, open a Windows Command Prompt window, and type “setx CUDA_LAUNCH_BLOCKING 1”. You must do this before you launch Visual Studio. Why do this?
  - It tells CUDA to wait for one GPU access to finish before starting the next GPU access. If you do not do this, then any attempt to benchmark how much time is spent in, e.g., computation vs. data copying will be incorrect.
  - Similarly, if you do not do this, then the CUDA error calls may be reported for a later call than the one that actually incurred the error.
Note that this is specific to you (it will not affect other users on the same PC), and is specific to one PC (you will have to do it again when you switch to a different PC). However, it will stick when you log onto the same PC on a different day.
- Use printf() statements. CUDA does support printf(). Note that printing is done from the CPU rather than the GPU; thus, CUDA buffers the printf() output and returns it to the CPU for printing. CUDA does not directly support C++ printing with cout; however, you can always use an ostrstream and then call printf() on its str() function if you like.
- Windows has a GPU timeout. If your GPU program, for some reason, is extremely slow (longer than about 5 seconds), then Windows will kill it. You will get a message like “unspecified launch failure” or “the launch timed out and was terminated.” This usually means that your code has some very blatant performance issue.
- Older GPUs (including the ones in our labs) are quite slow at double-precision arithmetic. Be sure to use float rather than double.
- Visual Studio has a memory checker that will catch most accesses to illegal memory. Turning it on can be very helpful. It requires several steps:
  - First, turn it on. The Nsight menu has an item labeled “Enable CUDA memory checker,” which does not seem to do anything! Instead, choose Nsight → Options. Then select “CUDA” on the left, and make sure “Enable Memory Checker” is set to True.
  - Next, be sure that it can tell you the exact line number where the problem occurs. From the Project→Properties menu, select Cuda C/C++ on the left, then select Device as the subsection of Cuda C++. Finally, set “Generate Line Number Information” to “Yes (-lineinfo)”.
  - Next, run your program. Instead of using the normal Visual Studio “Debug” → “Start without debugging,” you will use Nsight → Start CUDA debugging (legacy). Note that you must stick with the legacy debugger and not the next-gen debugger, since the latter is incompatible with our Maxwell-generation GPUs.
  - A small window may pop up and announce that you have memory errors. If so, you can see them by selecting the Output window, and then make sure its “Show output from” box says “Nsight” (as opposed to Show output from Debug). The errors will tell you the address that was accessed illegally, whether the illegal access was on a load (“ld”) or a
store ("st"), whether it was an illegal access to global memory ("gl") or shared memory ("sh"), and what the blockIdx and threadIdx values were for the illegal access.

- Note that the pop-up program-output window typically stays up at the end of your program run until you type any character. However, in memory-debugger mode the window will immediately vanish when your program is done. Thus, if your program errors out almost immediately you may never even see the pop-up program-output window.

- Some people have noticed that, after the first compiler error, the compiler seems to get into a state where it no longer recognizes any of the CUDA extensions, and thus complains about them. If this happens to you, just fix the first errors and then try again. Exiting and re-entering Visual Studio may help here too.