Lab 1: Device Query

Objective

The purpose of this lab is to introduce students to the CUDA hardware resources along with their capabilities. You are not expected to understand the details of the code but should understand the process of compiling and running code as the same process will be used in subsequent labs.

Instructions

The code provided queries the GPU hardware on the system. Specifically, the following hardware features are reported:

- GPU card's name
- Global memory size
- Size of shared memory per block
- Number of registers per block
- Warp size
- Maximum number of threads per block
- Maximum block dimensions
- Maximum grid dimensions
- Constant memory size
- Compute capability version
- Clock rate

We will be using the *CMake* tool to compile and build the code. Steps for compiling and building:

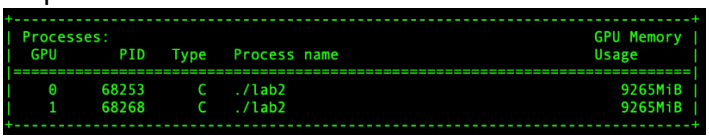
1. Connect to Rosie. If you are off-campus you have to be connected to the VPN first (use GlobalProtect). In this lab, we are going to work with a command-line interface. The following are some options for **ssh** clients in Windows:
    i. MobaXterm: https://mobaxterm.mobatek.net
    ii. Putty: https://www.putty.org
    iii. Git bash: https://git-scm.com
    iv. Windows Subsystem for Linux: https://docs.microsoft.com/en-us/windows/wsl/install-win10

    Type one of the following, in your favorite **ssh** client to connect to one of Rosie's management nodes:

    ```
    # connect to mgmt4
    $ ssh username@dh-mgmt4.hpc.msoe.edu
    ```

The one-time password on all of the accounts is **g0-raid3rS** (note the upper case S and the zero symbol after g). You should then be asked for a new password when you sign in via **ssh**.

2. Create a folder named source under your home directory. This is where all your lab implementations should be saved. In the source folder create a subfolder named lab-1.
3. Copy main.cu and CMakelists.txt from /data/cs4981_gpu_programming/lab-1/ to your lab-1 directory.
4. Connect to one of the compute nodes using: **ssh dh-nodeN**, where N = your_msoe_id_number % 18 + 2.
5. Create a folder named build under your home directory. This is where you should compile and build all your labs. Do not put any source code in this directory. In the build folder create a subfolder named lab-1.
6. Run CMake from your lab-1 subdirectory of the build folder: cmake ~/source/lab-1/. If successful then this should generate a Makefile.
7. Build your code by running this command: **make**. If successful, this should generate an executable named lab1. Otherwise, fix the reported compilation errors and run make again.
8. Check if someone else is using one of the four GPUs available in your compute node by running: nvidia-smi. If any processes are running then they will be listed at the end of the report, for example:



Note that you can use nvidia-smi -l N, where N is the number of seconds after which the report will be continuously updated (this can be quit using Ctrl-C). If any processes are running then run your code on a GPU that is not being used. The figure above shows that GPUs 0 and 1 are being used, out of 4 available. In this case, we could choose to run our code on GPU 3 as follows: CUDA_VISIBLE_DEVICES=3 ./lab1.

If no other processes are running then the executable generated as a result of compiling the lab can be run using the following: ./lab1. If successful, this should print the specifications for the GPUs in that node.

Now, modify the code so that it outputs the GPU specs to a file instead of the console. The name of the file should be given as input on the console. For example, if your executable is named lab1 then your program should be able to run using the following command:

./lab1 name_of_output_file.txt

Questions

1. What is the compute capability of the NVIDIA Turing architecture?
2. Suppose you are launching a one-dimensional grid and block. If the hardware's maximum grid dimension is 65535 and the maximum block dimension is 512, what is the maximum number of threads that can be launched on the GPU?
3. Under what conditions might a programmer choose not to want to launch the maximum number of threads?
4. What can limit a program from launching the maximum number of threads on a GPU?
5. What is shared memory?
6. What is global memory?
7. What is constant memory?
8. What does warp size signify on a GPU?
9. Is double-precision supported on GPUs with 7.5 compute capability?

Submission Instructions

You should upload a PDF of the answers to your questions, the source code, the output .txt file, and the binary executable to Canvas as part of a zipped folder named your-msoe-id-lab1.zip

Rubric

Code that crashes without producing output or code that doesn't compile is not considered to produce results and will be graded accordingly in the rubric.

CPU implementation **_____ / 30**

correct result (code compiles and executes)	_____ / 10
implementation / correct coding	_____ / 15
profiling/timing results	_____ / 5

GPU implementation **_____ / 50**

correct result (code compiles and executes)	_____ / 10
implementation / correct coding	_____ / 35
profiling / timing results	_____ / 5

Comments **_____ / 10**

Questions **_____ / 10**