CSUS COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

CSc 196U – Parallel Programming with GPUs
Fall 2018 / Dr. Muyan

<u>IMPORTANT NOTE:</u>

Do NOT submit A0 via SacCT.
Its purpose is to make sure you have a good understanding of the assignment environment and you are ready to solve real assignments.

However, A0 is a subset of A1:
DeviceQuery and VectorAdd problems will also be included in the A1.
Hence, you can start thinking about how to answer questions of DeviceQuery /VectorAdd and update template.cu of VectorAdd as we progress in lectures.

Note1: DeviceQuery is presented with its solution (the student does not need to update its code). Hence, in A1 submission, the student should not include DeviceQuery's template.cu and executable file.

Note2: VectorAdd in A1 may have different datasets than the ones presented in A0. Make sure that you test your solution with all datasets presented in A1 before submitting your A1.

Assignment#0 (DeviceQuery, VectorAdd)

Steps for solving the assignment (refer to "Assignment Environment" lecture slides for details):

- Download the assignment from the SacCT and unzip it (now you have "A0" directory - you must have already done this to read this a0-instruction.pdf file).
- Build the source using CMake (in "build" directory that you create under "A0" directory).
- Copy datasets directly under "build" directory (under "A0/datasets" directory, we have two dataset directories, copy each of them to "A0/build", e.g., copy contents of "A0/datasets/VectorAdd" to "A0/build/VectorAdd").
- For each problem in the set:
    - Set the project properties as indicated in the below "Project Setup" instructions (e.g., set command line options).
    - Update the template code (template.cu) as indicated in the below "Coding" instructions to solve the problem.

    - Build your solution under Visual Studio (right click on the project and hit “Build”) and run it under Visual Studio (right click on the project and hit “Debug -> Start new instance”.
    - Make sure your executable (located under “Debug” directory that resides under “build” directory) works from the command line as indicated in the below “Command-line Execution” instructions.
    - Create a new directory by giving it the name of the problem.
    - Copy the executable file (.exe file) and the updated template code (template.cu) to this newly created directory. If the problem has questions, also copy the pdf file that includes your answers to these questions.

- Zip all newly created problem directories, name this ZIP file as YourLastName-YourFirstName-a#.zip (e.g., Doe-Jane-a0.zip).
- Build and test your assignment in a lab machine (see the syllabus for tips) and **create a TEXT (i.e., not a pdf, doc etc.) file called "readme.txt"** that indicates the lab machine you have used to test your assignment. You may also include additional information you want to share with the grader in this text file.
- (This step is not necessary for A0) Submit the **TWO files you have created above (zip file and txt file) SEPERATELY to Canvas** (i.e., do NOT place the txt file inside the zip file). You will receive the grader comments on your text file when grades are posted.

Problem#1: DeviceQuery

Objective

The purpose of this problem is to introduce the student to the CUDA hardware resources along with their capabilities. The problem is presented with its solution (the student does not need to update its code). Hence, in the submission, the student should not include DeviceQuery’s template.cu and executable file.

Coding

The problem is presented with its solution. Hence, the student does not need to add extra lines of code.

The code provided queries the GPU hardware on the system. Specifically we log the following hardware features:

- GPU card's name
- GPU computation capabilities
- Maximum number of block dimensions
- Maximum number of grid dimensions
- Maximum size of GPU memory
- Amount of constant and share memory
- Warp size

Note: wbLog is a provided logging API (similar to Log4J).The logging function wbLog takes a level which is either OFF, FATAL, ERROR, WARN, INFO, DEBUG, or TRACE and a message to be printed.

Project Setup

- DeviceQuery does not have any test data. But we save the output of our execution on its empty data directory.

- Right click on the DeviceQuery_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
> .\DeviceQuery\Dataset\0\result.txt
```

(You will see the output of your execution in `result.txt` which resides under `build\DeviceQuery\Dataset\0\`)

Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\DeviceQuery_Template.exe`) can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\DeviceQuery_Template > .\DeviceQuery\Dataset\0\result.txt
```

Questions

1) 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 threads can be launched on the GPU?

2) Under what conditions might a programmer choose not want to launch the maximum number of threads?

3) What can limit a program from launching the maximum number of threads on a GPU?

4) What is shared memory?

5) What is global memory?

6) What is constant memory?

7) What does warp size signify on a GPU?

Problem#2: VectorAdd

Objective

The purpose of this problem is to introduce the student to the CUDA API by implementing vector addition. The student will implement vector addition by writing the GPU kernel code as well as the associated host code.

Coding

Edit the template.cu to perform the following:

- Allocate device memory
- Copy host memory to device
- Initialize thread block and kernel grid dimensions
- Invoke CUDA kernel
- Copy results from device to host
- Free device memory
- Write the CUDA kernel

Instructions about where to place each part of the code is demarcated by the `//@@` comment lines in the template.cu.

Project Setup

To test your program on test_case_0:

- Right click on the VectorAdd_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\VectorAdd\Dataset\0\output.raw

-i .\VectorAdd\Dataset\0\input0.raw,.\VectorAdd\Dataset\0\input1.raw

-o .\VectorAdd\Dataset\0\myoutput.raw -t vector

> .\VectorAdd\Dataset\0\result.txt
```

(all in one line - do not forget to put spaces between the sub-lines)

You will see the output of your execution in `result.txt` which resides under `build\VectorAdd\Dataset\0\`

Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\VectorAdd_Template.exe`) can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\VectorAdd_Template -e .\VectorAdd\Dataset\0\output.raw

-i .\VectorAdd\Dataset\0\input0.raw,.\VectorAdd\Dataset\0\input1.raw

-o .\VectorAdd\Dataset\0\myoutput.raw -t vector

> .\VectorAdd\Dataset\0\result.txt
```

(all in one line - do not forget to put spaces between the sub-lines)

Questions

(1) How many floating operations are being performed in your vector add kernel in terms of N, the size of the vector? explain.

(2) How many global memory reads are being performed by your kernel in terms of N, the size of the vector? explain.

(3) How many global memory writes are being performed by your kernel in terms of N, the size of the vector? explain.

(4) Name three applications of vector addition.