CSUS COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

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

Assignment#1 (DeviceQuery, VectorAdd, BasicMatrixMultiplication, TiledMatrixMultiplication)

Due Date: Monday, March 5th

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 "A1" directory - you must have already done this to read this a1-instruction.pdf file).
- Build the source using CMake (in "build" directory that you create under "A1" directory).
- Copy datasets directly under "build" directory (under "A1/datasets" directory, we have four dataset directories, copy each of them to "A1/build", e.g., copy contents of "A1/datasets/VectorAdd" to "A1/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 and build your solution.
    - 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.
- Build and test your assignment in a lab machine (see the syllabus for tips) and create a readme.txt that indicates the lab machine you have used to test your assignment.
- Zip all newly created directories, name this zip file as YourLastName-YourFirstName-a#.zip (e.g., Doe-John-a1.zip) and submit it to SacCT.

Each dataset directory provides **one or more test cases**. You should test your program with all the provided test cases by updating the command line options provided in the below “Project Setup” instructions. For instance, to test *test_case_1* of VectorAdd (located at “build\VectorAdd\Dataset\1”) you should update the command line option provided below (which tests *test_case_0* located at “build\VectorAdd\Dataset\0”) by changing all subtexts that say “…\Dataset\0\...” with “…\Dataset\1\...”.

Specific Problem Instructions:

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) Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

(5) Name three applications of vector addition.

Problem#3: BasicMatrixMultiply

Objective

Implement a basic dense matrix multiplication routine. Optimizations such as tiling and usage of shared memory should not be incorporated.

Coding

Edit the code in the code tab to perform the following:

- set the dimensions of the product matrix
- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- write the CUDA kernel

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

Project Setup

To test your program on test_case_0:

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

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

-i

.\BasicMatrixMultiplication\Dataset\0\input0.raw,.\BasicMatrixMultipli
cation\Dataset\0\input1.raw

-o .\BasicMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

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

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

You will see the output of your execution in `result.txt` which resides under

`build\BasicMatrixMultiplication\Dataset\0\`

Command-line Execution

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

```
.\Debug\BasicMatrixMultiplication_Template

-e .\BasicMatrixMultiplication\Dataset\0\output.raw

-i

.\BasicMatrixMultiplication\Dataset\0\input0.raw,.\BasicMatrixMultipli
cation\Dataset\0\input1.raw

-o .\BasicMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

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

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

Questions

(1) How many floating operations are being performed in your matrix multiply kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(2) How many global memory reads are being performed by your kernel in terms of the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(3) How many global memory writes are being performed by your kernel in terms of the following variables: *numCRows* and *numCColumns*? explain.

(4) Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

(5) Name three applications of matrix multiplication.

Problem#4: TiledMatrixMultiply

Objective

Implement a tiled dense matrix multiplication routine using shared memory.

Instructions

Edit the code in the code tab to perform the following:

- set the dimensions of the product matrix
- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- write the CUDA kernel

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

Project Setup

To test your program on test_case_0:

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

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

-i

.\TiledMatrixMultiplication\Dataset\0\input0.raw,.\TiledMatrixMultipli
cation\Dataset\0\input1.raw

-o .\TiledMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

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

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

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

Command-line Execution

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

```
.\Debug\TiledMatrixMultiplication_Template

-e .\TiledMatrixMultiplication\Dataset\0\output.raw

-i
.\TiledMatrixMultiplication\Dataset\0\input0.raw,.\TiledMatrixMultipli
cation\Dataset\0\input1.raw

-o .\TiledMatrixMultiplication\Dataset\0\myoutput.raw -t matrix

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

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

Questions

(1) How many floating operations are being performed in your matrix multiply kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(2) How many global memory reads are being performed by your kernel in terms the following variables: *numCRows, numCColumns*, and *numAColumns*? explain.

(3) How many global memory writes are being performed by your kernel in terms of the following variables: *numCRows* and *numCColumns*? explain.

(4) Describe what further optimizations can be implemented to your kernel to achieve a performance speedup.

(5) Compare the implementation difficulty of this kernel compared to the *BasicMatrixMultiply* problem. What are the new code additions that programmers can make errors with this implementation?

(6) Suppose you have matrices with dimensions bigger than the max thread dimensions. Describe an approach that would perform matrix multiplication in this case.

(7) Suppose you have matrices that would not fit in global memory. Describe an approach that would perform matrix multiplication in this case.