CSUS COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

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

Assignment#3 (ListScan, Histogram)

Due Date: Monday, April 23rd

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 "A3" directory - you must have already done this to read this a3-instructions.pdf file).
- Build the source using CMake (in "build" directory that you create under "A3" directory).
- Copy datasets directly under "build" directory (under "A3/datasets" directory, we have two dataset directories, copy each of them to "A3/build", e.g., copy contents of "A3/datasets/ListScan" to "A3/build/ListScan").
- 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 (e.g., ListScan).
    - 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 and readme.txt in a file named as YourLastName-YourFirstName-a#.zip (e.g., Doe-John-a3.zip) and submit it to SacCT.

Each dataset directory provides **five 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 ListScan (located at "build\ListScan\Dataset\1") you should update the command line option provided below (which tests test_case_0 located at "build\ListScan\Dataset\0") by changing all subtexts that say "…\Dataset\0\..." with "…\Dataset\1\...".

Specific Problem Instructions:

Problem#1: ListScan

Objective

The objective of this problem is to implement a kernel to perform an **inclusive** parallel scan on a 1D list. The scan operator will be the addition (plus) operator. You should implement the **work efficient scan kernel** and handle input lists that have sizes larger than the maximum thread block size as discussed in the lecture notes. You can assume that the input list length will be at most 10000 elements.

The boundary condition can be handled by filling "identity value (0 for sum)" into the shared memory of the last block when the length is not a multiple of the thread block size.

The device computation should be split into two kernels: one that computes scanned blocks and one that adds scanned block sums to all values of the scanned blocks.

Coding

Edit the template.cu to perform the following:

- allocate device memory
- copy input host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernels
- copy output device memory to host
- deallocate device memory
- write the CUDA kernels

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 ListScan_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\ListScan\Dataset\0\output.raw
-i .\ListScan\Dataset\0\input.raw
-o .\ListScan\Dataset\0\myoutput.raw -t vector
> .\ListScan\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\ListScan\Dataset\0\`

Command-line Execution

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

```
.\Debug\ListScan_Template -e .\ListScan\Dataset\0\output.raw
-i .\ListScan\Dataset\0\input.raw
-o .\ListScan\Dataset\0\myoutput.raw -t vector
> .\ListScan\Dataset\0\result.txt
```

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

Questions

(1) Name 3 applications of parallel scan.

(2) How many iterations are being performed in the reduction step of your scan kernel in terms of the input length N? Explain.

(3) How many floating operations are being performed in the reduction step of your scan kernel in terms of the input length N? Explain.

(4) How many iterations are being performed in the post-reduction reverse step of your scan kernel in terms of the input length N? Explain.

(5) How many floating operations are being performed in the reduction step of your scan kernel in terms of the input length N? Explain.

(6) How many global memory reads are being performed by your kernel in terms of input length N? Explain.

(7) How many global memory writes are being performed by your kernel in terms of input length N? Explain.

(8) Describe what optimizations does the work-efficient scan kernel perform to achieve a performance speedup over that of the work-inefficient scan kernel.

Problem#2: Histogram

Objective

The purpose of this problem is to implement an efficient histogramming algorithm for an input array of integers within a given range. You should implement the **histogram kernel that uses privatization technique** as described in the class notes**.** Each integer will map into a single bin, so the values will range from 0 to (NUM_BINS - 1). The histogram bins will use unsigned 32-bit counters that must be saturated at 127 (i.e., if the bin value is more than 127, make it equal to 127). The input length can be assumed to be at most 500000 elements. NUM_BINS is fixed at 4096.

The device computation should be split into two kernels: the first kernel should compute the histogram results without applying saturation and the second kernel should apply saturation to the histogram results that are produced by the first kernel. Note that these two stages could

have been combined into a single kernel. However, in this assignment, we will keep them separated.

Coding

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

- allocate device memory
- copy input host memory to device
- clear the bins on device using cudaMemset() (see the sample use of cudaMemset() in template.cu of ListScan)
- initialize thread block and kernel grid dimensions
- invoke CUDA kernels
- copy output device memory to host
- deallocate device memory
- write the CUDA kernels

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 Histogram_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\Histogram\Dataset\0\output.raw
-i .\Histogram\Dataset\0\input.raw
-o .\Histogram\Dataset\0\myoutput.raw -t integral_vector
> .\Histogram\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\Histogram\Dataset\0\
```

Command-line Execution

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

```
.\Debug\Histogram_Template
-e .\Histogram\Dataset\0\output.raw
-i .\Histogram\Dataset\0\input.raw
-o .\Histogram\Dataset\0\myoutput.raw -t integral_vector
> .\Histogram\Dataset\0\result.txt
```

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

Questions

(1) Describe how does the privatization technique improve the performance of the histogram kernel.

(2) How many global memory reads are being performed by your histogram kernel in terms of the input length N? Explain.

(3) How many global memory writes are being performed by your histogram kernel in terms of the grid size (number of blocks launched) G and NUM_BINS? Explain.

(4) How many atomic operations are being performed by your histogram kernel in terms of the input length N, the grid size G, and NUM_BINS? Explain.

(5) For the histogram kernel, what contentions would you expect if every element in the array has the same value?

(6) For the histogram kernel, what contentions would you expect if every element in the input array has a random value?