GPU Programming Basics

Northeastern University *for all* Julian Gutierrez
NUCAR Laboratory Nicolas Agostini, David Kaeli

Final Project

Histogram Equalization

Objective

1. Understand how Histogram Equalization is applied to images.
2. Write an optimized GPU code in CUDA that provides the same functionality of the histogram equalization from OpenCV but can perform the algorithm faster.

The project can be developed in groups of a maximum of 3 students.

Part 1: Setting up and running the baseline code

The code for the project can be found in the folder on the discovery cluster:

```
/scratch/bohmagostini.n/GPUClassS20/FINPROJ/heq
```

First of all, you need to open 2 terminals and connect them to discovery cluster. Please enable X forwarding (**`ssh -X`** `<username>@login.discovery.neu.edu`) so that you can execute commands that display graphics).

- Terminal 1: this terminal will be used to connected to an interactive compilation node.
- Terminal 2: this terminal will be used to schedule the gpu jobs, it will always be connected to the login node.

In Terminal 1, to request an interactive node for code development and compilation, we can use the following command. Once a resource is available, you will be connected automatically into the compute node. If you are having trouble with this, go back to Lab 2 and review part 1. **NOTE: No partition nor reservation is used (we are using a general node for compiling).**

```
# TYPE this command, do not copy and paste

srun --pty --export=ALL --partition=short --tasks-per-node 1 --nodes 1
--mem=4Gb --time=02:00:00 /bin/bash # Reserve the compilation node
```

Copy the following folder to your scratch directory (or folder of your choosing):

```
mkdir -p /scratch/$USER/GPUClassS20
cd /scratch/$USER/GPUClassS20
cp -r /scratch/bohmagostini.n/GPUClassS20/FINPROJ/ .
cd FINPROJ/heq/
```

In the new folder, using terminal 1, you will find a Makefile which will be used to compile the code by running:

```
make all
```

To clean files that are generated, we have to use the command:

```
make clean
```

Go over all the code to understand how every part works. Please observe what the kernel of the GPU is doing in this baseline code. It's your job to make it work correctly.

To be able to execute you need to load the module to support OpenCV by running this command or adding it to your .bashrc file. This command has been added to the bash script so there is no need for you to load them when executing the code using the gpu nodes:

```
module load opencv/3.4.3-contrib
```

Remember, you should also have the following line in your ~/.bashrc file:

```
module load cuda/9.2
```

You can execute the command using the following notation (though remember we are using sbatch):

```
./heq <input image>
```

If many of you are using this node at the same time, timing results may vary, so if that's the case, feel free to use the gpu partition without specifying the reservation.

For example, to launch the code, in terminal 2:

```
sbatch exec.bash
```

Which will run the following command:

```
./heq input/bridge.png
```

This command will output the following to the sbatch file output:

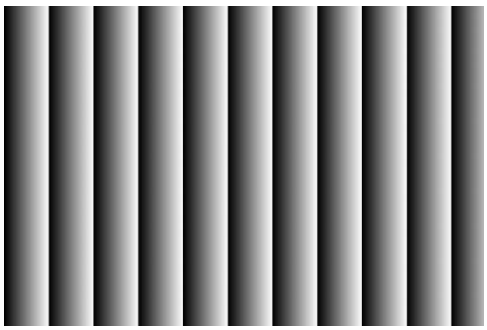
```
Kernel Execution Time: 0.178080 ms
CPU execution time: 17.4714 ms
GPU execution time: 10.5595 ms
Percentage difference: 81.1443%
```

This command will also output the following images:

Input.jpg (Input Black and White version)

Output_cpu.jpg (Output from the OpenCV function)

Output_gpu.jpg (Output from the GPU function)



Given the application will output multiple files, the easiest way to visualize them is by using FileZilla to copy them to your computer (particularly when looking at images). Use the same configuration we saw in the first lab for the application.

Part 2: Writing your GPU Program

To better understand how to implement a histogram equalization, please review the following link:

http://www.programming-techniques.com/2013/01/histogram-equalization-using-c-image.html

The following are a couple of suggestions in the development of the code:

1. When you are developing your code, the first thing you should do is write a code that is doing the correct calculation, and once it's working correctly, you try to optimize it.
2. Divide the algorithm into stages, and possibly run different kernels for each stage.
3. Remember that we should always consider the time it takes to copy the data to the GPU. If it's taking really long, how can you improve it? Running NVPROF could provide a good insight for this.
4. Test code with different sized images. What's the impact in the performance? What about an all-black image?
5. The goal is to have a fast implementation of the algorithm that has over 98% accuracy when compared to the OpenCV implementation.
6. Start working on it as soon as possible.
7. **NOTE: You are only allowed to modify the heq.cu file. Don't modify the main file. This might limit some of your possible optimizations.**

Part 3: Presentation and Competition

To share with your fellow classmates what you did, your team will have to present on the last day of class. No more than 2 slides will be needed explaining your approach for implementing this algorithm and any advanced tricks you used to improve the performance and mention some of the misfortunes you went through when working on it.

In the meantime, the professor will be coordinating the execution of each code to compare the performance. The results obtained from running the script that automatically recollects the results will be final, even if a minor tweak could potentially make it run faster or if your personal tests achieved better performance. Take into consideration that the code will be launched using a big image.