Optimization of an Image Processing Algorithm: Histogram Equalization

Julian Gutierrez
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
jgutierrez@ece.neu.edu

Fritz Previlon
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
previlon.f@husky.neu.edu

David Kaeli
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
kaeli@ece.neu.edu

Many textbooks rely on classical linear algebra examples to illustrate best practices in parallel programming (e.g., matrix multiplication and vector add). Despite their common use in class, these examples lack sophistication of a complete application. We have found that students seem to be more motivated to work with imaging processing algorithms, where the student can view the before and after image, visually inspecting the results of their processing.

This assignment focuses on improving the performance of the histogram equalization algorithm applied to an image. Histogram equalization is a popular image processing algorithm used to increase the contrast of an image to better highlight its features. It is a common algorithm used in many scientific applications such as x-ray imaging, thermal imaging and as a pre-processing task for multiple computer vision/deep learning algorithms.

The following guidelines are used for this assignment:

- All students work on the same project.
- The project can be developed in groups of (at most) 3 students.
- The students need to study the application to understand how the algorithm works.
- They develop an optimized GPU implementation using CUDA, providing the same functionality (but faster) of the histogram equalization found in the OpenCV package.
- Students are encouraged to explore their own unique algorithm if the output result is comparable to that of the OpenCV library (+- 5% difference allowed).
- Students obtain all the points if they have successfully completed the assignment.
- Extra points (or rewards) are given to those who achieved the best performing implementation.

Students are given a baseline code that works with a CPU-based OpenCV interface, and a simple CUDA kernel which reads the input image and provides the necessary structure to modify the image. This structure allows the students to focus on the algorithm implementation and performance, without having to develop the initial program structure. Additionally, evaluating the final code from students becomes easier if they all follow the same coding structure. A key factor in the effectiveness of this assignment is dedicating a class session discussing their implementations, sharing the different ways of optimizing their resulting CUDA code with their classmates.

Figure 1. Example input image and output for the histogram equalization algorithm.

Concepts covered within this assignment include: 1) CUDA programming in C, 2) performance tuning with profiling tools (nvprof and nvvp), and 3) algorithmic optimizations specific for image processing algorithms (including memory performance improvement through coalescing reads and shared memory, and efficient reduction schemes for histograms).

The assignment is appropriate for students at all academic levels, as long as they have a passing knowledge of CUDA as part of their past coursework (with most CUDA and GPU architecture concepts covered before the assignment). This assignment has been used as a final project for a free GPU programming class offer to undergraduates and graduate students at Northeastern for the past 5 years. A GPU was awared to the best performing project. Additionally, multiple image processing algorithms can be used as variations for this assignment, such as the Sobel filter.

The strengths of this assignment include:

- Motivational (we received positive feedback from students).
- Challenging, yet doable.
- Encompasses multiple GPU programming optimization concepts.

The weaknesses of this assignment include:

- Oriented specifically toward the CUDA programming language (although the assignment can be adapted to other parallel programming languages).
- Requires CUDA-enabled hardware.