PC-2020/21 Histogram Equalization

Lorenzo Arena
lorenzo.arena@stud.unifi.it

Abstract

This project was made as the final work for the "Parallel Programming" exam at the University of Florence. The objective was to create three version of a software to equalize the histogram of a given JPEG image. The first version had to be implemented as a sequential program, while the other two had to be parallel; for the parallel versions one has been implemented using OpenMP on top of the already implemented sequential code and the other is built using CUDA to take advantage of a GPU computing power. The projects has been developed in C on a Ubuntu machine; all tests were made on a Ryzen 7 1700 CPU and a NVIDIA Quadro P2000.

1. The algorithm

Histogram equalization in image processing can be used to increase an image's contrast. This is accomplished by spreading the most frequent intensities values across the whole histogram. It works especially well on images which presents foreground and background which are both dark or both bright.

1.1. implementation

We can start by considering a grayscale image of n pixels, in which n_i is the number of occurrences of gray level i. The probability of an occurrance of a pixel of level i (with $0 < i < L$) in the image is:

$$p_x(i) = \frac{n_i}{n}$$

The histogram is the distribution of the pixel levels in the range $[0, L-1]$.

We can define the *cumulative distribution function* as the cumulative sum of all the probabilities lying in its domain:

$$cdf_x(i) = \sum_{j=0}^{i} p_x(x = j)$$

To get a linearized cdf, which will produce a flatten histogram, we must normalize the is to the $[0, L-1]$ range by using the following formula:

$$cdfn_x(i) = \frac{cdf(i) - cdf_{min}}{(M * N) - cdf_{min}}$$

where M and N are the image's dimensions. The normalized cdf must then be applied to the original histgram:

$$h(v) = round(cdfn_x(i) * (L-1))$$

While this can be applied to grayscale images by using the pixel value, for color images applying the equalization process to the R, G and B channels would break the image's color balance since the relative distribution of the color channels would be changed. Thus, the pixels must be converted to another color space (like HSL) where the algorithm can be applied to the *luminance* channel without creating changes in the *hue* or *saturation* channels.

2. The sequential implementation

3. The OpenMP implementation

4. The CUDA implementation

5. Results