Technical university „Gheorghe Asachi” Iasi
Faculty of Automatic Control and Computer Engineering
Distributed systems and web technologies

Histogram equalization

Students:

Dobreanu Mircea-Constantin

Chelaru Ștefan

Problem statement

Our assignment is to equalize the histogram of an input grayscale image, which has the effect of improving the contrast of the image. So the black regions of the original image become even more black and the white regions become even whiter.

CPU implementation

The basic algorithm is already implemented in an efficient way by the OpenCV library, and the basic flow of the algorithm is:

1. Compute histogram of the input image.
2. Compute the cumulative distribution function from the input image.
3. For each pixel of the original image assign a new intensity value based on its value in the original image and the cumulative distribution function.

GPU implementation

The same steps apply as above but we have to account for the parallel nature of computing on GPUs.

1. Compute histogram of the input image by counting the number of pixels of a specific intensity. This is done with the help of the atomic add operation provided by CUDA. Probably the biggest factor in execution time.
2. Compute the cumulative distribution function (prefix sums/inclusive scan) based on the Hillis-Steele algorithm. Basically there are log2(n) stages in which the values are propagated to cells that are at an offset distance where offset is a power of 2. The implementation is not necessarily efficient but since there are only 256 possible gray levels, the potential for performance improvements here is not that great.
3. Based on the pixel's value in the original image and the cumulative distribution, compute the new value in the improved image. This operation is inherently parallel.

Used images:

Image 1

512x384

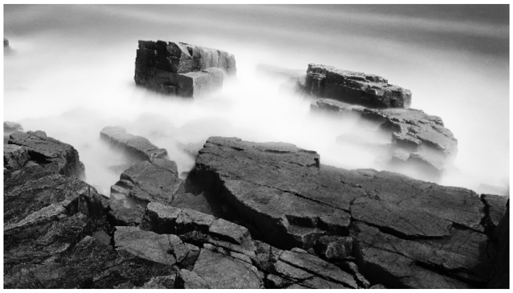
Image 2

910x682

Image 3

1000x667

Image 4



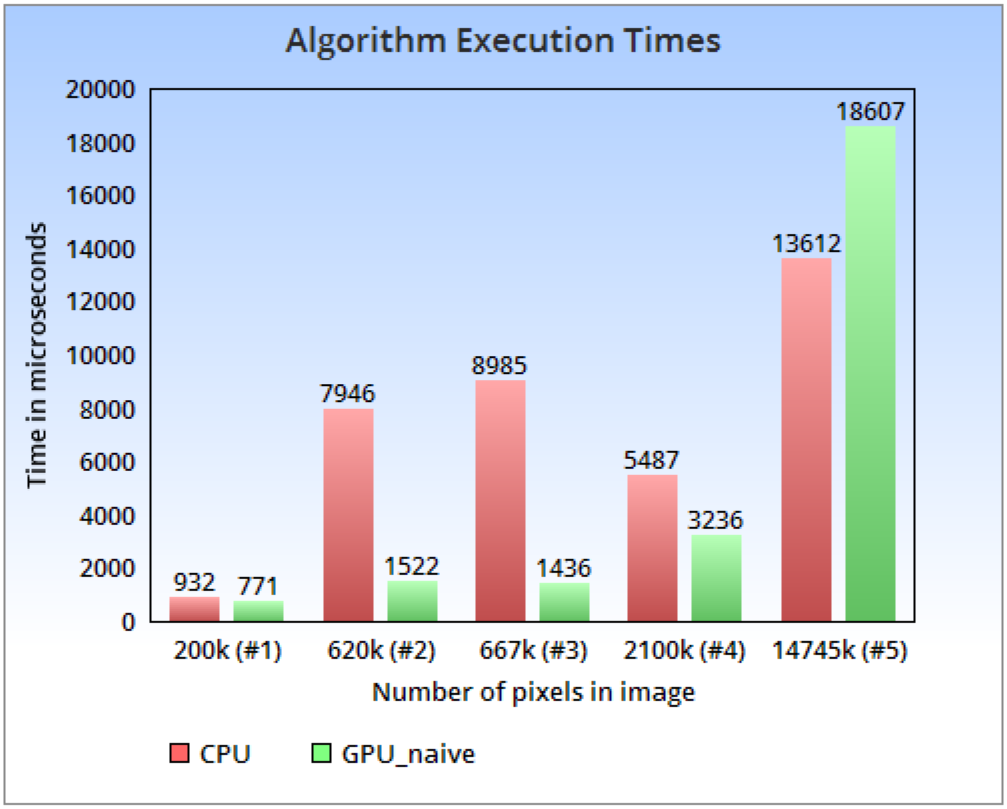
1920x1080

Image 5

5120x2880

Image number	CPU total time (µs)	Time to transfer image from CPU to GPU (µs)	The time to equalize the histogram on the GPU for the input image (µs)	The time to transfer the equalized image back to CPU (µs)	GPU total time (µs)
1	932	112	563	96	771
2	7946	223	996	303	1522
3	8985	216	1011	208	1436
4	5487	563	2196	476	3236
5	13612	3299	11958	3349	18607

*Time is measured in microseconds



Device specifications for measurements:

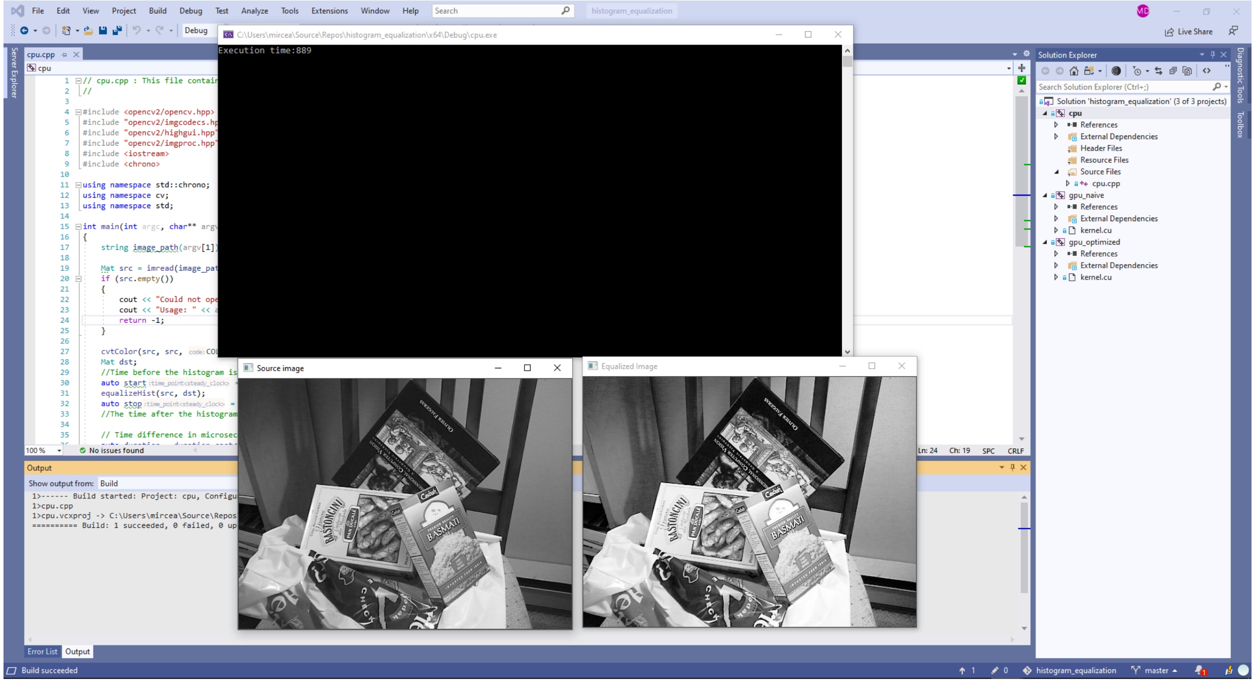
CPU: i7-9750H CPU @ 2.60GHz

16Gb RAM

GPU: Nvidia GTX 1650 (CUDA capability version 6.1, CUDA driver version 11.1)

Visual Studio 2019

Example runs



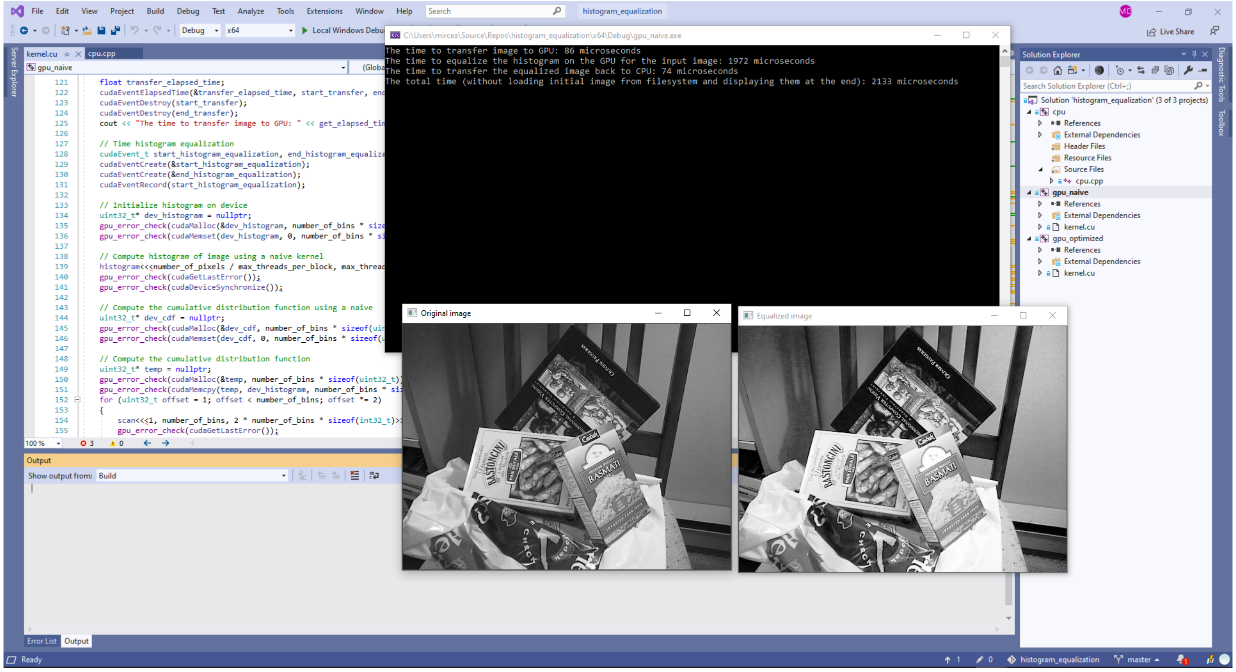
```
// cpu.cpp : This file contain
//

#include <opencv2/opencv.hpp>
#include "opencv2/imgcodecs.hp
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <iostream>
#include <chrono>

using namespace std::chrono;
using namespace cv;
using namespace std;

int main(int argc, char** argv
{
    string image_path(argv[1])

    Mat src = imread(image_pat
    if (src.empty())
    {
        cout << "Could not ope
        cout << "Usage: " << a
        return -1;
    }

    cvtColor(src, src, code: COL
    Mat dst;
    //Time before the histogram is
    auto start :time_point<steady_clock> =
    equalizeHist(src, dst);
    auto stop :time_point<steady_clock> =
    //The time after the histogram

    // Time difference in microsec
```

```
Execution time:889
```

```
1>------ Build started: Project: cpu, Configu
1>cpu.cpp
1>cpu.vcxproj -> C:\Users\mircea\Source\Repos
========== Build: 1 succeeded, 0 failed, 0 up
```

C:\Users\mircea\Source\Repos\histogram_equalization\x64\Debug\cpu.exe
Source image
Equalized Image
Solution Explorer
Solution 'histogram_equalization' (3 of 3 projects)
Build succeeded

Example CPU run



```
    float transfer_elapsed_time;
    cudaEventElapsedTime(&transfer_elapsed_time, start_transfer, end
    cudaEventDestroy(start_transfer);
    cudaEventDestroy(end_transfer);
    cout << "The time to transfer image to GPU: " << get_elapsed_tim

    // Time histogram equalization
    cudaEvent_t start_histogram_equalization, end_histogram_equaliza
    cudaEventCreate(&start_histogram_equalization);
    cudaEventCreate(&end_histogram_equalization);
    cudaEventRecord(start_histogram_equalization);

    // Initialize histogram on device
    uint32_t* dev_histogram = nullptr;
    gpu_error_check(cudaMalloc(&dev_histogram, number_of_bins * size
    gpu_error_check(cudaMemset(dev_histogram, 0, number_of_bins * si

    // Compute histogram of image using a naive kernel
    histogram<<<number_of_pixels / max_threads_per_block, max_thread
    gpu_error_check(cudaGetLastError());
    gpu_error_check(cudaDeviceSynchronize());

    // Compute the cumulative distribution function using a naive
    uint32_t* dev_cdf = nullptr;
    gpu_error_check(cudaMalloc(&dev_cdf, number_of_bins * sizeof(uin
    gpu_error_check(cudaMemset(dev_cdf, 0, number_of_bins * sizeof(u

    // Compute the cumulative distribution function
    uint32_t* temp = nullptr;
    gpu_error_check(cudaMalloc(&temp, number_of_bins * sizeof(uint32_t))
    gpu_error_check(cudaMemcpy(temp, dev_histogram, number_of_bins * si
    for (uint32_t offset = 1; offset < number_of_bins; offset *= 2)
    {
        scan<<<1, number_of_bins, 2 * number_of_bins * sizeof(int32_t)>>
        gpu_error_check(cudaGetLastError());
```

```
The time to transfer image to GPU: 86 microseconds
The time to equalize the histogram on the GPU for the input image: 1972 microseconds
The time to transfer the equalized image back to CPU: 74 microseconds
The total time (without loading initial image from filesystem and displaying them at the end): 2133 microseconds
```

C:\Users\mircea\Source\Repos\histogram_equalization\x64\Debug\gpu_naive.exe
Original image
Equalized image
Solution Explorer
Solution 'histogram_equalization' (3 of 3 projects)
Ready

Example GPU run

Note: These runs are only for showing the program working over images. These screenshots were not done on the machine used for measuring.