



UNIVERSITÀ
DEGLI STUDI
FIRENZE

RGB Images Histogram Equalization

Francesco Lombardi
Alessandro Minervini

Introduction

Histogram Equalization is an image's histogram based technique used in image processing to improve an image. Given a greyscale one, we have to:

1. Compute its histogram;
2. Apply the following rule to compute equalization:

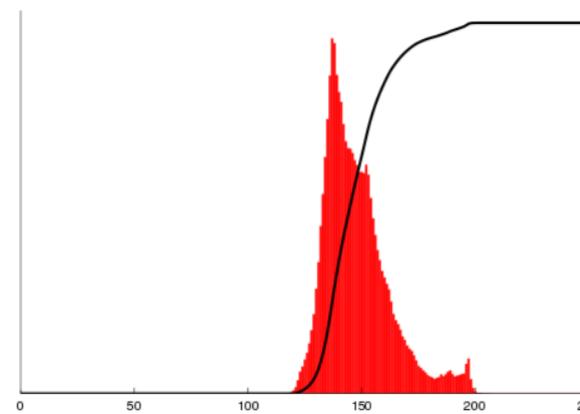
$$h_{eq}(hist) = \text{round}\left(\frac{cdf(hist) - cdf(0)}{(N * M) - 1} * (L - 1)\right)$$

where cdf is the cumulative distribution function, $N * M$ is image size and $L - 1$ is the maximum value for a pixel.

3. Update image histogram with the computed values.



An example of greyscale image histogram equalization:



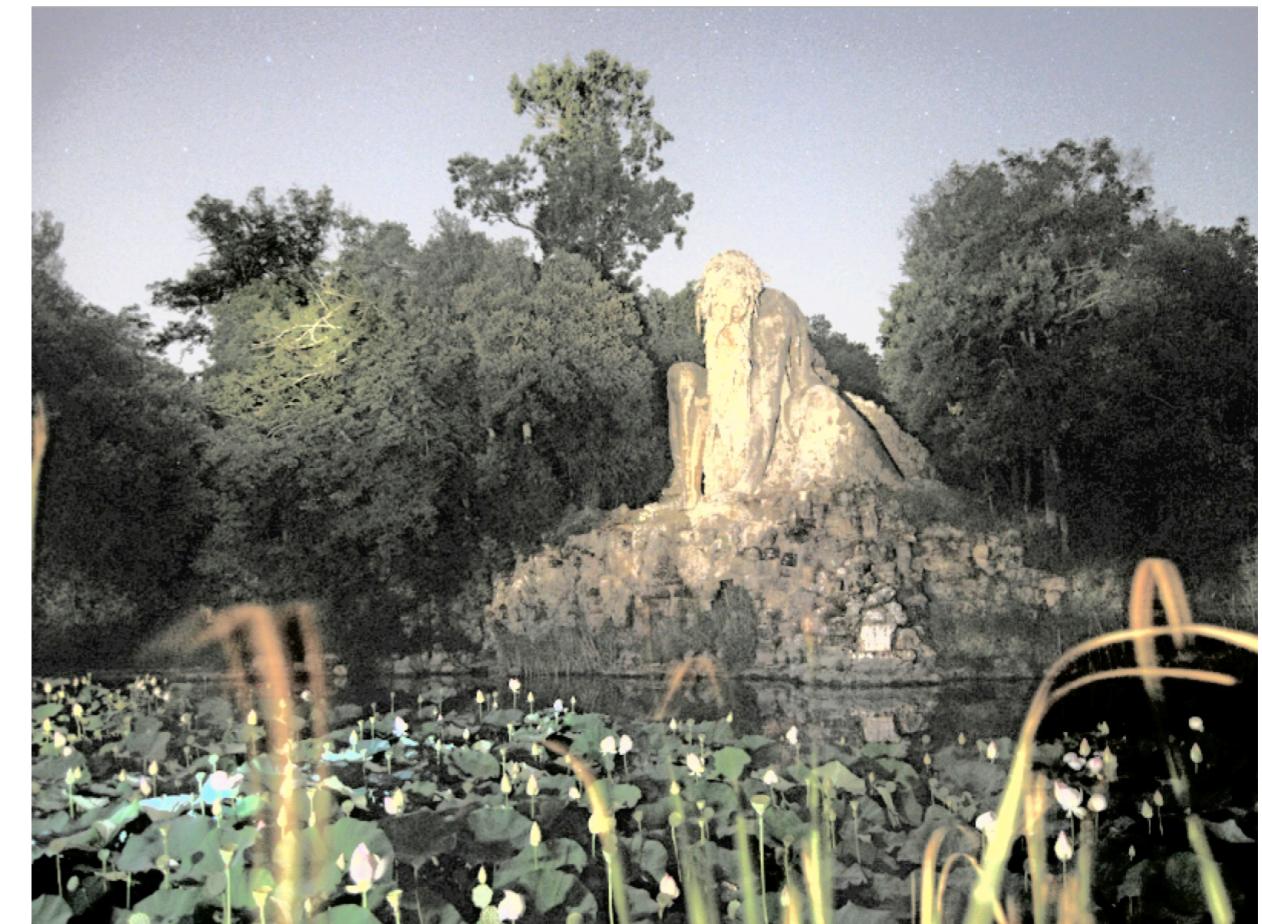


Histogram Equalization can be realized also with RGB images. In order to do this we have followed the steps below:

1. Switch to a different color space having a Y channel (YUV in our case);
2. Compute the histogram of Y channel values;
3. Equalize the histogram and map old values with the new ones;
4. Turn back to RGB color space.



An example of RGB image histogram equalization:



Languages

We have implemented the task using different technologies:

- **C++** for the sequential version;
- **OpenMP** for the first parallel version;
- **CUDA/C++** for the second parallel version.



Implementation

Sequential version is made by 3 main functions:

```
function Make_Histogram
Data: image, histogram, YUV_image;
initialize histogram;
for i = 0, i < image.cols, i ++
    for j = 0, j < image.cols, j ++
        R, G, B = image(i,j);
        Y = R * 0.299 + G * 0.587 + B * 0.114;
        U = R * -0.168736 + G * -0.331264 + B * 0.5
            + 128;
        V = R * 0.5 + G * -0.418688 + B * -0.081312
            + 128;
        histogram[Y]++;
        YUV_image(i,j) = Y, U, V;
    end
end
```

Algorithm 1: Make_histogram

Implementation

function Cumulative_Histogram

Data: histogram, equalized_histogram, image.cols,
 image.rows;
 cumulative_histogram[256];
for $i = 0, i < 256, i++$ **do**
 cumulative_histogram[i] = histogram[i] +
 cumulative_histogram[i-1];
 equalized_histogram[i] = (cumulative_histogram[i]
 - histogram[0])/(cols * rows - 1)*255);
end

Algorithm 2: Cumulative_histogram

function Equalize

Data: image, equalized_histogram, YUV_image;
 initialize histogram;
for $i = 0, i < image.cols, i++$ **do**
for $j = 0, j < image.rows, j++$ **do**
 Y = equalized_histogram[YUV_image(i,j)];
 U = YUV_image(i,j);
 V = YUV_image(i,j);
 R = max(0, min(255,(int)(Y + 1.4075 * (V -
 128))));
 G = max(0, min(255,(int)(Y - 0.3455 * (U -
 128) - (0.7169 * (V - 128)))));
 B = max(0, min(255,(int)(Y + 1.7790 * (U -
 128))));
 image(i,j) = R, G, B;
end
end

Algorithm 3: Equalize

Implementation

OpenMP version has been implemented reusing the sequential code and entrusting the **for loop** computation to an OMP specific structure:

```
#pragma omp parallel default(shared)
{
    #pragma omp for schedule(static)

    for (int i = 0; i < image.rows; i++) {

        for (int j = 0; j < image.cols; j++) {

            /* code here */
        }
    }
}
```

Implementation

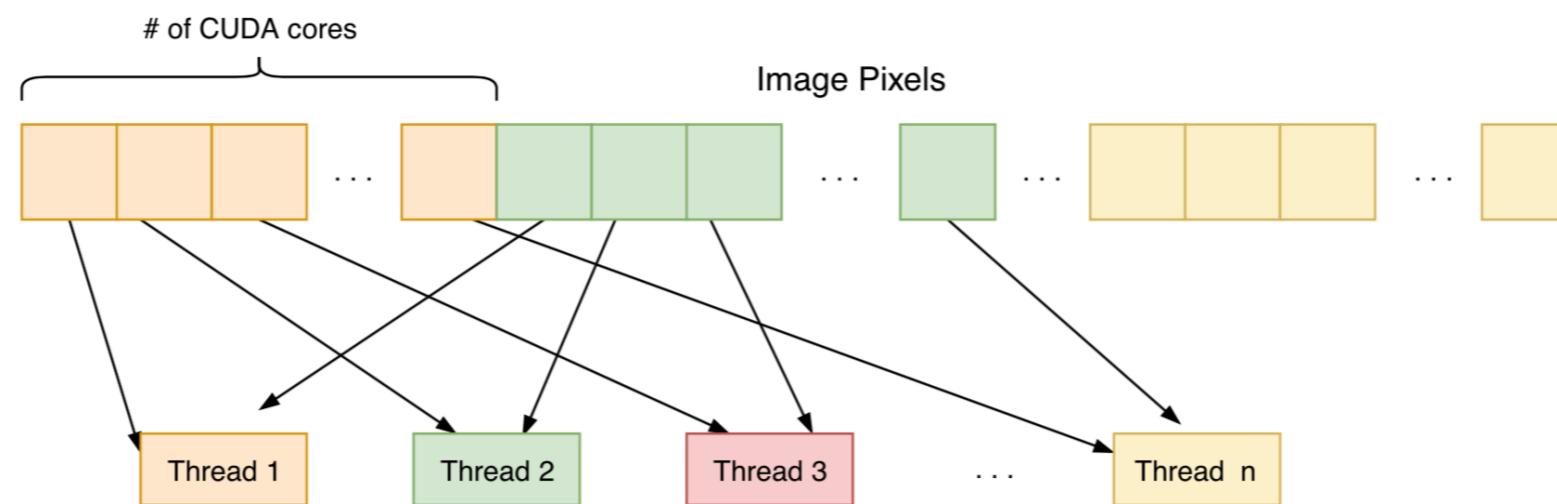
To implement CUDA version some characteristic functions have been used:

- *cudaMalloc()* to allocate arrays in the GPU;
- *cudaMemcpy()* to copy arrays from CPU to GPU and vice versa;
- *cudaFree()* to free GPU.

Moreover CUDA kernels *make_histogram()*, *equalize()* and *YUV2RGB()* have been implemented to execute the functions shown before.

In order to implement CUDA version we have paid special attention on image's pixels management:

- Images have been computed assigning one CUDA core to each pixel (as long as it was possible)
 - Otherwise CUDA cores follow the pattern below:

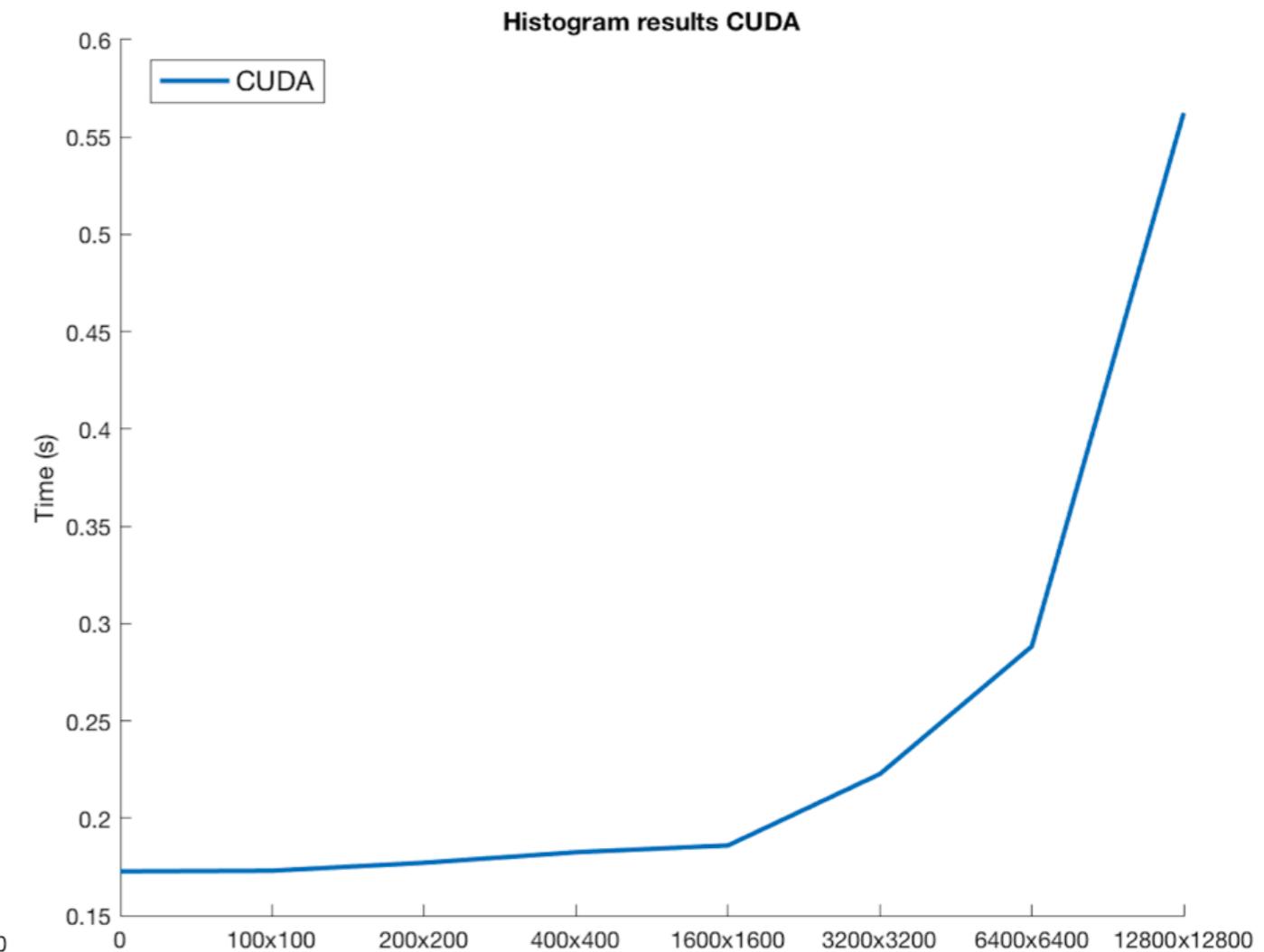
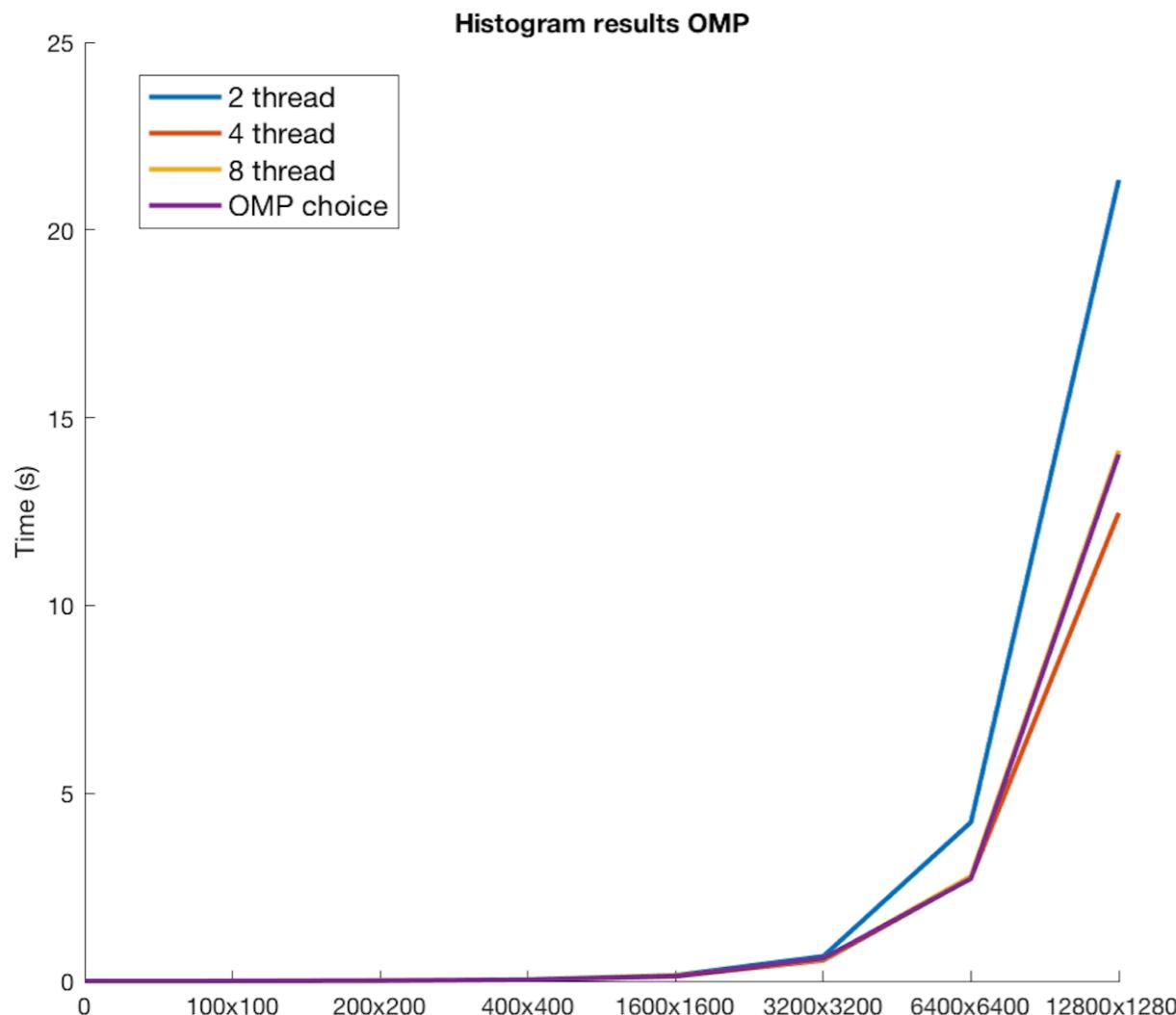


Results

The following results have been obtained:

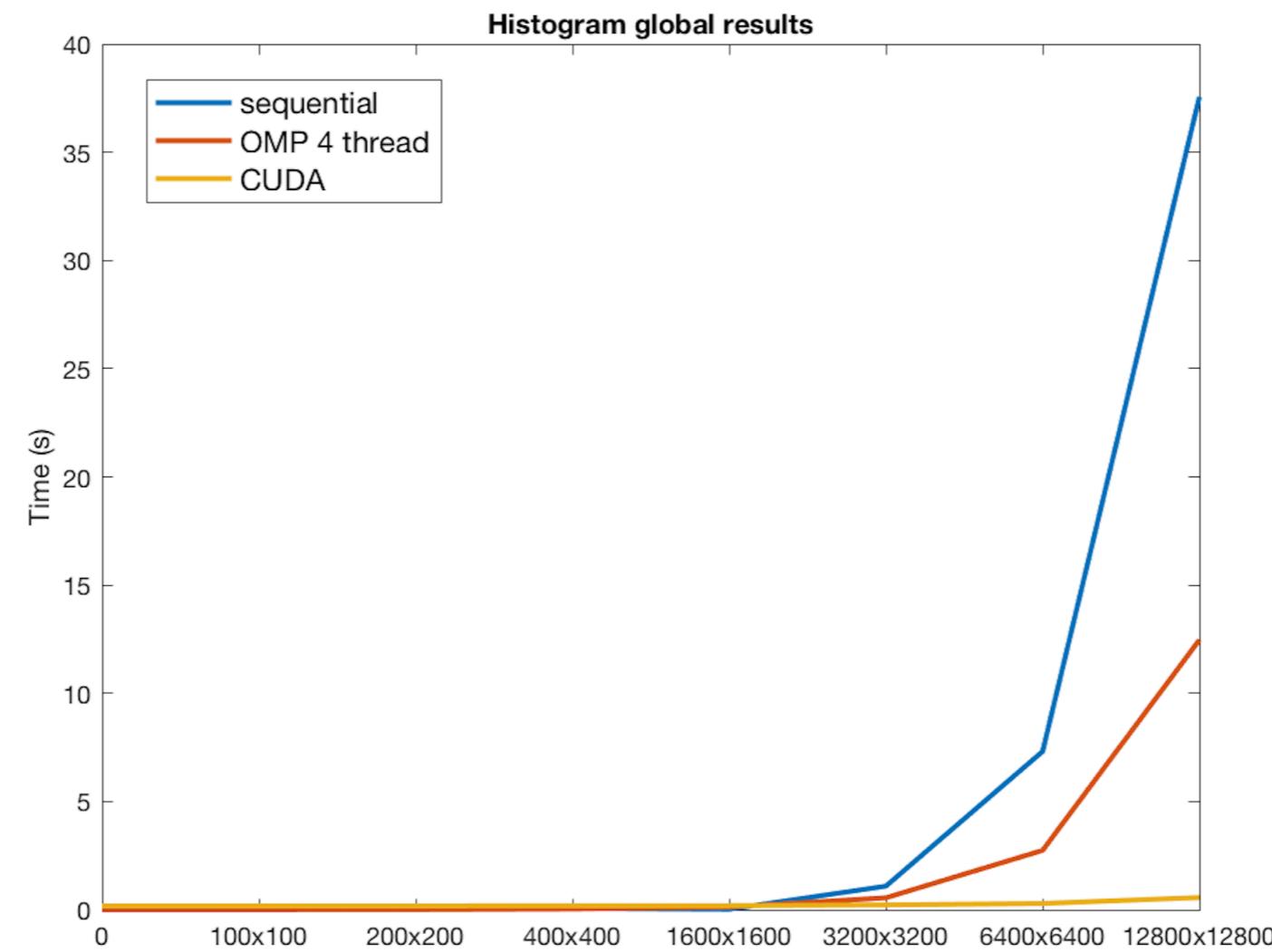
- when varying image dimensions (100x100, 200x200, … , 12800x12800);
- when varying the number of threads (until saturation);
- mediating between 10 runs of the same task.

Results



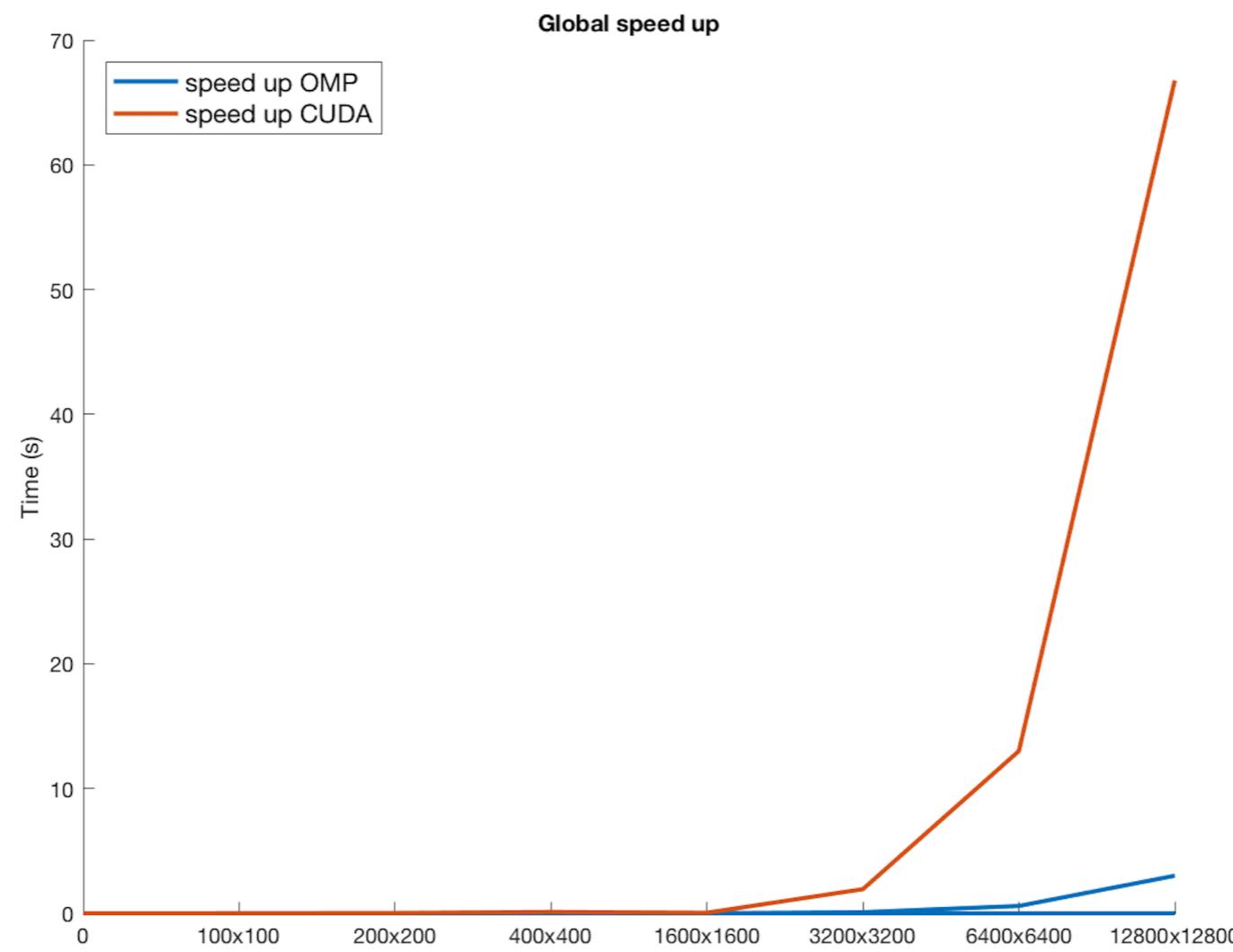
Results

The best configurations of different implementation are compared below:



Speed Up

Relative speed ups:

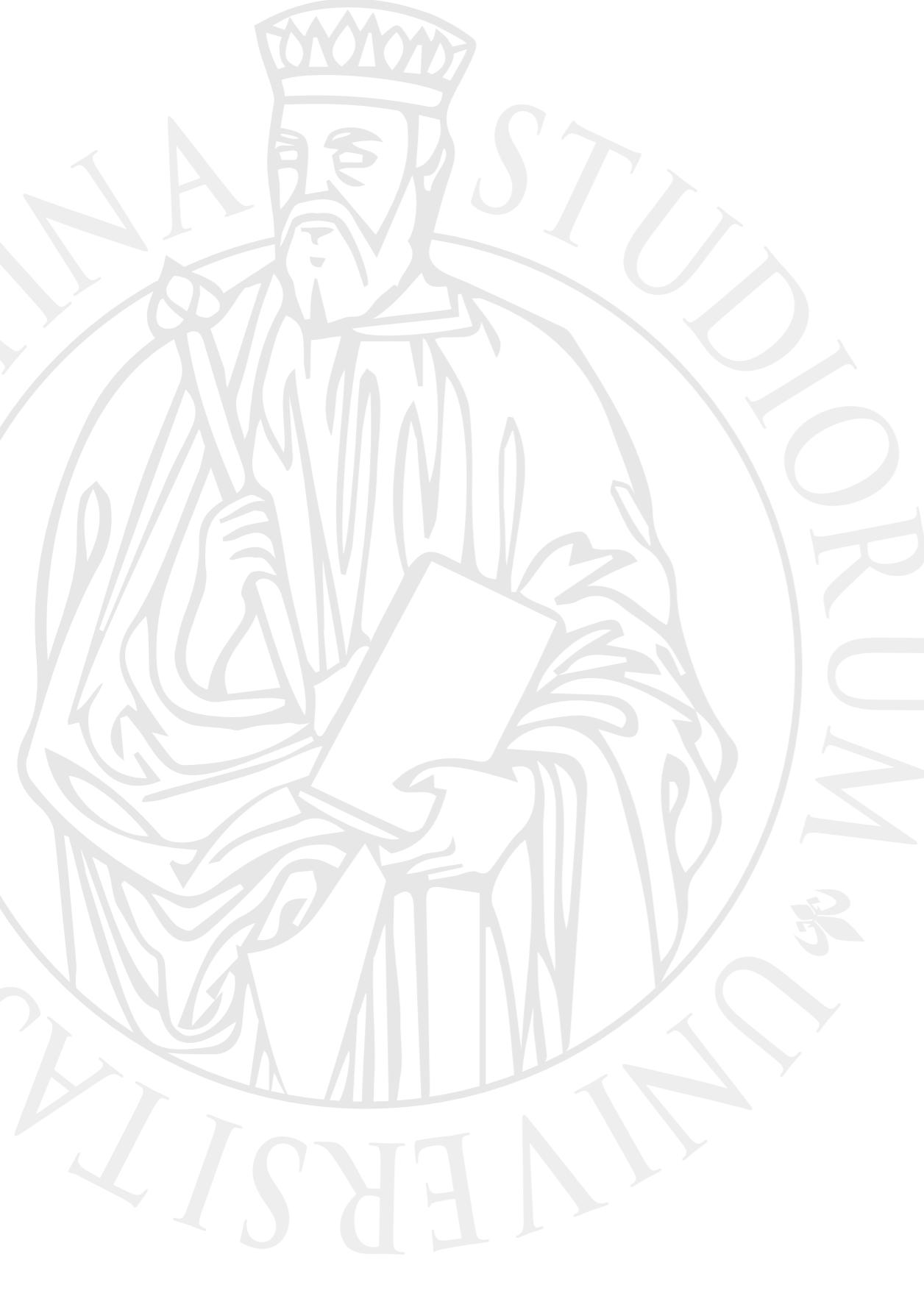


Notes

1. OpenCV library has been used to load, resize and (in CUDA) save images.
2. Two different time evaluators have been used to compute execution time:
 - OpenMP' s *omp_get_wtime()* for sequential and OMP;
 - *gettimeofday()* for CUDA version.



UNIVERSITÀ
DEGLI STUDI
FIRENZE



RGB Images Histogram Equalization

Francesco Lombardi
Alessandro Minervini