UNIVERSITÀ
DEGLI STUDI
FIRENZE

Histogram Equalization

Leonardo Casini
Harjinder Singh Sandhu

Abstract

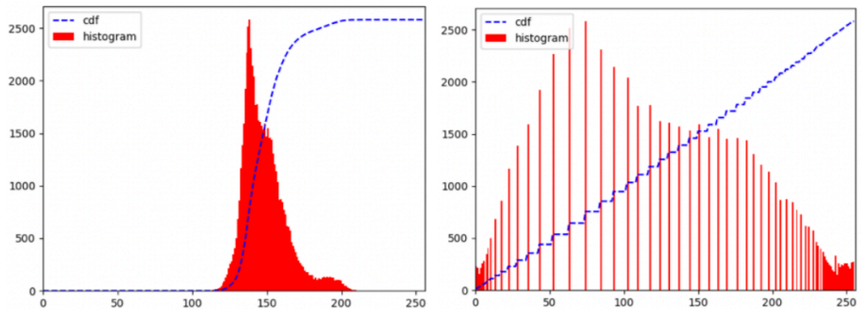
- **Image histogram:** graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. The histogram represents the number of pixels for each intensity value considered.

- **Histogram equalization:** method used in Image Processing through which the contrast can be calibrated using the image histogram. It accomplishes this by effectively spreading out the most frequent intensity values.

- **Increases the global contrast:** especially when the usable data of the image is represented by close contrast values.

- **Invertible method**: if the histogram equalization function is known, then the original histogram can be recovered.

(a) Original Image (b) Equalized Image



(c) Original Histogram (d) Equalized Histogram

Theoretical formulation

- The *probability* of an occurrence of a pixel of level *i* in the image is:

$$p_x(k) = p(x = k) = \frac{n_k}{n} \qquad 0 \leq k < L$$

- *cumulative distribution function:*

$$cdf_x(k) = \sum_{l=0}^{k} p_x(l)$$

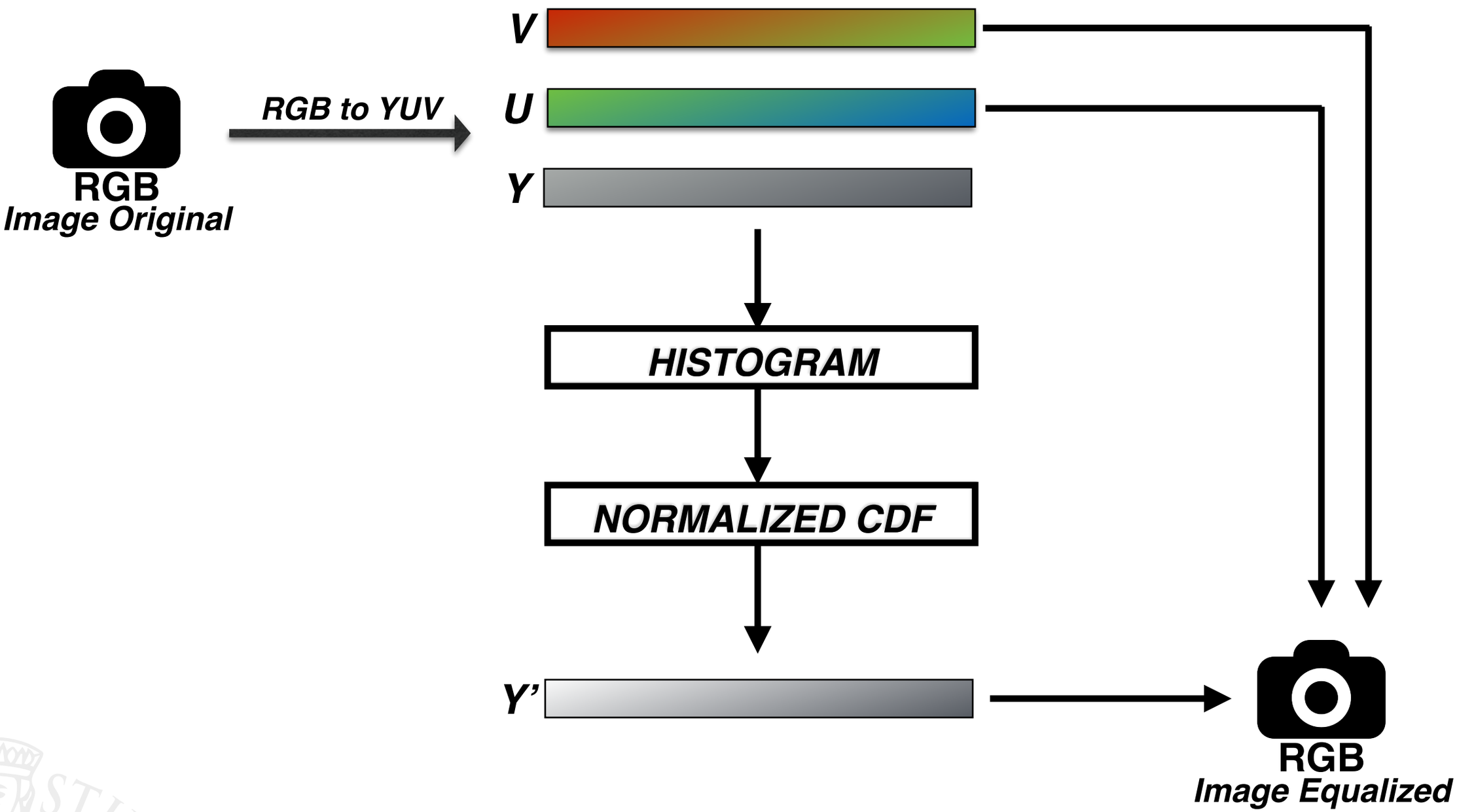
- *normalize* it such that the maximum value is 255 *:*

$$y(i,j) = h(x(i,j) = round\Big(\frac{cdf(x(i,j)) - cdf_{min}}{(M \cdot N) - cdf_{min}} \cdot (L-1)\Big)$$

Technologies

- Sequential Implementation: C++
- Parallel Implementation 1: OpenMP
- Parallel Implementation 2: CUDA

Sequential


RGB
Image Original
RGB to YUV
V
U
Y
HISTOGRAM
NORMALIZED CDF
Y'
RGB
Image Equalized

Parallel: OpenMP

- OpenMP (Open Multi-Processing) is an API that provides C/C++/ Fortran the ability to run certain parts of the code in parallel .
- The implementation is similar to the sequential one. OpenMP consists of a set of compiler *#pragma* that controls how the program works.

```
#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 */
        }
    }
```

Parallel: CUDA

- CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia.

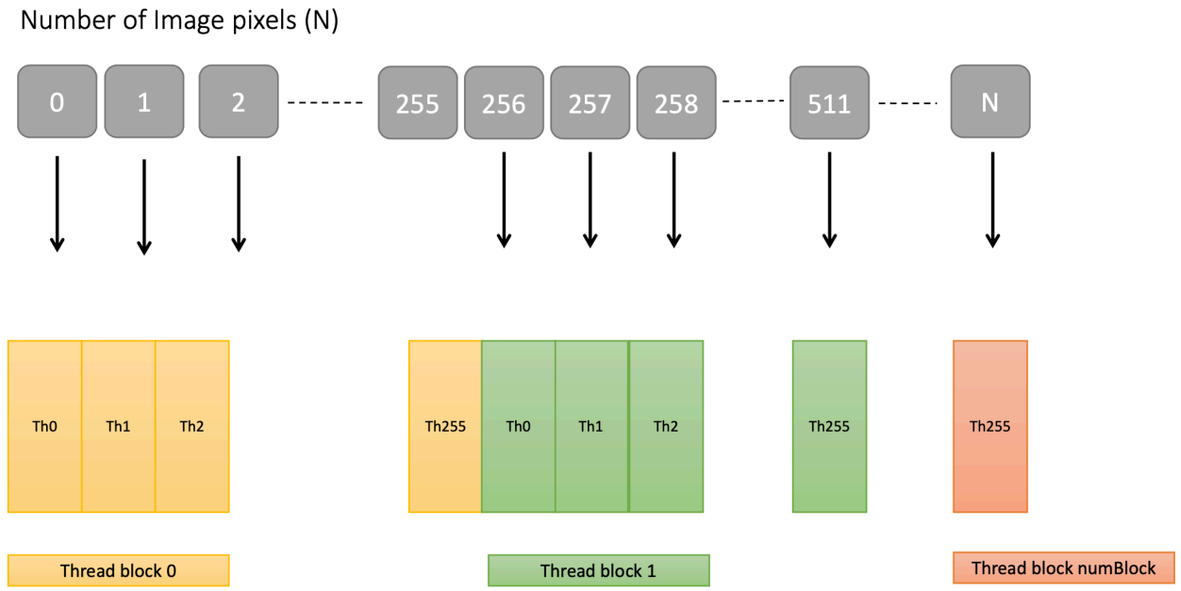
- It allows to use a CUDA-enabled GPU for parallel calculation.

- Functions used while implementing:
  - *cudaMalloc():* to allocate arrays in the GPU
  - *cudaMemcpy():* to copy arrays from CPU to GPU and vice versa
  - *cudaFree():* to free GPU

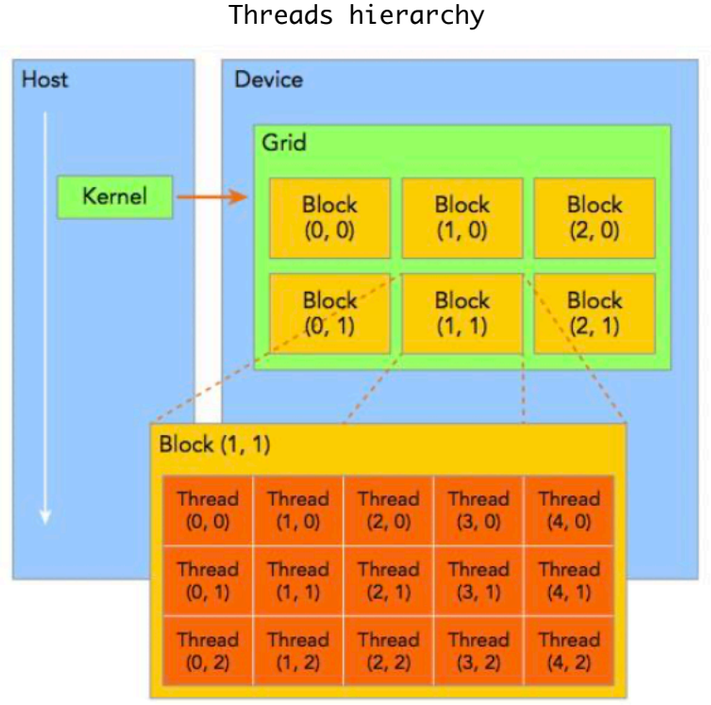
- The functions illustrated in the sequential version have been reproduced as three CUDA kernels: *hist_maker*, *normalizeCdf* and *equalize*.

Parallel CUDA

- The main idea for tasks parallelization has been the following: if possible, depending on the number of CUDA cores, pixels of the image have been processed one per thread.
- Each thread block contains 256 threads.

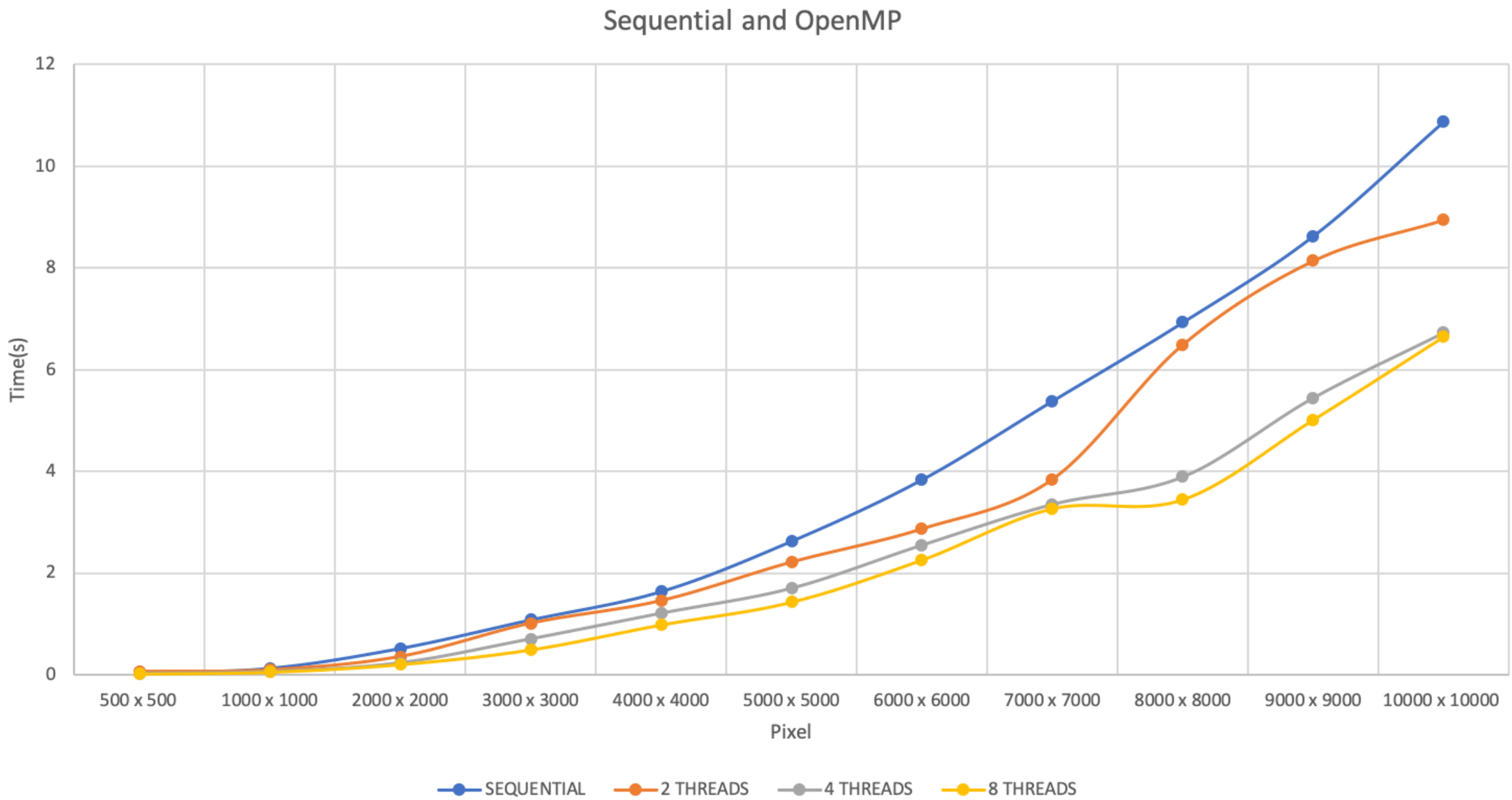


numBlocks = (N + blockSize – 1) / blockSize

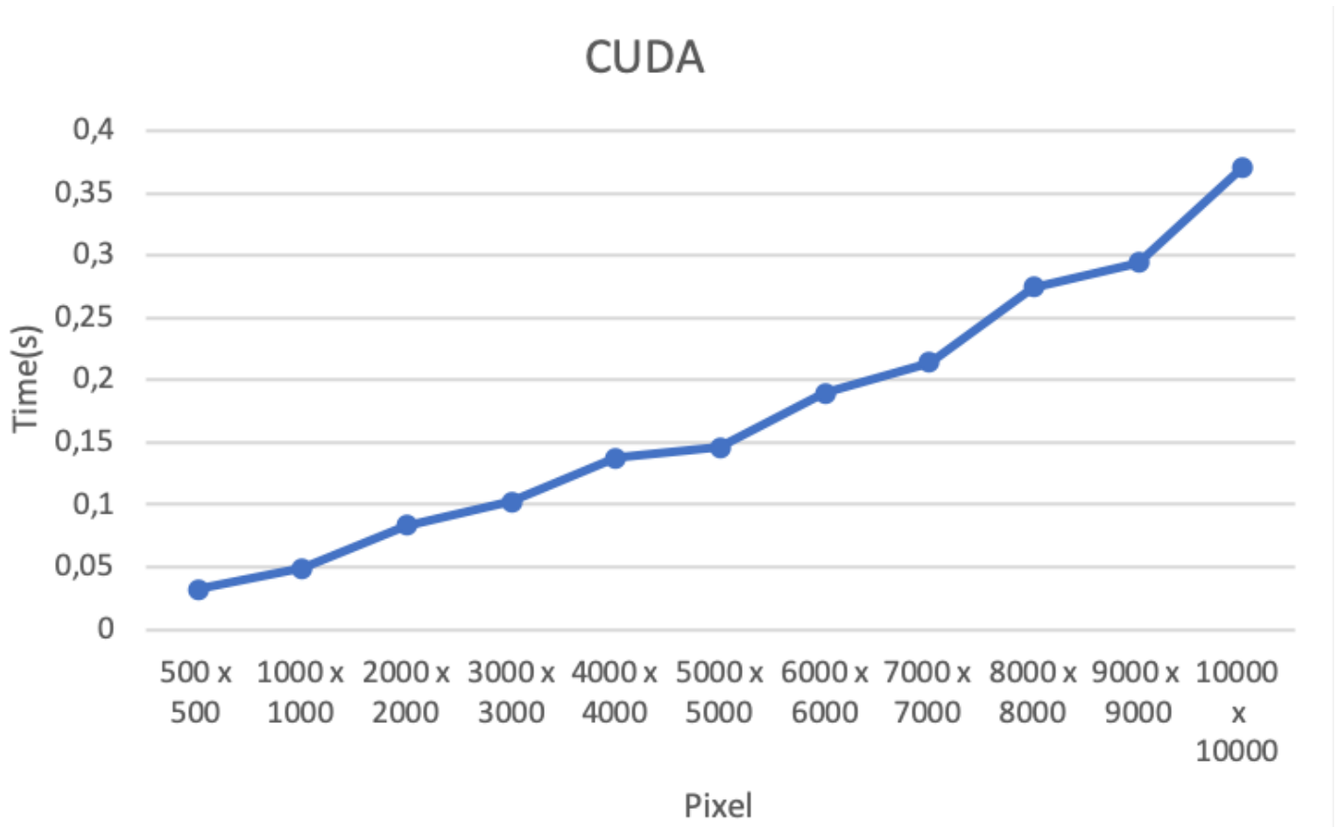



```
-kernel_name <<< grid, block >>> (argument list);
```

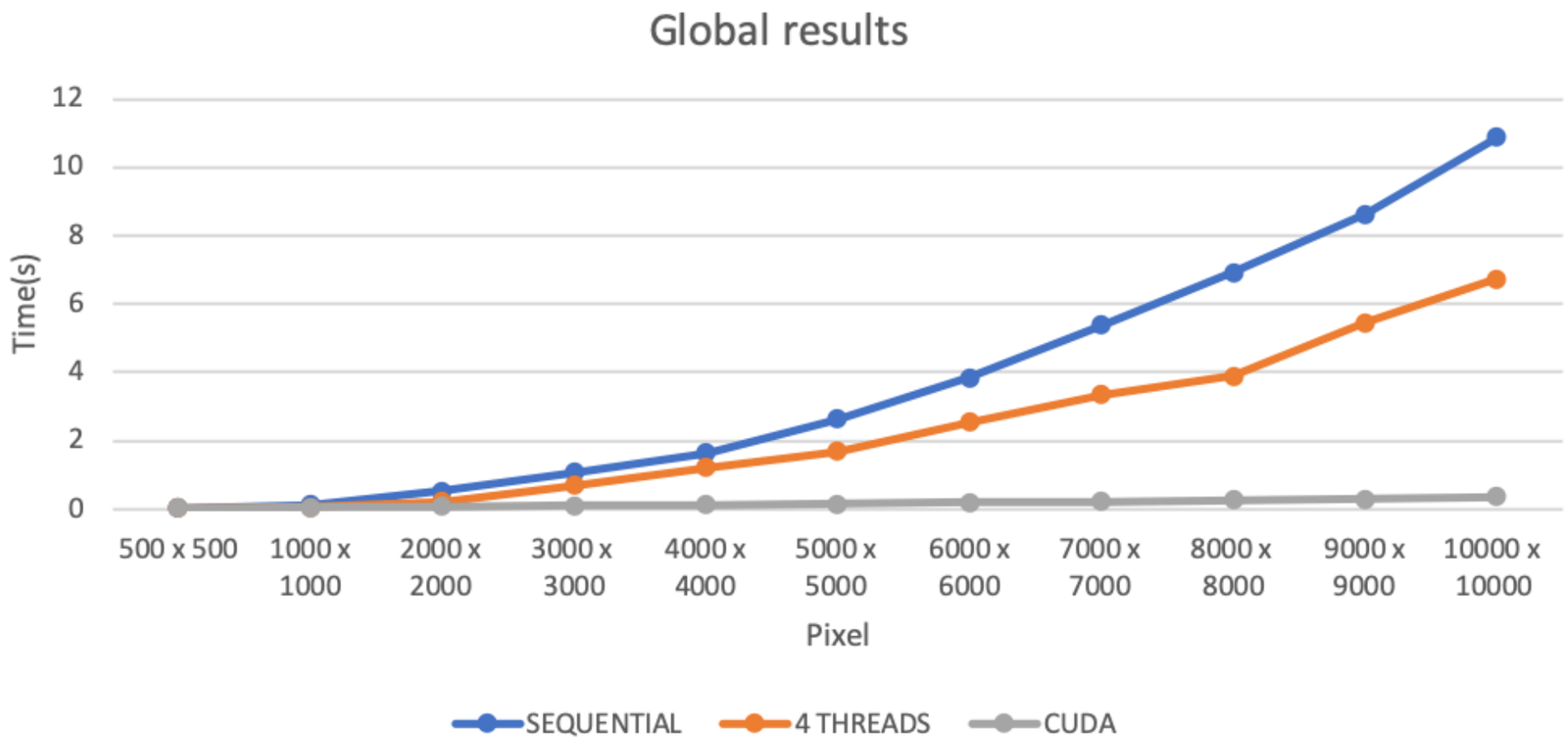
Results


Sequential and OpenMP
12
10
8
6
4
2
0
Time(s)
500 x 500
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
6000 x 6000
7000 x 7000
8000 x 8000
9000 x 9000
10000 x 10000
Pixel
SEQUENTIAL
2 THREADS
4 THREADS
8 THREADS

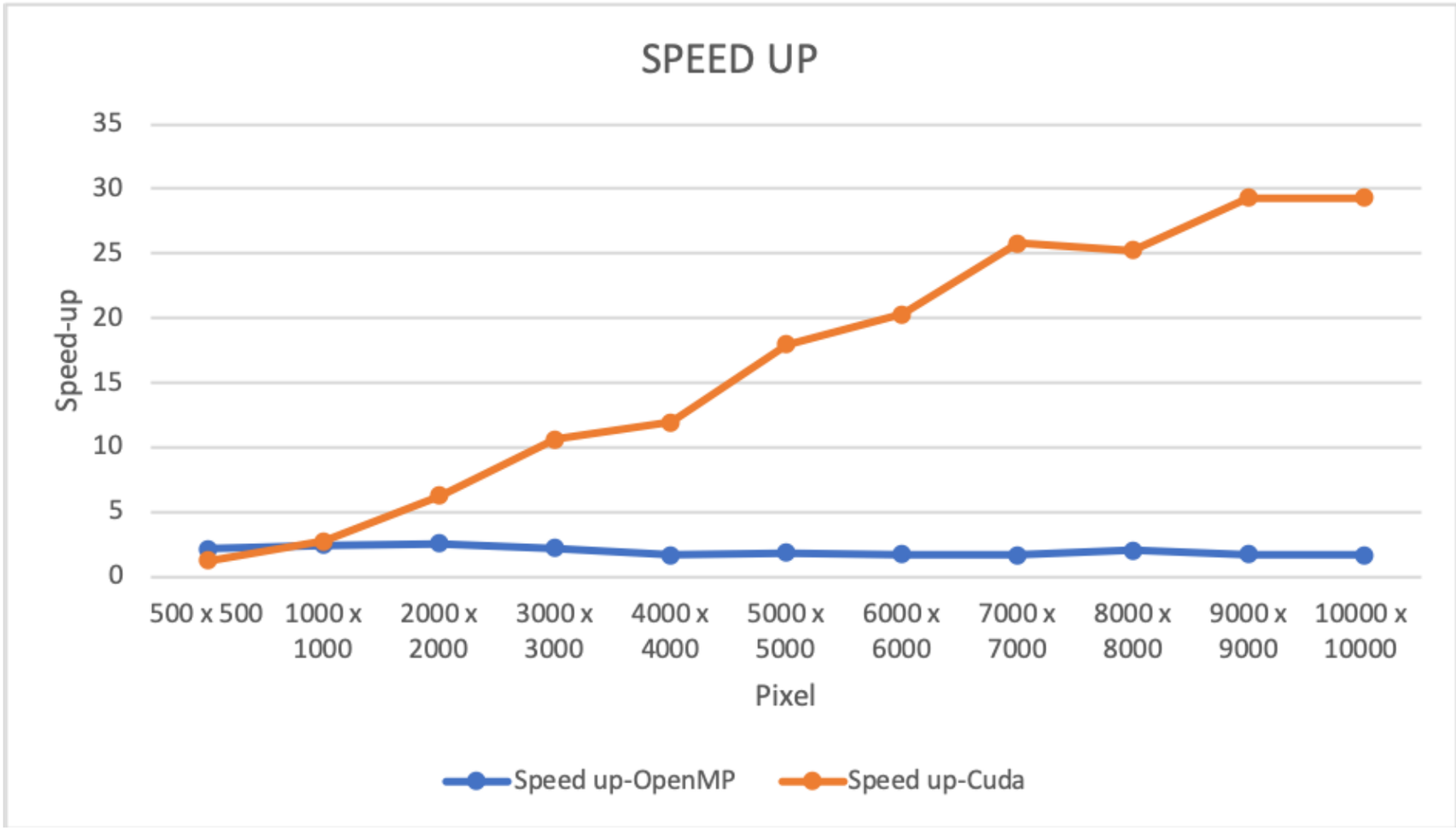
Results


CUDA
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
Time(s)
500 x 500
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
6000 x 6000
7000 x 7000
8000 x 8000
9000 x 9000
10000 x 10000
Pixel

Results


Global results
Time(s)
12
10
8
6
4
2
0
500 x 500
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
6000 x 6000
7000 x 7000
8000 x 8000
9000 x 9000
10000 x 10000
Pixel
SEQUENTIAL
4 THREADS
CUDA

Results


SPEED UP
35
30
25
20
15
10
5
0
Speed-up
500 x 500
1000 x 1000
2000 x 2000
3000 x 3000
4000 x 4000
5000 x 5000
6000 x 6000
7000 x 7000
8000 x 8000
9000 x 9000
10000 x 10000
Pixel
Speed up-OpenMP
Speed up-Cuda

UNIVERSITÀ
DEGLI STUDI
FIRENZE

Histogram Equalization

Leonardo Casini
Harjinder Singh Sandhu