



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE



# Histogram Equalization

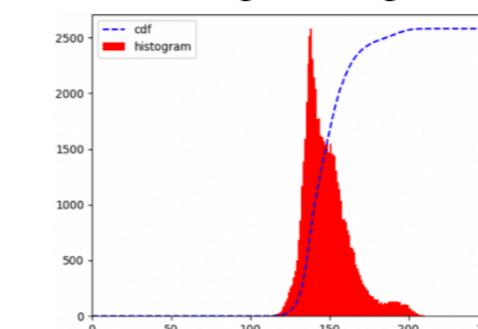
Leonardo Casini  
Harjinder Singh Sandhu

# Abstract

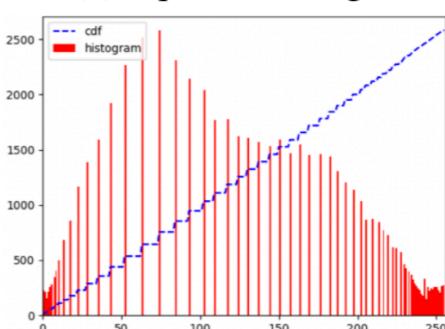
- **Image histogram:** graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value. The histogram represents the number of pixels for each intensity value considered.
- **Histogram equalization:** method used in Image Processing through which 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:

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

- *cumulative distribution function*:

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

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

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

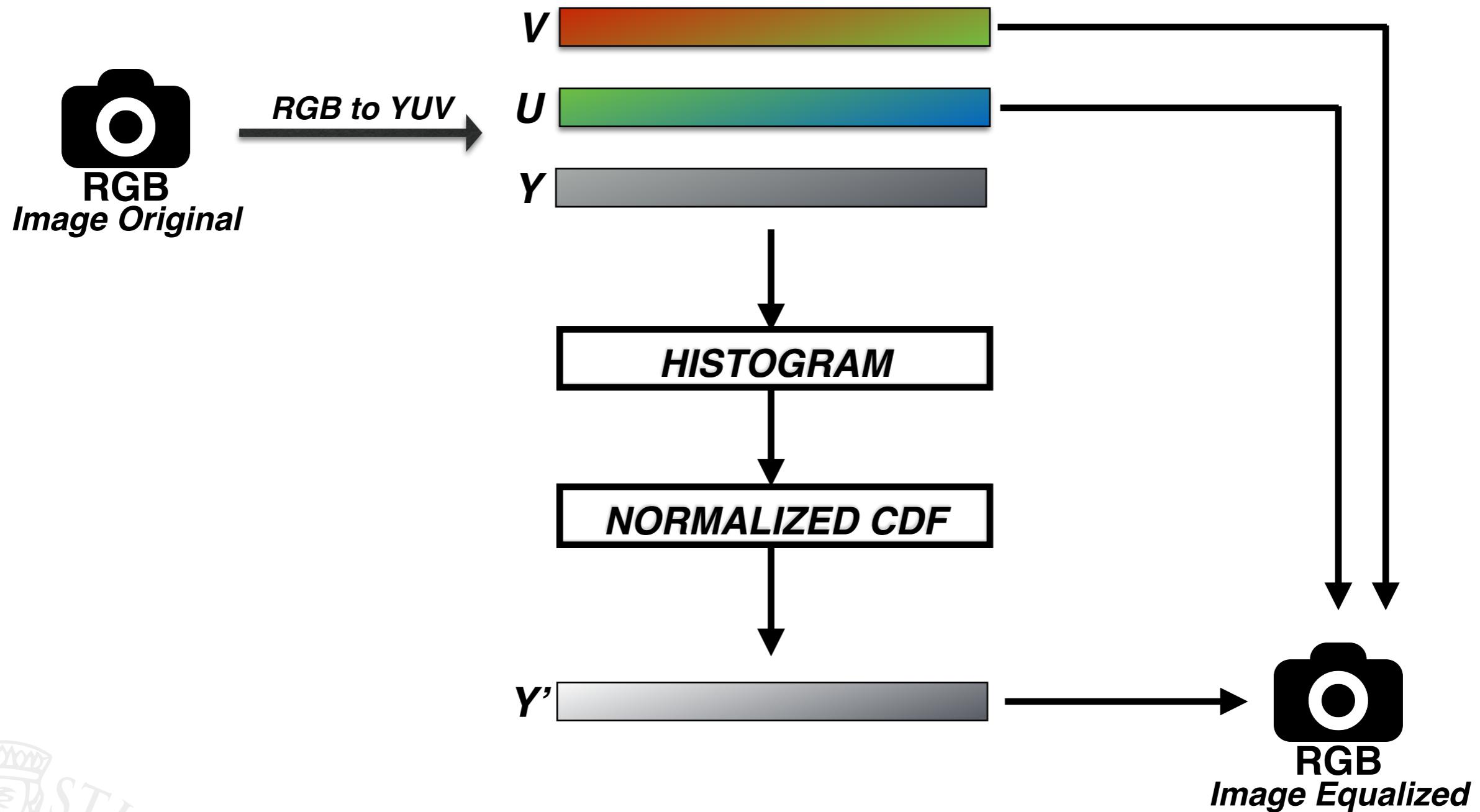
# Technologies

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





# Sequential





# 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

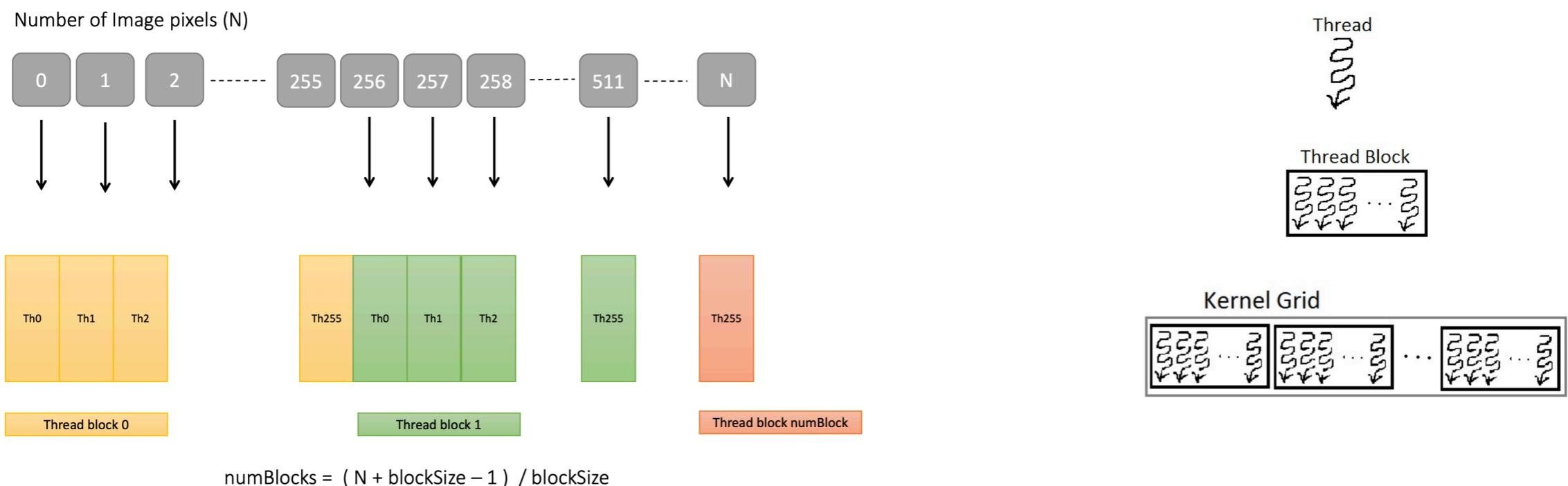


- CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia.
- It allows to use a CUDA-enabled GPU for parallel calculation.
- Functions used while implementing:
  - *cudaMalloc()*: to allocate arrays in the GPU
  - *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*.

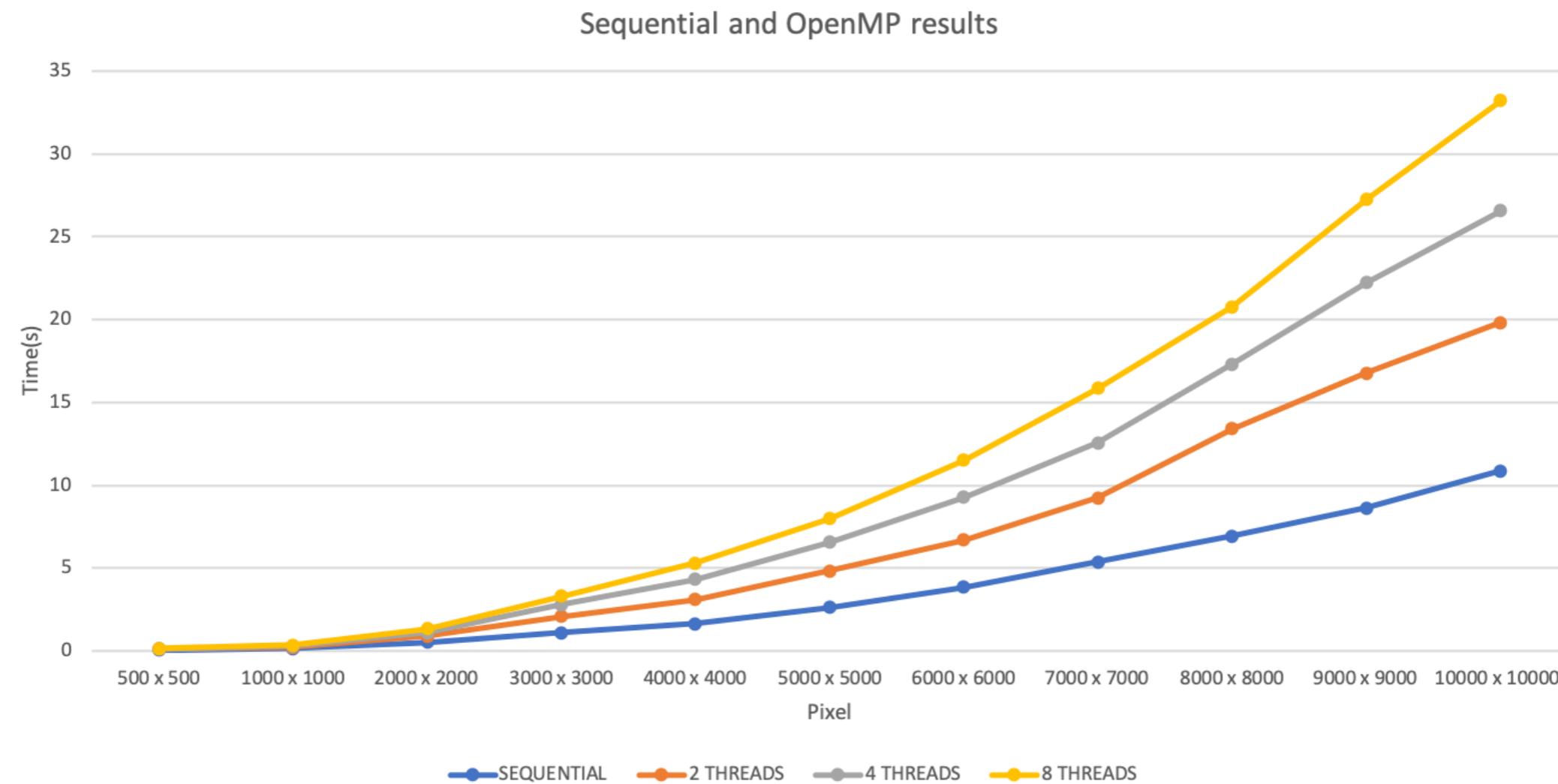
# Parallel CUDA



- 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.

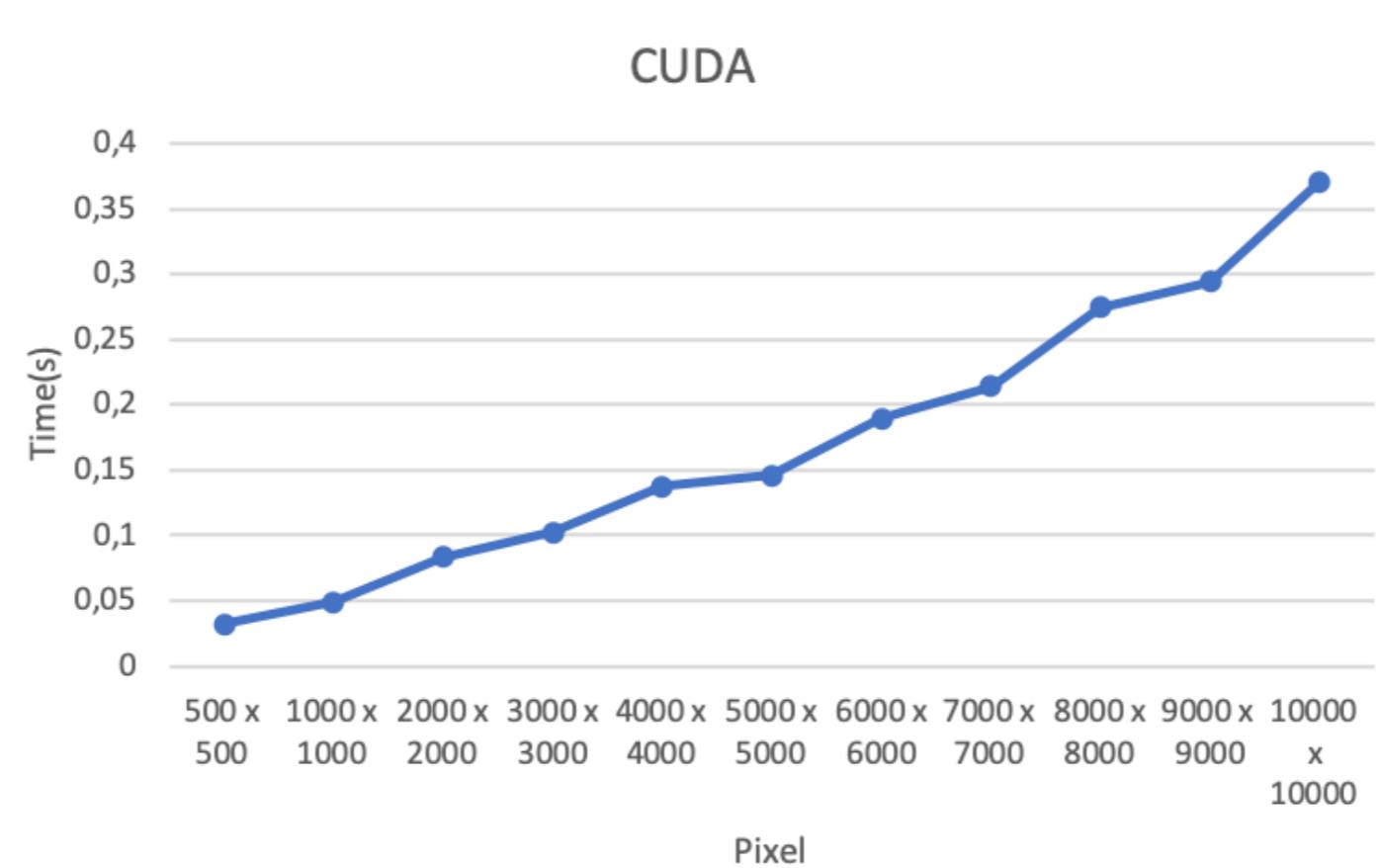


# Results



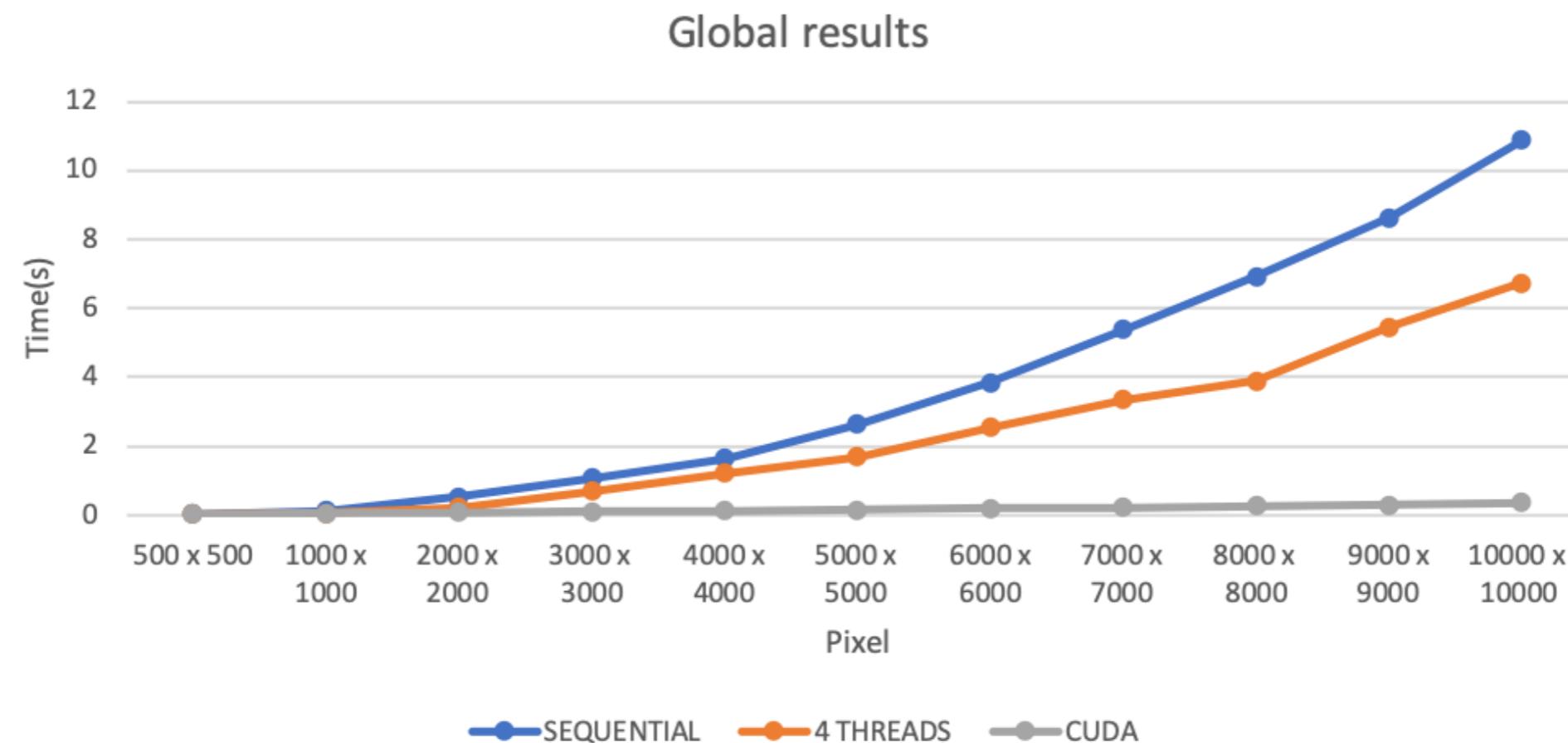


# Results

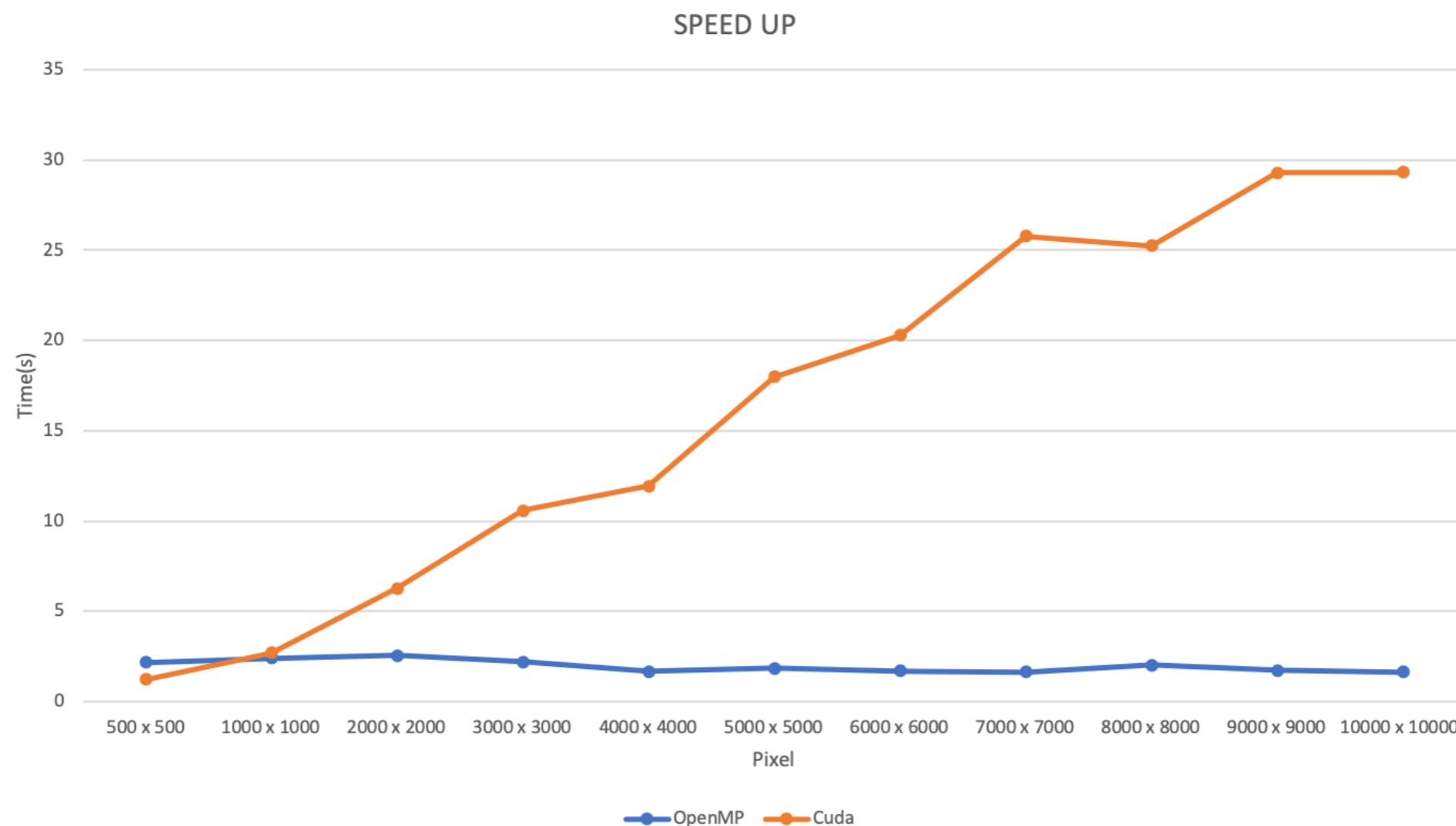




# Results



# Results





UNIVERSITÀ  
DEGLI STUDI  
FIRENZE



# Histogram Equalization

Leonardo Casini  
Harjinder Singh Sandhu