UNIVERSITÀ
DEGLI STUDI
FIRENZE

Histogram Equalization

Giulio Calamai
Marco Loschiavo

UNIVERSITÀ DEGLI STUDI FIRENZE

Abstract

- An **image histogram** is a type of histogram that acts as a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each tonal value.

- Histogram equalization is a method used in Image Processing through which the contrast can be calibrated using the image histogram.

- This method usually increases the global **contrast** of many images, especially when the usable **data** of the image is represented by close contrast values.

- Through this adjustment, the **intensities** can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

Math

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

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

- *cumulative distribution function*

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

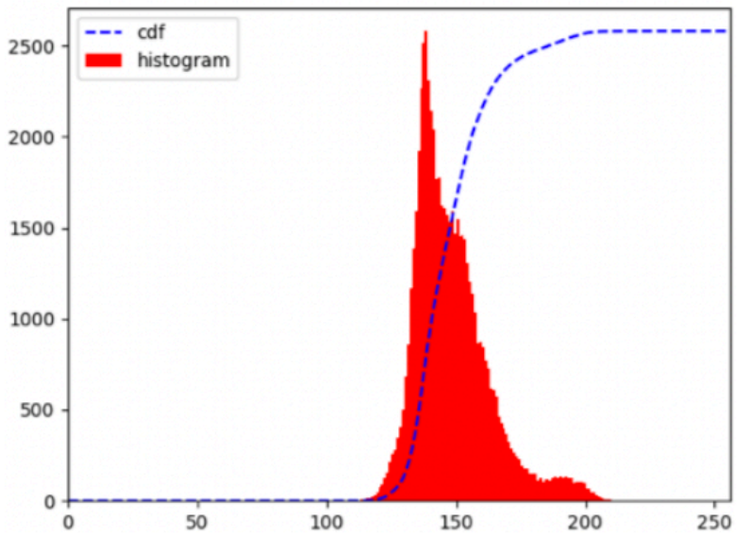
- *normalize* it such that the maximum value is 255:

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

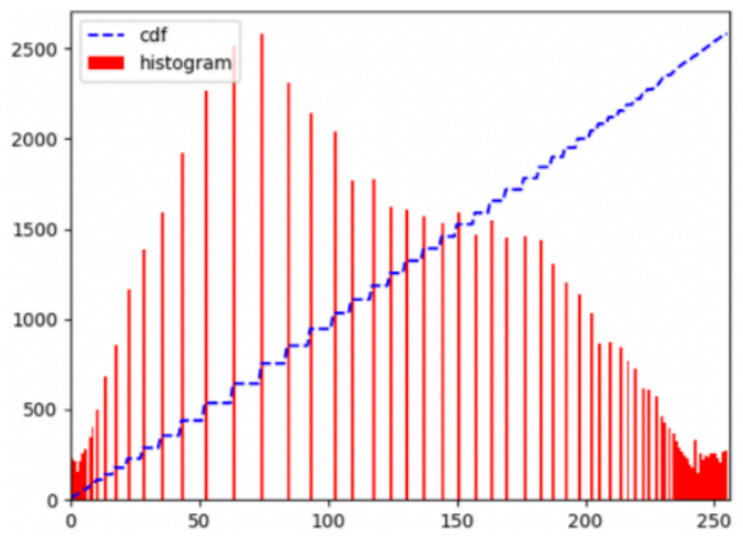
Example

(a) Original Image

(b) Equalized Image



(c) Original Histogram

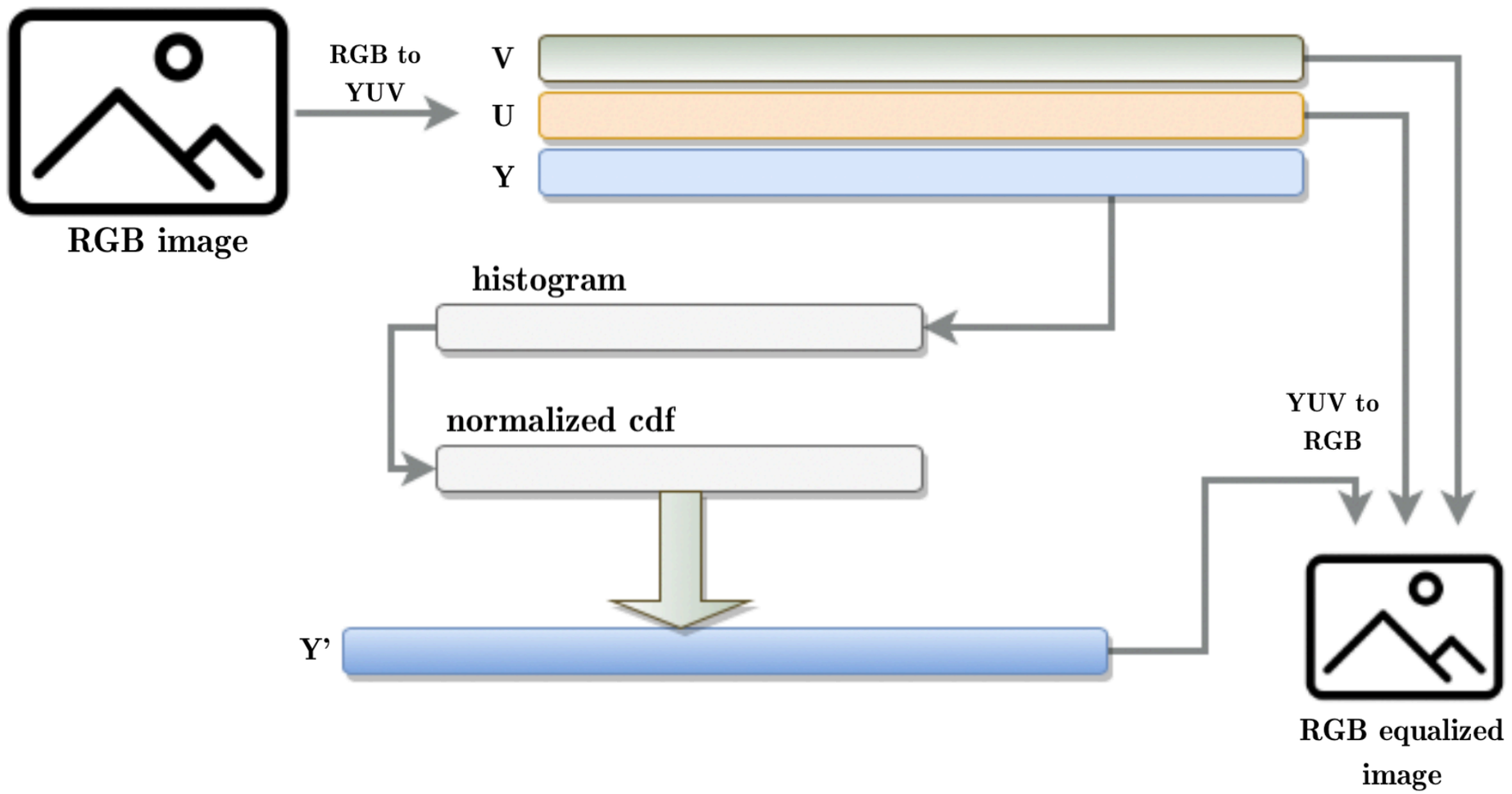


(d) Equalized Histogram

Technologies

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

Serial



- RGB⇒YUV

Parallel : OpenMP

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 /*
            }
        }
    }
```

Parallel : CUDA

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

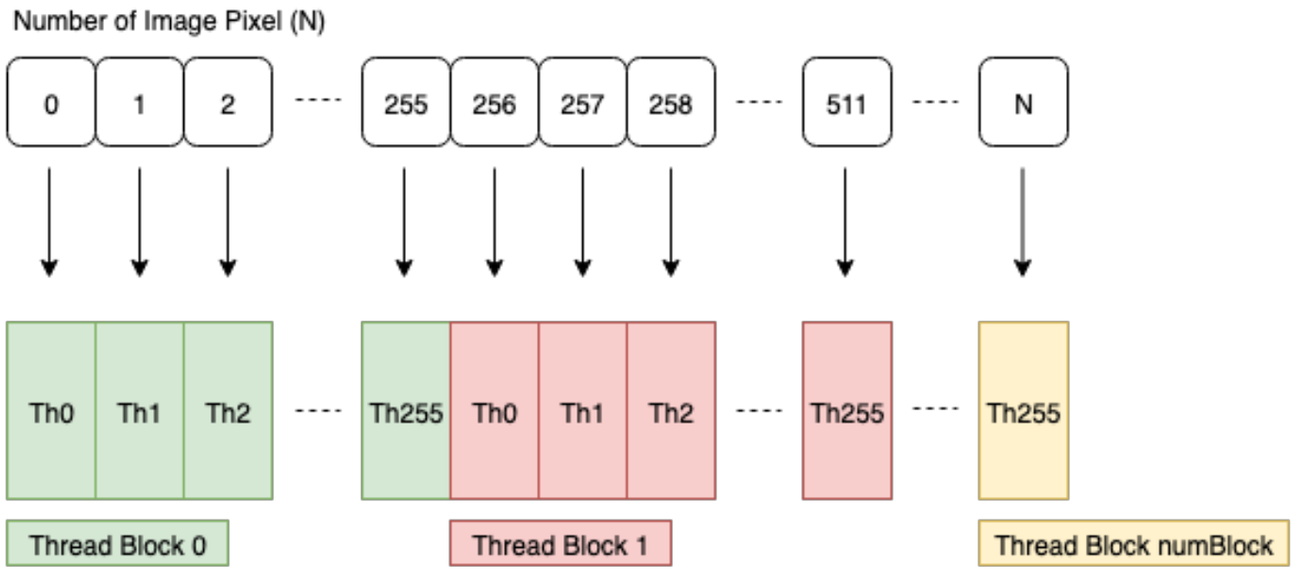
The CUDA kernels have been implemented are:

- make_histogram()
- normalizeCdf()
- equalize()

Parallel : CUDA

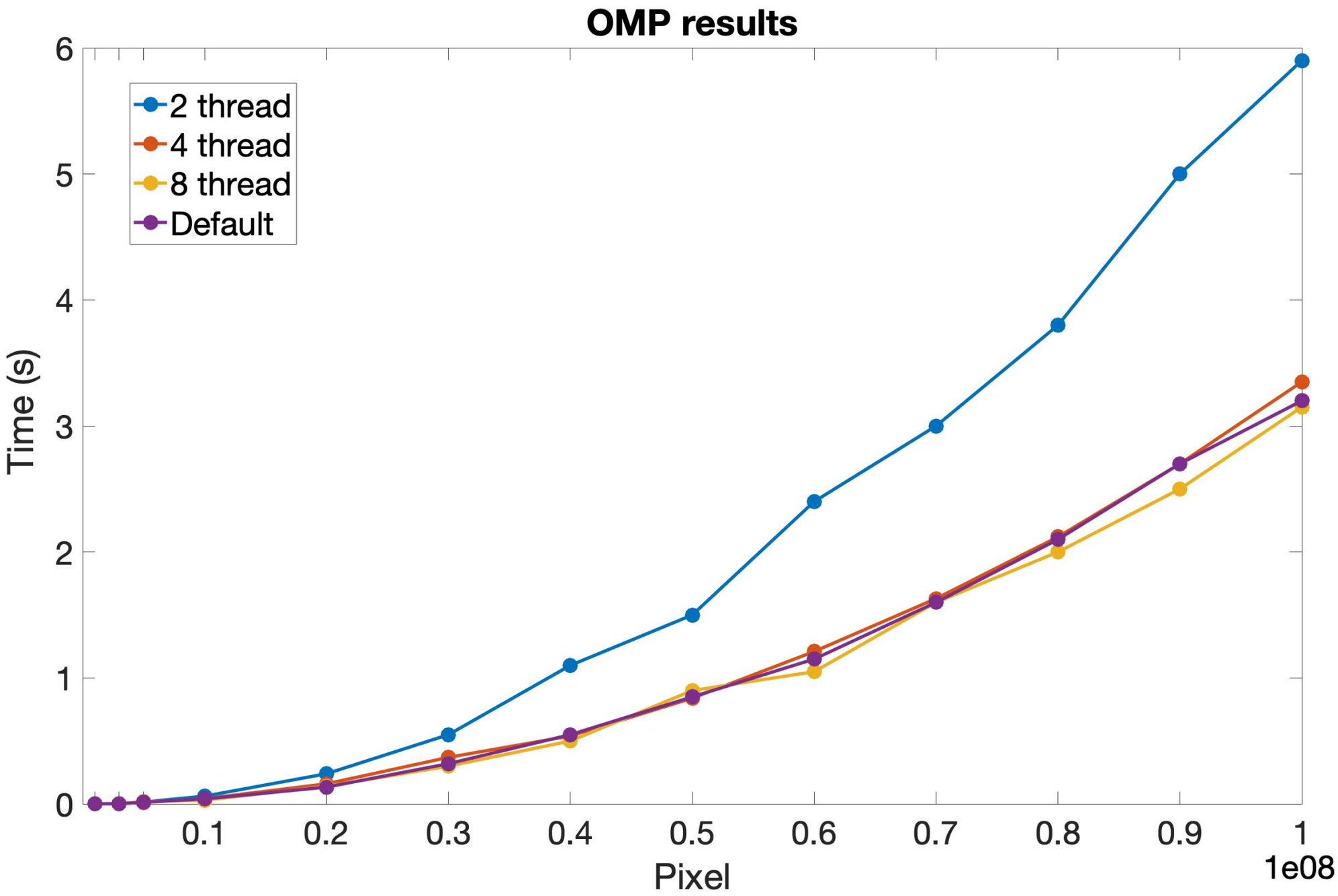
In order to implement CUDA version we have paid special attention on image's pixels management:

- Images have been computed assigning one thread for each pixel
- The thread blocks contain 256 threads

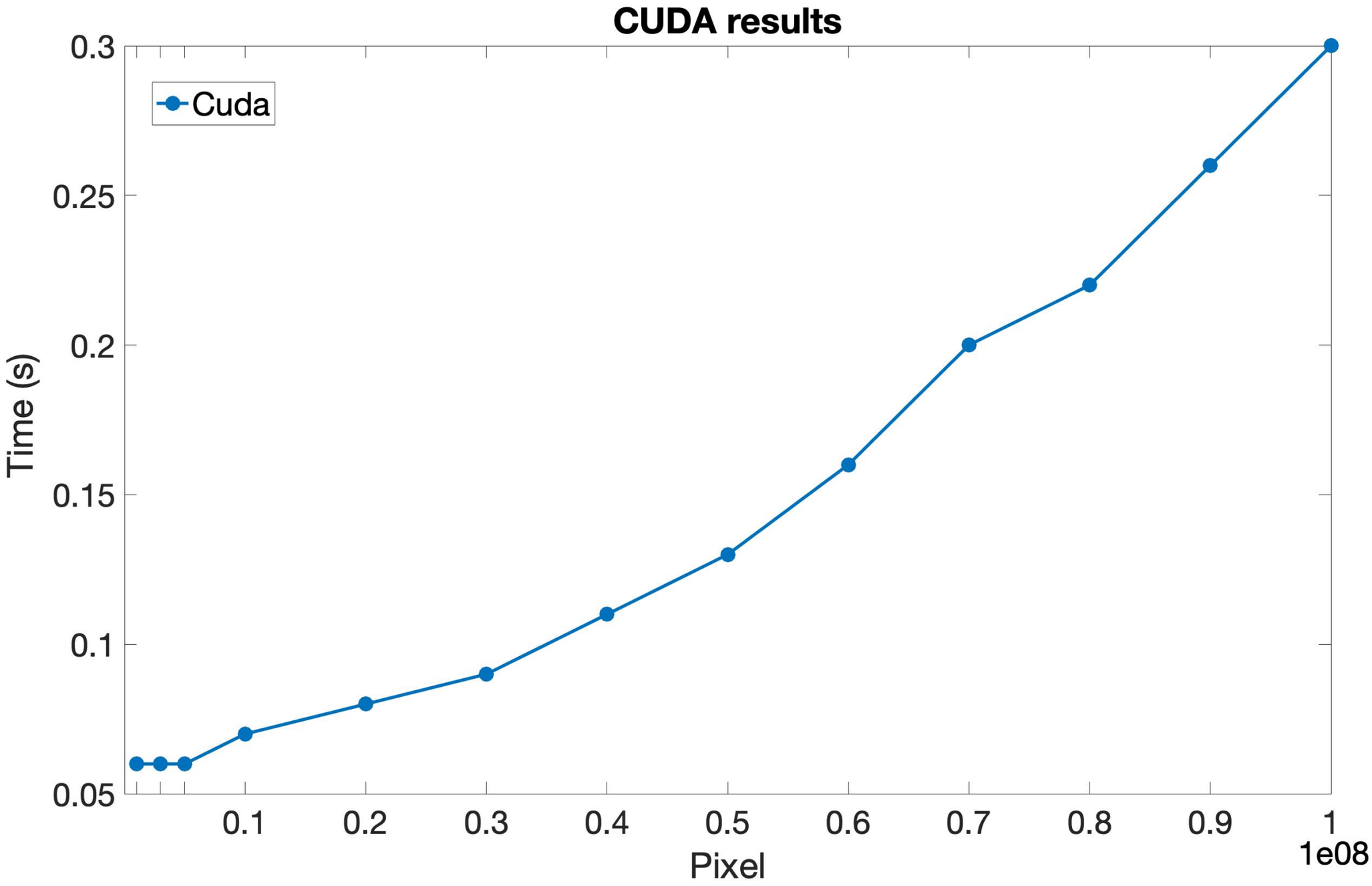



```
numBlocks = (N + blockSize -1) / blockSize
```

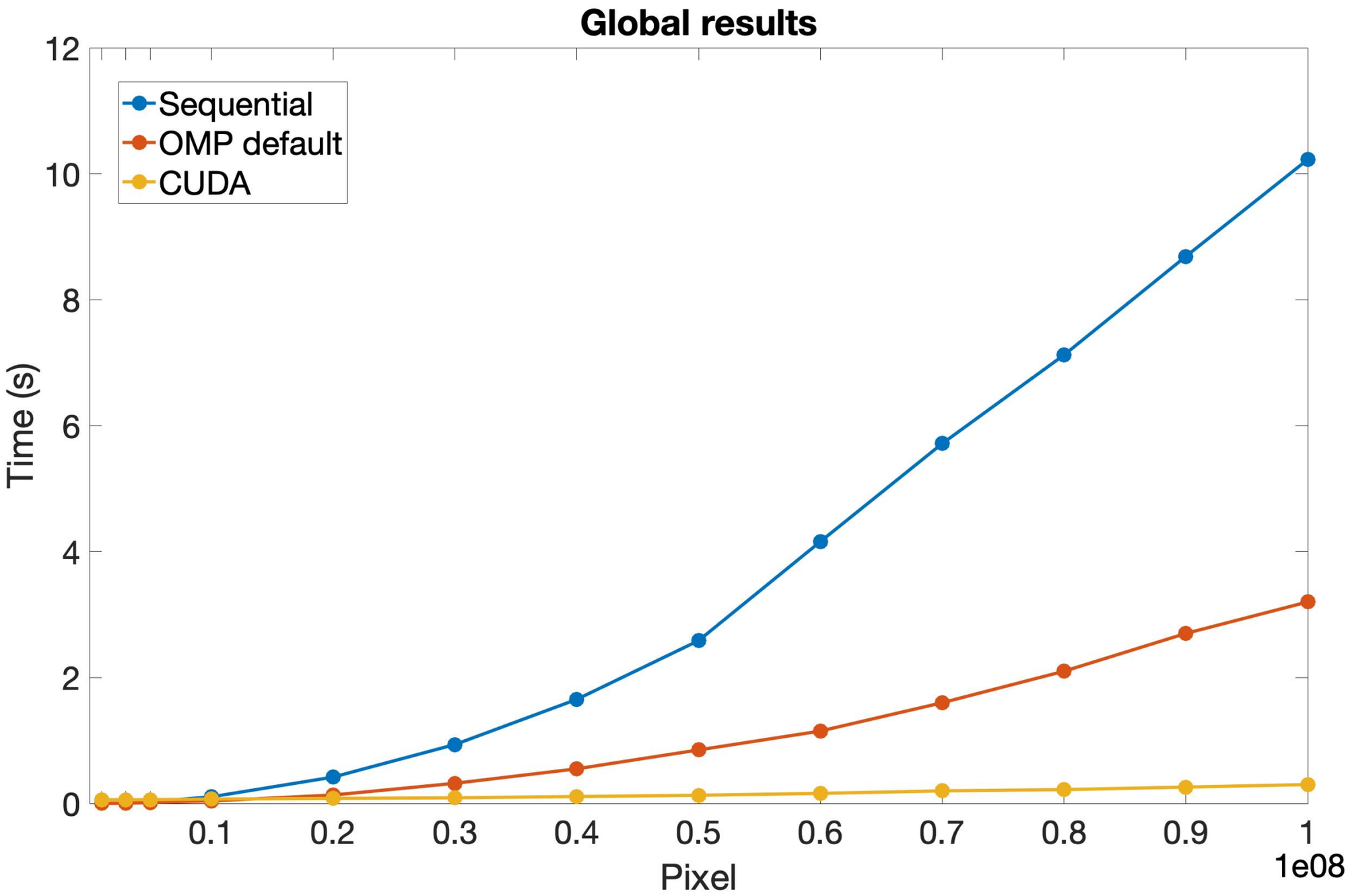
Results


OMP results
2 thread
4 thread
8 thread
Default
Time (s)
0
1
2
3
4
5
6
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1e08
Pixel

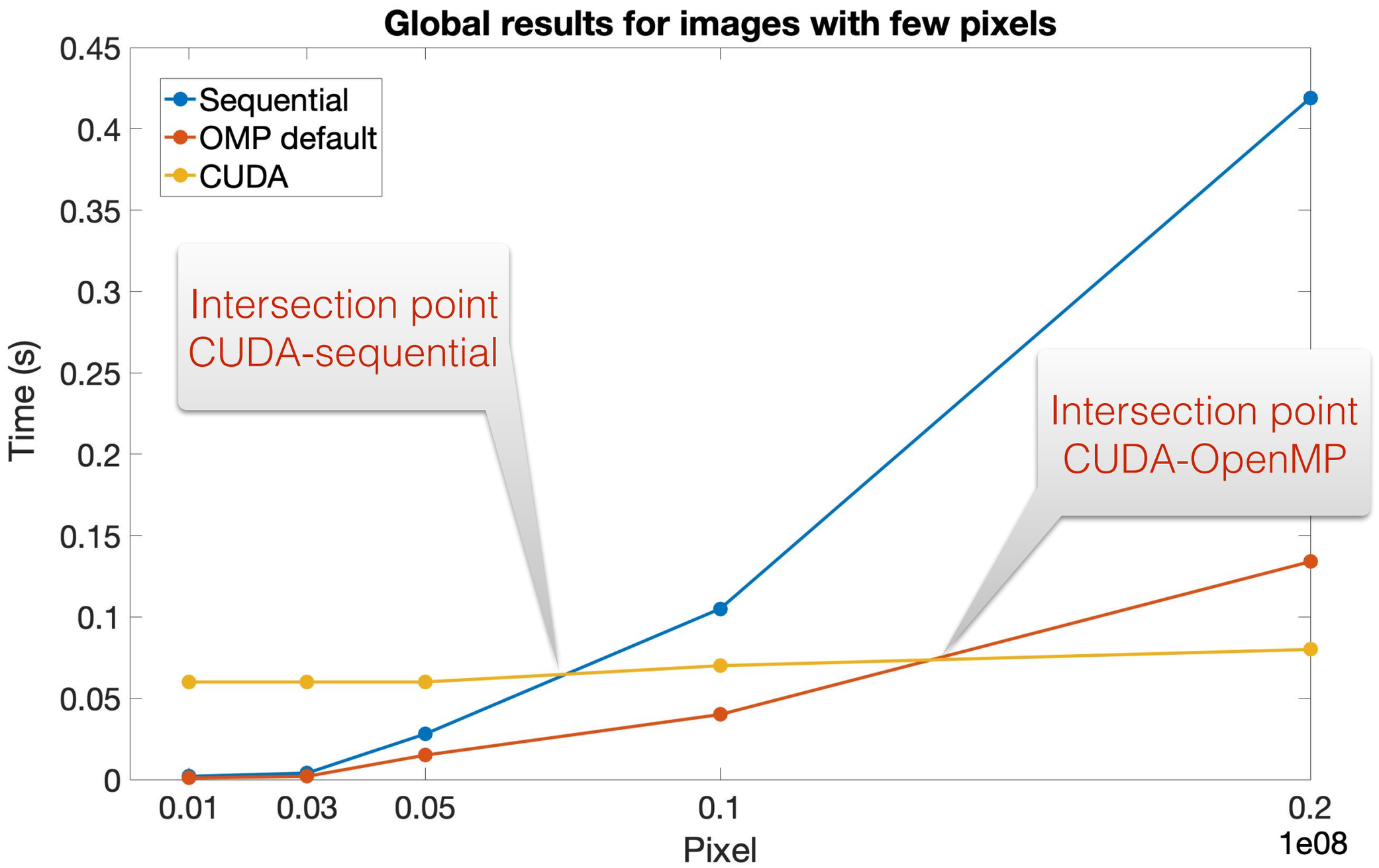
Results


CUDA results
Cuda
Time (s)
0.3
0.25
0.2
0.15
0.1
0.05
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1e08
Pixel

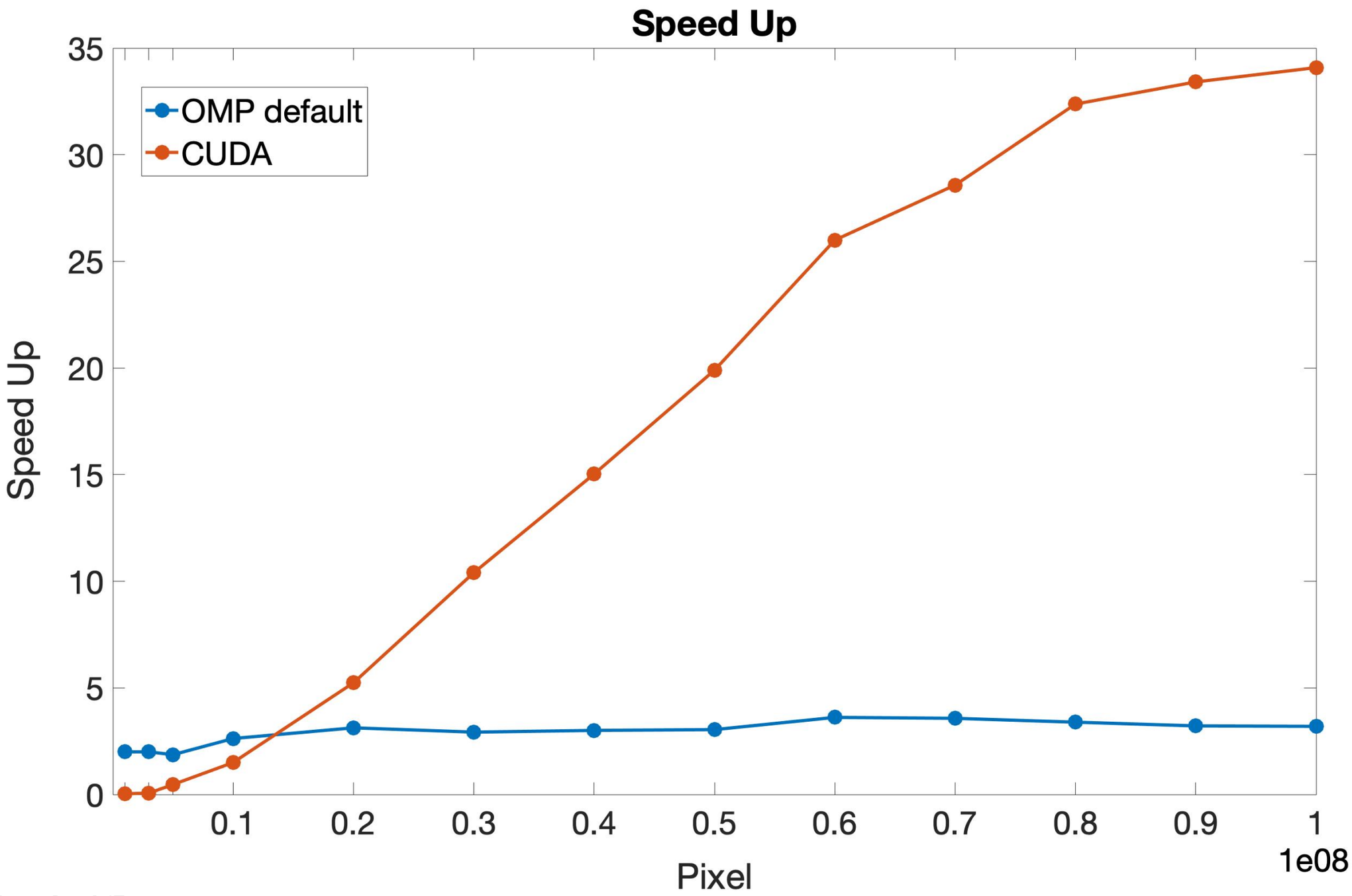
Results


Global results
Sequential
OMP default
CUDA
Time (s)
Pixel
1e08
0
2
4
6
8
10
12
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Results


Global results for images with few pixels
Sequential
OMP default
CUDA
Intersection point
CUDA-sequential
Intersection point
CUDA-OpenMP
Time (s)
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.01
0.03
0.05
0.1
0.2
Pixel
1e08

Speed Up


Speed Up
OMP default
CUDA
Speed Up
35
30
25
20
15
10
5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1e08
Pixel