Technical Report #2

Image Box Blurring

Alejandro Herce Bernal

September 2018

1. Introduction

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

For this project, we're using OpenCV to manipulate images by applying a 5x5 box blur while evaluating the performance differences between CPU and GPU processing.

2. Development

A. Technical specifications

- CPU
  - Intel Core i7-4790k
  - Frequency: 4.4 Ghz
  - Cores: 4
  - Logic cores: 8
  - L1 cache: 256 KB
  - L2 cache: 1.0 MB

    - L3 cache: 8.0 MB
- GPU
    - Nvidia GeForce GTX 980 Ti
    - Memory: 6 GB GDDR5
    - CUDA Cores: 2816
    - GPU Clock Speed: 1.24 GHz
    - Memory Clock Speed: 3505 MHz

B. Testing

For the testing, different configurations were selected to measure the difference in performance. Regular CPU processing, CPU with OpenMP and GPU with CUDA.

In both CPU and GPU different configurations were tested to measure performance. In CPU, different number of threads were tested, in GPU, different block size configurations.

For each of the settings, 10 tests were executed and the execution time was averaged.

The image used is a 4K wallpaper with a step of 11520, 2160 rows and 3840 columns.

A. GPU Testing

For GPU testing, different block sizes were selected with the same formula for the grid.

Block Size	Average time (10 tests, in ms)
8x8	0.026
16x16	0.026
32x32	0.025
64x64	Incomplete image
16x2	0.029
16x4	0.026
16x8	0.025
32x2	0.024
32x4	0.028
32x8	0.026

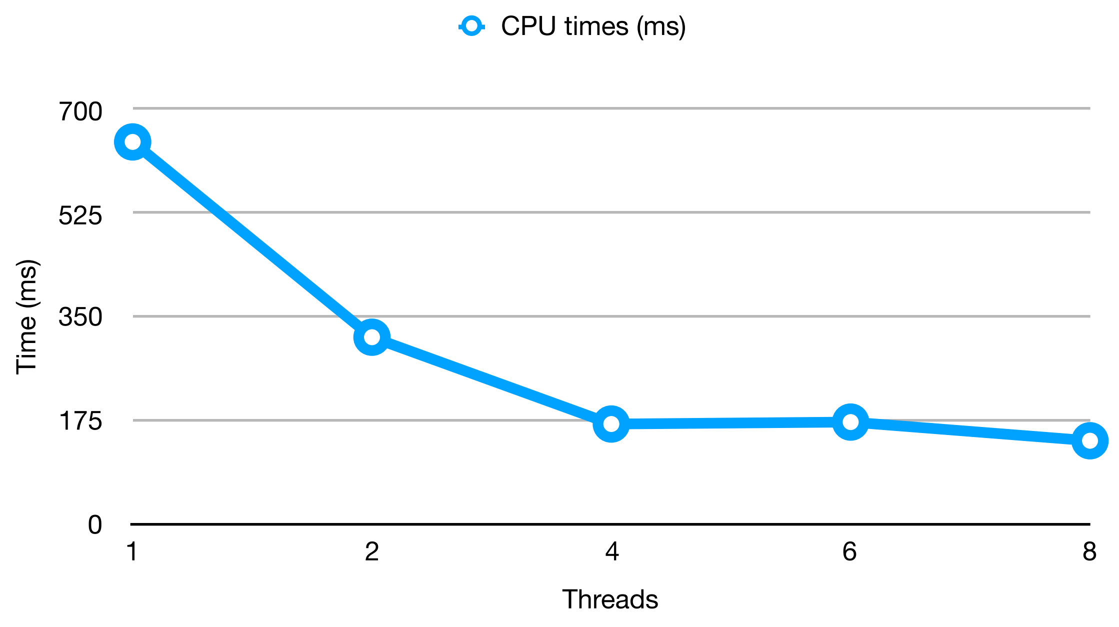
Block Size	Average time (10 tests, in ms)
32x16	0.026
64x2	0.027
64x4	0.026
64x8	0.028
128x2	0.028
128x4	0.034
128x8	0.027

The most consistent results were offered by the 32x32 and 32x2 configurations, where results between each of the 10 tests were very similar, changing no more than 0.003 milliseconds.

And inverting the dimensions, for example using 2x32 instead of 32x2, did not make any significant difference to the times, at most it varied around 0.002 and 0.005 milliseconds.

B. CPU

For CPU testing, we varied the number of threads for each tests. The processor has 4 physical cores with 8 virtual cores.



Threads	Average time (10 tests, in ms)
1	645.34
2	315.26
4	168.53
6	171.82
8	140.24

C. Comparison and Speedup

For speedup times, we have the following information:

	CPU time	GPU time	Speedup
CPU parallel	140.24	0.024	5,843.333
CPU no parallel	645.34	0.024	26,889.166

3. Conclusions

Usually the trick with these kind of tests is finding the optimal block size. But the main difference between this test and the previous one is that changing the block sizes doesn't affect significantly the execution time. While it gets a millisecond better, at most, all configurations tested were performing pretty much the same. And as mentioned before, inverting the dimension, for example 2x32 instead of 32x2, didn't affect significantly the execution time. However, some block configurations, specially the bigger ones, returned a black image and failed to execute the blur.