Technical Report

Omar Sanseviero Güezmes
September 2018

Summary

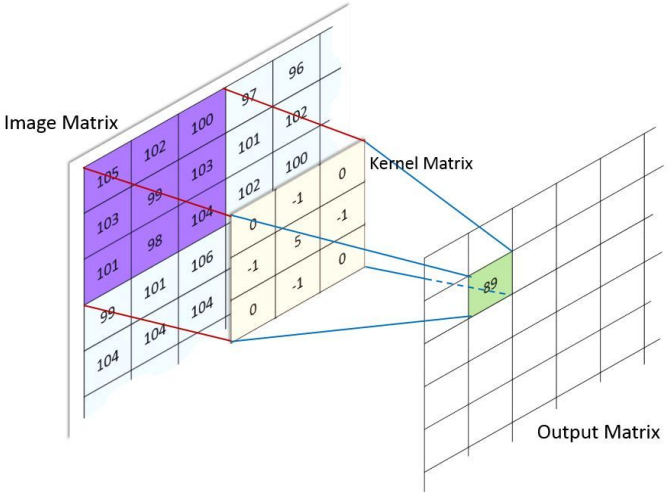
This report explores implementing a box blur algorithm using CPU and GPU and comparing the running times. Using GPU shows a huge advantage for images of all sizes when comparing against CPU implementations. The report compares empirical results (execution time) of the different configurations and different image sizes.

1. Introduction

Images are stored as matrices, and, as such, allow us to use the boost of using parallel computing with GPU. A computing intensive operation with images is applying convolutional filters to change them, like using a blur algorithms or edge detection. A simple blur algorithm is box linear filter, in which the result of each pixel is the average of its neighboring pixels. This report compares the computing times of applying a blur filter to images of different sizes with CPU and GPU.

2. Development and results

To implement the box blur, we use a simple O(n*m) algorithm, where n is the size of the image and m is the size of the filter. For every pixel in the image, the neighboring pixel values are averaged. For this experiment, we'll be using a 5x5 filter. Something to consider when using convolutions is how to work with the borders. There are different ways of achieving this. In this case, we use the 'same padding' strategy so the output image size is the same than the input image size. For border pixels, we only use the number of pixels used to calculate the average. For example, in a corner, the sum of the pixel values is divided by 9 instead of 25.



OpenCV is a library that allows to easily do image manipulation. We'll use it to load and display the images. We measure time with 400x640, 1224x1840, 1836x3264 images. Time is measured using chrono *high_resolution_clock*[1]. We'll use the direct mat classes for the CPU implementations, and convert the matrix to a linear array for the GPU implementations, which might have a direct impact in the performance.

The programs are being run using GeForce GTX 670. It has 1344 CUDA cores and 7 multiprocessors. The GPU max clock rate is 0.98 GHz. The memory clock rate is 3004 Mhz. For CPU the programs are being run in an Intel i7-4770 Processor, which has 4 cores. The clock speed is 3.4 GHz. The CPU threading is implemented using OpenMP.

The following tables show 5 trials for each configuration. All times in the tables are measured in milliseconds. Changing the number of threads in the GPU implementations did not have any representative impact in performance.

CPU			
	400x640	**1224x1840**	**1836x3264**
Trial 1	111.862617	1007.567810	2664.621826
Trial 2	112.598030	1010.070374	2689.509277
Trial 3	112.987930	1008.305115	2656.525391
Trial 4	111.837479	1001.849731	2683.803223
Trial 5	112.779030	1014.352783	2667.580078
Average	**112.4130172**	**1008.4291626**	**2672.407959**

CPU with 8 threads			
	400x640	**1224x1840**	**1836x3264**
Trial 1	158.770020	999.300964	2678.642090
Trial 2	113.180824	1011.381775	2666.061768
Trial 3	116.896111	1001.065613	2654.340820
Trial 4	112.647850	1009.670593	2678.490479

[1] https://en.cppreference.com/w/cpp/chrono/high_resolution_clock

Trial 5	113.959671	1008.388123	2658.730469
Average	**123.0908952**	**1005.9614136**	**2667.2531252**

GPU 2D grid, 1D block, 128 threads per block			
	400x640 5x400	**1224x1840** 15x1224	**1836x3264** 26x1836
Trial 1	0.014175	0.016186	0.022186
Trial 2	0.013603	0.015078	0.018780
Trial 3	0.013326	0.017702	0.020118
Trial 4	0.013399	0.015522	0.019190
Trial 5	0.013344	0.016399	0.019396
Average	**0.0135694**	**0.0161774**	**0.019934**

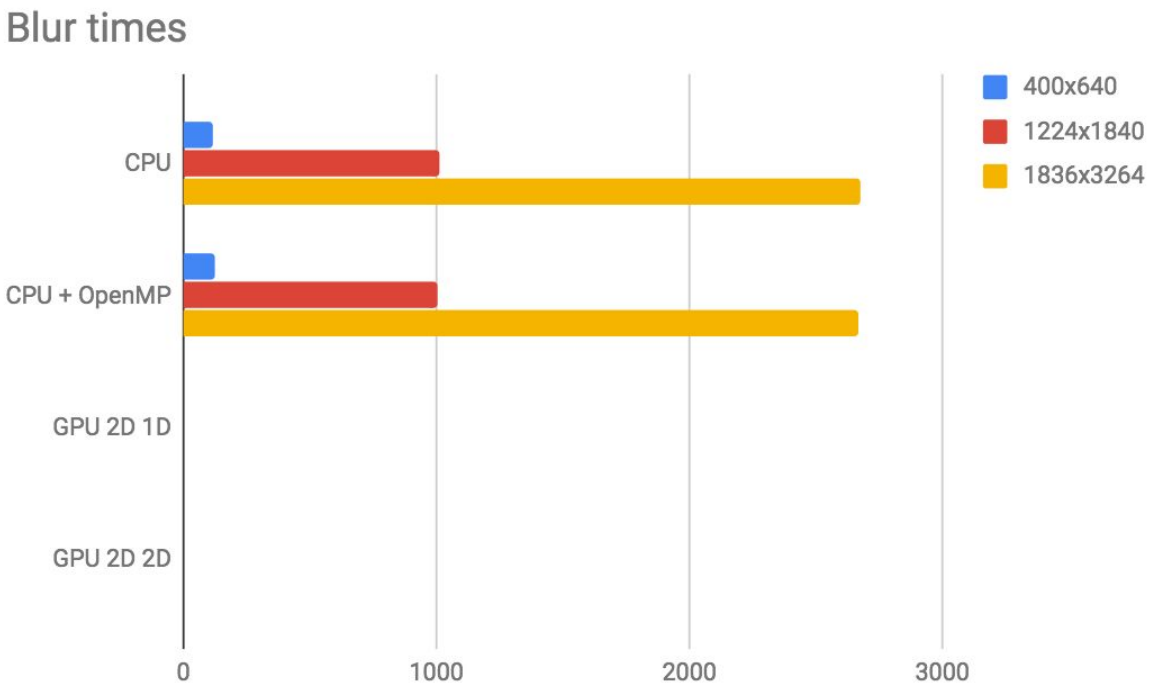
GPU 2D grid, 2D block, 1024x1024 block size			
	400x640	**1224x1840**	**1836x3264**
Trial 1	0.004619	0.004693	0.007993
Trial 2	0.003835	0.005615	0.007245
Trial 3	0.004257	0.005183	0.007875
Trial 4	0.004176	0.006003	0.007440
Trial 5	0.004044	0.004461	0.007257
Average	**0.0041862**	**0.005191**	**0.007562**

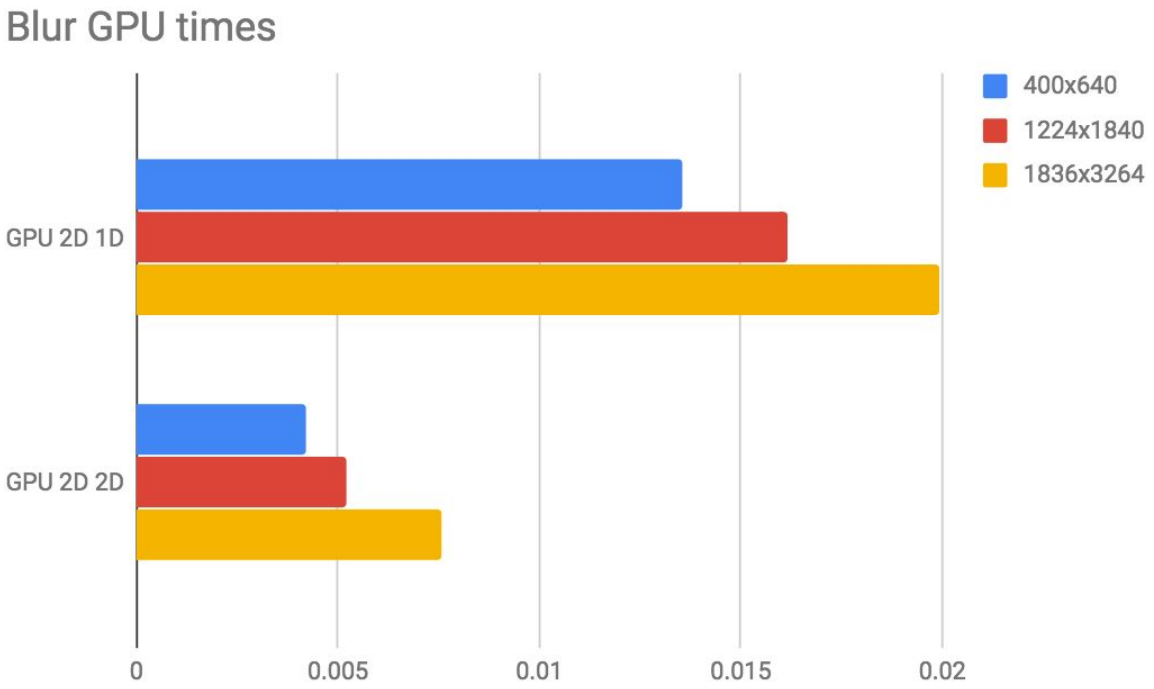
3. Analysis

The following table summarizes the results from the previous section.

	400x640	**1224x1840**	**1836x3264**

CPU	112.4130172	1008.4291626	2672.407959
CPU + OpenMP	123.0908952	1005.9614136	2667.2531252
GPU 2D 1D	0.0135694	0.0161774	0.019934
GPU 2D 2D	**0.0041862**	**0.005191**	**0.007562**





We obtain the best results using 2D grid with 2D array of threads per block. The difference between 2D 1D and 2D 2D is small, but still significant. The difference between CPU and CPU with threads is really small and not necessarily significant.

Speedup from CPU			
	400x640	**1224x1840**	**1836x3264**
CPU + OpenMP	-1.09498	1.00245	1.0019326
GPU 2D 1D	8284.3027105	62335.67585	134062.805207
GPU 2D 2D	26853.23615	194264.912849	353399.62430

As the data gets larger, the GPU speedup increases. Changing the block size in the 2D 2D GPU had a huge impact in the time. 1024x1024 block size has a really high speedup when compared to CPU.

We obtain the best results using 2D grid with 2D array of threads per block. The difference between CPU and CPU with threads is really small and not necessarily significant.

4. Conclusions

Using GPU shows clear advantage for images of all sizes. Changing the block size in the 2D 2D configuration had a high impact on time, and using 1024x1024 block size had a speedup of up to times vs small block sizes or 1D block size (eg 128x1).

A thing that is important to consider is that in the CPU implementations, the solutions are using the OpenCV Mat class. In the GPU implementations, a linear array is used, which offers additional advantages.