TECHNICAL REPORT
SEBASTIÁN GALGUERA ORTEGA
SEPTEMBER 2018
MULTICORE PROGRAMMING

Abstract

The following document describes the process that was followed in order to apply a convolution filter on an image matrix, using sequential and parallel approaches with OpenMp and Nvidia’s CUDA. The aim of the task was to provide a way to apply different filters to process an image and transform its visual structure. Tables are provided to describe the nature of the tools and procedures involved.

1. Introduction

Images can be understood and manipulated as an N matrix that may have a single value or some array of values for each of its indexes. In this case, a 2D matrix was used containing a 3 channel structure that comprises the main three BGR colors, each one ranging from 0 to 255 in magnitude. For this implementation, OpenCV was used for the consumption of an image and its proper allocation in memory and in a structure that defines its features.

Basically, image processing can be done by a convolution, which mathematically measures how much overlapped is a function in respect of another. Using the point wise multiplication property of matrixes, it can provide new characteristics to an existing image.

It can be expressed in the following manner:

$$r(i) = (s * k)(i, j) = \sum_{n} \sum_{m} s(i-n, j-m)k(n, m)$$

For this practice, we used a blur matrix which values compute for the mean of the scalar product of the filter matrix with a 1 to 1 proportional window (sub-matrix) o the image matrix. However, any filter can work for this programs, only having to state the size of the matrix allocated in the global space or the constant memory for the CUDA implementation. Roughly, a convolution looks like this:

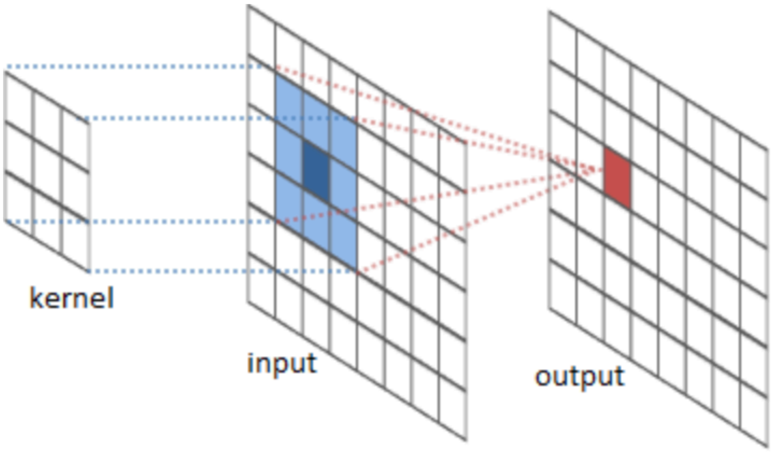


FIGURE 1.0 Convolution process visualization.

2. Development

The main tests were executed on a remote server via secure shell. Two sizes were selected for the calculation. Also, the number of threads and the block and grid sizes were tuned in order to make a comparison between the parameter settings.

The computer specifications:
Brand: Alienware 32 GB RAM
Model name: Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz
CPU: 8
CPU: 3.9 GHZ
CPU Cores: 6
Threads per Core: 2
GPU: NVIDIA Corporation GK104 [GeForce GTX 670] [10de:097a]
GPU Cores: 1344
Architecture: x86_64

The following table specifies the average elapsed time measured in milliseconds of each function for each image size. Chart provided as appended content in section 4.

TABLE OF AVERAGE ELAPSED TIME (ms)	640 × 400	3840 × 2160
Host Sequential	122.058792	4003.184326
Host with Threads	36.226521	1074.146973
CUDA (16,16)	0.023146	0.024533

Now, a table of improvement, a quantity that is a function of time of execution:

TABLE OF IMPROVEMENT (S(n)=T(1)/T(n))	640 × 400	3840 × 2160
Host Sequential	0	0
Host with Threads	3.3693213875	3.7268497018

CUDA (16,16)	5,273.4291886287	163,175.491215913

The following table describes the average time of 3840 × 2160 images with different kernel configurations. Also, a chart will be appended in section 4.

TABLE OF KERNEL CONFIGURATIONS (2D)	AVERAGE TIME (3840 × 2160)
(512,1)	**0.020182**
(1,512)	**0.022501**
(256,2)	**0.020448**
(2,256)	**0.027241**
(64,16)	**0.031636**
(16,64)	**0.023784**
(16,16)	0.024533
(32,32)	**0.023313**

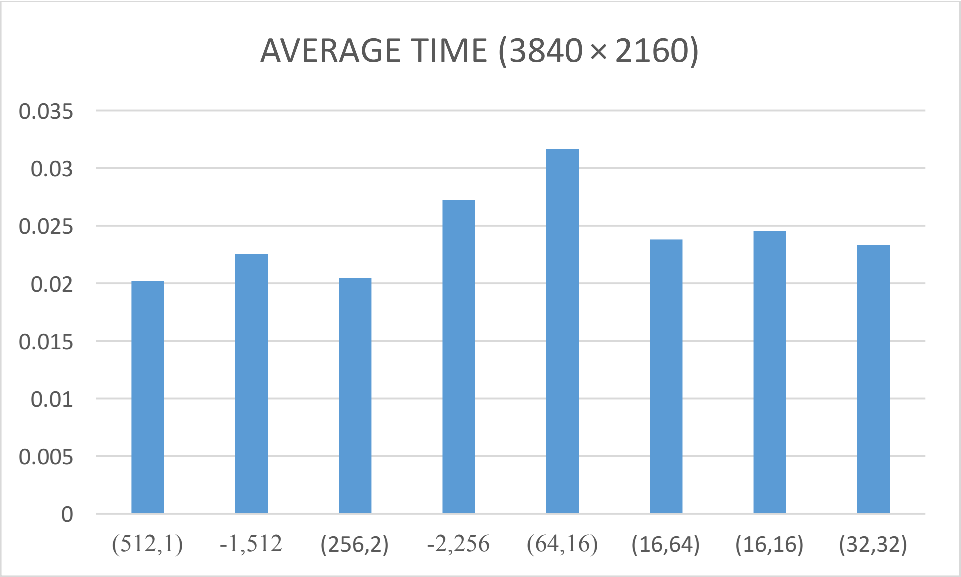
Little is won in the 2D configuration of the kernel. It can be seen how time is almost the same, perhaps of the nature and proportion of the matrix and the filter.

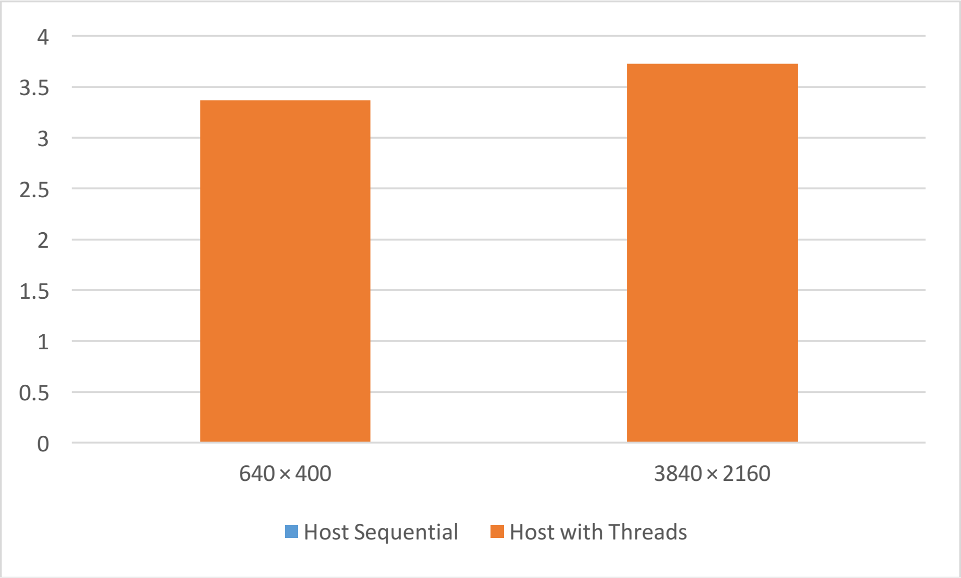
3. Conclusion

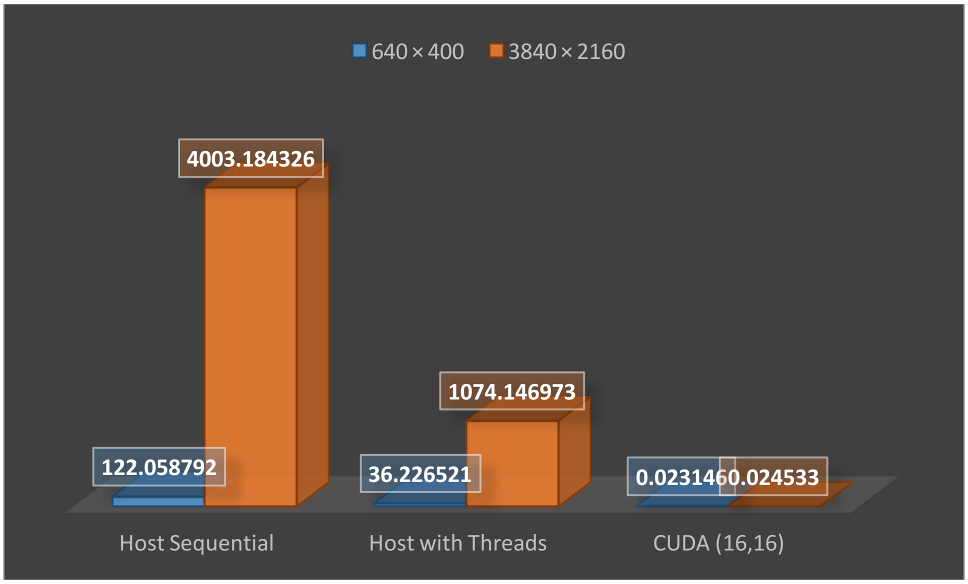
With this, it can be concluded that the vast superiority of CUDA emerges in this kind of problems. Of course, convolution filtering can have several caveats while dealing with idle threads, different kernels and configurations, but the incredible speed of NVIDIAs technology is obvious. Also, it is worth to note that the average time of computation is independent of matrix size for this examples. Many other optimizations can be done in these codes in order to make them faster like using a serialized array as a filter and using compiler optimizations. Even though this can be optimized in the future, the current results are beyond sequential realm.

4. References and Figures

“GEFORCE RTX™.” *NVIDIA*, NVIDIA, 1 Sept. 2018, www.nvidia.com/es-la/.
Podlozhnyuk, Victor. “Image Convolution with CUDA .” NVIDIA, 11 June 2007.


AVERAGE TIME (3840 × 2160)
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
(512,1)
-1,512
(256,2)
-2,256
(64,16)
(16,64)
(16,16)
(32,32)


4
3.5
3
2.5
2
1.5
1
0.5
0
640 × 400
3840 × 2160
🟦 Host Sequential
🟧 Host with Threads



640 × 400
3840 × 2160
4003.184326
122.058792
1074.146973
36.226521
0.023146
0.024533
Host Sequential
Host with Threads
CUDA (16,16)