A Comparative Performance Evaluation of OpenCV and libccv

Additional Work Plan (Contract 4716.8)

Prof. Guido Araujo

1 Introduction

This workplan has for goal to define the scope and activities of the tasks needed to characterize the best image processing library for Samsung mobile devices. As such, it will do a thorough profiling of the OpenCV and libccv libraries when executing on Samsung devices, using multicore and GPU architnectures. It will also make a preliminary evaluation of the potential of using the AClang compiler to optimize such libraries for GPUs.

This workplan is divided as follows. Section 2 does a qualitative assessment of the the OpenCV and libccv libraries to better understand the following library features: (a) code quality; (b) availability of parallelization and optimization techniques; and (c) maintainability and support. In Section 3 we do a very preliminary comparative analysis of the performance of three well-known filters from these libraries, which reveal that there is plenty of scope for performance improvement in libccv. As a matter of fact, for these particular filters, OpenCV on ARM-NEON considerably outperforms libccv. In Section 4 we shortly describe the goals of this work plan and provide an execution schedule. Finally, in Section 5 we describe the activities required for this work plan.

2 Analysis of OpenCV and libccv

This section shows a short, preliminary, comparative study between the Open Source Computer Vision Library (OpenCV)[1] and the Computer Vision Library (libccv)[2].

2.1 OpenCV

OpenCV is an open source computer vision and machine learning software library that includes more than 2,500 optimized algorithms [1]. The OpenCV's community today counts with more than 47,000 people that contribute directly with bug reports and improvements.

There are several advantages of using OpenCV, as follows:

- OpenCV has support for vector extensions (AVX, SSE and NEON). It implements different vector extensions through a *template* called Hardware Acceleration Layer (HAL). The idea is that one can use a single SIMD code template that is compiled to either SSE or NEON instructions, depending on the target platform. In the case of Samsung mobile devices, the hardware can take advantage of routines that have NEON implemented;
- OpenCV makes use of some BLAS routines (OpenBLAS, MLK, ATLAS);
- OpenCV has an active community. With the reports and contributions done by the OpenCV community in its version 3.0, about 200 bugs were fixed; and, besides that, some improvements were done related to the enhancement of 40 routines in the Android environment;
- There are several Android applications using OpenCV, e.g, Facebook, Instagram, Snapchat, etc;
- OpenCV has implementations of CUDA and OpenCL for several routines. Besides that, OpenCV also provides support for TBB and OpenMP.

On the other hand, we also detected some drawbacks when using OpenCV OpenCL routines on Samsung mobile devices. Its OpenCL implementations were built with improvements targeting only Intel and AMD. Hence, the execution on Samsung mobile GPUs may produce slowdowns since it is not optimized for this kind of environment.

2.2 libccv

As discussed in [2], libccv has for goal to provide a simpler and organized image processing code that can be easily deployed. According to its author (Liu Liu), libccv implements a handful state-of-art algorithms, developed mainly to execute on mobile environments (Android & iPhone-iOS).

The advantages claimed by libccv are much less beneficial when compared to OpenCV. Since it has just a few routines implemented with vector extension, libccv makes few uses of BLAS routines. Besides that, it makes almost no use of parallelism even within routines that clearly expose good data parallelism potential.

Tabela 1: Absolute time of OpenCV and libccv executions.

Environment	**Blur (seconds)**	**Canny (s)**	**Sobel (s)**
libccv	1.639797	0.722346	0.225597
OpenCV-NEON-CPU	0.031524	0.406320	0.055624
OpenCV-OpenCL	0.165318	error	0.465160

Moreover, libccv has some additional disadvantages that make its usage not as attractive as OpenCV; they are:

- libccv has a limited documentation;
- As far as we know from public domain, there is only one person supporting libccv (its creator - Liu Liu);
- It implements fewer routines when compared to OpenCV;
- Maybe it was faster than OpenCV sometime between 2010 – 2014;
- It does not have any OpenCL routine;
- The code is very complex, with several macros, and is hard to maintain;
- The last release was done in 2014; and
- There is almost no developer community - consequently - it does not have support.

As a preliminary conclusion, we have observed that libccv is not prepared to exploit the maximum computational power of Samsung mobile devices. Its code does not use almost any parallelization construct, and thus it does not utilize the parallelization features of the ARM-NEON and GPU architectures available in Samsung devices.

3 OpenCV vs libccv

In a preliminary study, we have performed some experiments to compare the execution time between OpenCV and libccv. Table 1 shows the absolute execution time taken from the execution of three filters: Blur, Canny and Sobel. Each filter was re-executed 5 times, with the same 4K picture, in order to compute its average execution time.

The experiments were performed in a Samsung mobile S7 edge with an Exynos 8890 Octa-core CPU (4x2.3 GHz Mongoose & 4x1.6 GHz Cortex-A53) integrated with an ARM Mali-T880 MP12 GPU (12x650 Mhz) running Android OS, v6.0 (Marshmallow). The libccv library was compiled with the flag NEON activated; the OpenCV-NEON-CPU was also compiled with NEON activated. The execution used two threads in the CPU execution, and the OpenCV-OpenCL was compiled with support of OpenCL kernels.

Since we tested both OpenCV and libccv with NEON-CPU and the same programs (filters) it was expected that the performance would be similar for both libraries. However, due to its parallelized and optimized code, OpenCV revealed much better speed-ups than libccv (e.g. in the case of the Blur filter, it produced a 51x speed-up). As shown in Table 1, OpenCV-NEON-CPU also outperformed libccv and OpenCV-OpenCL. In the case of OpenCV-OpenCL, this happened because the OpenCV OpenCL kernels were not designed to run on Samsung mobile GPUs, but only on the Intel-Iris and AMD-Kaveri architectures.

4 Our Proposal

```
1  #pragma omp target device(GPU_DEVICE)
2  #pragma omp target map(to:a_ptr[0:(rows-1)*aSize]) \
                      map(from:b_ptr[0:(rows-1)*bSize])
3  #pragma omp parallel for collapse(2)
4  for (i = 1; i < rows - 1; i++) {
5    for (j = 0; j < cols; j++) {
6      for (k = 0; k < ch; k++) {
7        ((int * )b_ptr)[(i - 1) * bSize +  j * ch + k] = \
8        (int)(a_ptr[(i + 1) * step +  j * ch + k] + \
9        2 * a_ptr[(i - 1) * step +  j * ch + k] + \
10       a_ptr[(i - 2) * aSize + j * ch + k]);
11     }
12   }
13 }
```

Code 1: A block of code from Sobel Filter

As observed during this study, both libraries were not fully designed to execute in a Samsung mobile environment. This creates a series of opportunities o achieve better performance on OpenCV or libccv, by using the AClang to compile and optimize OpenMP 4.X code to Samsung GPUs.

As a example of an AClang potential, we extracted an example of a code block from the Sobel filter of libccv as shown the Code 1, that reveal how a programmer can extract parallelism through OpenMP 4.X in AClang. By just adding some annotations to the code (*pragmas*), AClang is capable of generating optimized OpenCL code for execution on Samsung devices. As a future extension of this work plan, we are considering to measure AClang execution on OpenCV and libccv in order to check the improvements that can be achieve by the execution of optimized GPU kernels.

5 Schedule

For this work plan we will focus on characterizing and make a comparative study of OpenCV and libccv. In order to implement this, we have to perform the following two major tasks: (a) make a thorough study of the OpenCV and libccv libraries; (b) do a performance analysis of the major routines used in these bechmarks for Samsung multicore and GPU architectures. The schedule of the activities required to perform these tasks is shown in Table 2.

1. Study of the libccv library.
2. Study of the OpenCV library.
3. Design of a benchmark that uses libccv functions.
4. Design of a benchmark that uses OpenCV functions.
5. Performance analysis of OpenCV and libccv benchmarks in multicore and GPU
6. Comparative evaluation of OpenCCV and libccv performance
7. Preparation of the benchmark release and final report

Tabela 2: Schedule for the next months

Activity	Months 2017/2018					
	Out.	Nov.	Dez.	Jan.	Feb.	Mar.
1	✓	✓				
2		✓	✓			
3		✓	✓	✓		
4			✓	✓	✓	
5		✓	✓	✓	✓	
6			✓	✓	✓	
7					✓	✓

Referências

[1] Open Source Computer Vision Library. Accessed: September 20, 2017. [Online]. http://opencv.org/.

[2] A Modern Computer Vision Library. Accessed: September 20, 2017. [Online]. http://libccv.org/.

[3] ACLang compiler. Accessed: September 20, 2017. [Online]. https://omp2ocl.github.io/aclang/.