Research Article

A FAST AND ENERGY EFFICIENT PARALLEL IMAGE FILTERING IMPLEMENTATION ON RASPBERRY PI'S GPU

Aytunç POLAT[1], *Salih BAYAR*[2*]

This paper presents a powerful processing technique for fast and energy-efficient image filtering algorithm focusing energy and time-sensitive embedded and robotic platforms. Digital video processing is getting more and more popular in battery-powered devices like mobile robots and smartphones whereas in most cases, it leads overhead on the main central processing unit (CPU) and it consumes a significant amount of energy from the battery. It is suitable for parallelism since there is no data dependency between the steps of the two-dimensional convolution algorithm. We propose a vector version of the two-dimensional convolution algorithm, which can run parallel on embedded processors that has general purpose graphic processing unit (GPGPU), to reduce computation time and energy consumption. Our in-depth experiments shows that using GPGPU could reduce the execution time while guaranteeing lower power consumption and offloading the system CPU. Experimental results showed that we achieved up to 105 times faster operation and 100 times less energy consumption compared to the CPU implementation. Besides, we reduced the CPU overhead up to 10 times.

Key words*: General Purpose Graphic Processing Unit (GPGPU), parallelism, parallel processing, vector processing, speedup.*

1. Introduction

Digital image processing, augmented reality and computer vision are quickly progressing research fields in collaboration with plenty amount of applications in both academia and industry. At the same time, it becomes harder for processors to deal with the increasing digital image sensor resolutions and frame rates efficiently. To satisfy user expectations, higher performance is compulsory while energy efficiency is evenly important for long battery life [1, 2]. When we consider the battery-powered devices, real-time performance requirement is still one of the key components that hard to ensure with a limited energy capacity.

To enhance instruction per second value of the single-core processors, processor designers chose to increase the clock frequency until physical drawbacks appeared such as overheat, instability and power inefficiency [3]. These drawbacks lead the industry to the development of Single Instruction Multiple Data (SIMD) machines and multicore processors [4]. The general expression of Complementary Metal-Oxide-Semiconductor (CMOS) power model is given in Eq. (1).

[1] Department of Electrical and Electronics Engineering, Marmara University, Istanbul, Turkey, (aytuncpolat07@gmail.com) https://orcid.org/0000-0001-7700-9159

[2] Department of Electrical and Electronics Engineering, Marmara University, Istanbul, Turkey, (salih.bayar@marmara.edu.tr) https://orcid.org/0000-0002-4600-1880

Received: 31 March 2020; Accepted: 25 August 2020 Doi: https://doi.org/10.36222/ejt.708805

$$P_{chip} = A\,C\,f\,V^2 + I_{leak}V + P_{short} \tag{1}$$

In Eq. (1), A is electrical current, C is characteristic capacity, f is frequency, V voltage, I_{leak} is leakage current. First term of the Eq. (1) $A.C.f.V^2$ is the power of caused charging and discharging of the logic gates. Second term $I_{leak}.V$ describes the leakage power loss independent from logic state of the gate. The last term P_{short} is the short circuit power which appears while switching process from one state to another state.

The ARM CPU cores, which are in almost every embedded processor, have a high instruction per second value per watt. However, in most cases, they require video acceleration co-processors and display controllers for better performance. Blake et al. [5] present a very detailed work about processor comparison. Another work was made by Pang et al. [6] that is an extensive comparison of the energy efficiency of clusters of ARM and x86 processors. There are many works about image processing using Compute Unified Device Architecture (CUDA). One of these works is the work of Yang et al. presented in[7]. Today, embedded mobile processors have many-core Graphics Processing Units (GPUs) that have high data throughput. That brings an opportunity to accelerate computationally intensive applications on low power platforms [8]. Since using GPGPUs to accelerate algorithms can be power-intensive, optimized methods should be constructed to run algorithms [9]. 2D image convolution is a key operation in digital image processing and computer vision algorithms for object tracking, feature extraction and segmentation [10]. In this study, we are particularly interested in the performance comparison of a popular embedded processor and an embedded GPU.

In the digital domain, convolution is a two-step process which is composed of multiplication and accumulation operations successively. To do that, two overlapping samples multiplied element-by-element and those multiplication results registered onto a new data field. Since there is no data dependency among the possible different convolutions, the operation can be easily parallelized onto small processors like GPU cores. Likewise, multiplication operation is also data-independent in convolution. Hence, parallel vector operations can be applied to this operation, as well. 2D convolution is illustrated in Figure 1.

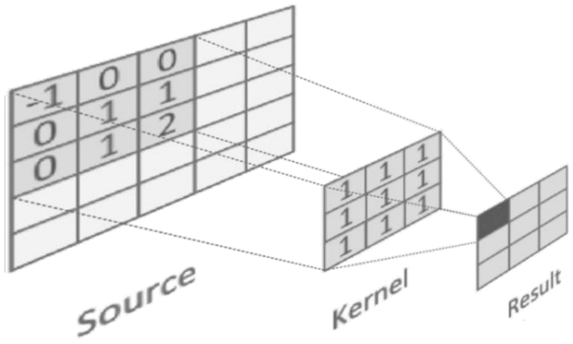


Figure 1. 2D Convolution.

General expression of the 2D convolution is give in Eq. (2).

$$g(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x-s,y-t) \tag{2}$$

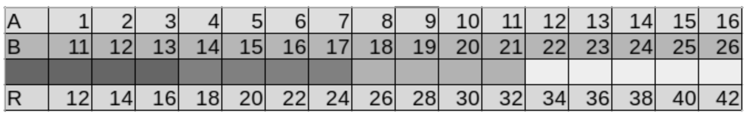
Where $g(x,y)$ is the output image, $f(x,y)$ is the input image and w is the filter kernel.

2. Architecture

VideoCore is a low power multimedia processor owned by Broadcom and its VideoCore IV series graphic processors found in the Raspberry Pi 1, 2, 3. The 3D engine architecture based multiple floating-point shading processor units called Quad Processing Unit(QPU) [11]. The main specification points of graphic processing units are:

- 250MHz clock frequency
- 25M triangles
- 1G pixels/s with single bilinear texturing, simple shading, 4x multisampling.
- Supports 16x coverage mask antialiasing for 2D rendering at full pixel rate.
- 720p standard resolution with 4x multisampling.
- Supports 16-bit HDR rendering.
- Fully supports OpenGL-ES 1.1/2.0 and OpenVG1.1.

A QPU is a 16-way Single Instruction Multiple Data (SIMD) processor. An SIMD machine has a single controller that spreads instructions to all of the processors. The processors execute these instructions at the same time [12]. Each QPU has two vector floating-point Arithmetic Logic Units (ALU) which deal with multiply and non-multiply operations in parallel using single instruction cycle delay. Exemplarily each SIMD processor performs add operation on 16-dimension vectors of 32-bit integer or floating-point values like following Figure 2.



A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
B	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
R	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42

Figure 2. Vector Operation Scheme

In Figure 2, A and B are input vectors and R is result vector.

Inside the QPU, the architecture is a 4-way SIMD processor muxed 1x4 over four clock cycles. Each 16-dimension vector formed with four quads and this is where the quad processing unit name comes from. QPUs are arranged into groups of slices which share instruction cache, a special function unit, texture and memory lookup units and interpolation units. The QPUs are scheduled automatically by the VideoCore hardware with QPU scheduler. Raspberry Pi-3 contains 12 QPUs in its graphic engine. Each QPU has following key features:

- Uniform 64-bit instruction encoding
- 4-way physical parallelism.
- 16-way virtual parallelism.
- A dual-issue floating point ALU.
- Two large single ported register.
- Five accumulators.
- I/O mapped into the register file.

In 2014, Broadcom Inc. announced to release of documentation for VideoCore IV and sources files for linux drivers under BSD license so that we were able to use the VideoCore GPU as a very powerful parallel accelerator in our work.

3. Algorithm

3.1. Scalar Algorithm

To make an appropriate comparison between conventional scalar 2D convolution algorithm and our proposed algorithm, we first developed a scalar algorithm code in C++ language which can easily run on a single thread or multi-threaded structures like Pthread or OpenMP. To run the scalar algorithm on multi-threads, we broke down the input image into stripes as many as threads available. With this method, different threads can work on different sub-images independently. Pseudocode for 2D convolution is given in Alg. 1.

```
Algorithm 1: Scalar 2D Convolution Algorithm
Input: Kernel, Input image
Output: Output Image

for (horiziontalScan){
      for (verticalScan){
            for(k=0;k<kernelSize;k++){
                  for(l=0;l<kernelSize;l++){
                        pixVal+=input.at(i+k-a, j+l-a)*kernel[k][l];
                  }
            }
            output.at(i,j)=pixVal/kernElm;
            pixVal=0;
      }
}
```

In Alg. 1, the input variable is input image matrix, the output variable is output image, *kernelSize* variable is the length of the one dimension of the kernel matrix, *kernElm* variable is the number of the kernel elements and a variable is integer part of half of the *kernelSize*. In the real scalar code, loop unrolling technique used in the inner loops that perform kernel multiplication to optimize execution speed of the scalar algorithm. This helps much to make a fair comparison between algorithms.

3.2. Vectoral Algorithm

We use QPUlib which is already embedded QPU library on the latest Raspbian image. Our first attempt to use QPUs to perform vector operations almost identical to the scalar version. There is only one difference which is every kernel multiplication operation between kernel and input image executing on a dedicated QPU slot during a single kernel multiplication. Since we use a 3x3 kernel, kernel multiplication utilizes nine QPU elements during execution. Summation of multiplications still runs as scalar at the first attempt. This type of parallelism might accelerate math operations but loading pixel data to GPU and storing pixel data to the memory (i.e. Random Access memory (RAM)) slows down the overall performance of the algorithm. Since memory access is critical in time, memory to compute ratio is also a key factor to apply parallelism on any algorithm [13].

$y = *ptr$ statement is a blocking data transfer statement in C and C like programming languages. Blocking statements slow down the algorithm drastically. In order to get rid the effects of data transfer operations, non-blocking data transfer methods must be used. To do that we split the algorithm into three parts:

- data fetch,
- data store and,

- computation

These parts can be executed simultaneously; during the computation part, previous data output is stored into the memory and the next input data is fetched from memory at the same time.

Inside the QPUs, there are FIFO buffers that can store four elements. Using FIFO buffers, we can load and store the data during computation. This type of load and store operations are called non-blocking memory access operations. The same algorithm can run on multiple QPUs simultaneously. Each QPU computes a different part of the overall image like multi-threading, which we already used in scalar code.

4. Experimental Results

We did comprehensive experiments to find out the speed and energy consumption of conventional CPU algorithm and our proposed GPU algorithm. To do that, we used grayscale images of sized 500x500 and 1000x1000 pixels. In the convolution algorithm, 3x3 blurring kernel is used. The algorithm has been applied to the image 50 times in both CPU and GPU experiments. Every experiment repeated ten times and their outputs averaged to obtain accurate results.

Two different sizes of the image are used to get rid of the effects of thermal throttling function on the CPU and GPU. In this way, we can figure out thermal throttling happened or not. At Room Temperature and Pressure (r.t.p) conditions with passive cooling, we did not experience any thermal throttling issues both CPU and GPU algorithm.

The electrical current measured with an Allegro ACS712 hall effect current sensor through micro USB power input. Customized hardware has been used for data acquisition from the current sensor. In our experiments, we use 5.1Volts for power supply voltage. Current value obtained from current sensor in real time. Electrical power calculated with instantaneous power formula which is given in Eq. (3).

$$P = Vt \tag{3}$$

Energy consumption calculated with electrical energy formula is given in Eq. (4).

$$E = Pt \tag{4}$$

Since electrical power P is a dynamic variable, we use total energy formula to integrate energy consumption during execution of algorithms. Total energy expression is given in Eq. (5).

$$E = \int_{t=0}^{T} P(t)dt \tag{5}$$

where P is instantaneous power, V is voltage, I is electrical current, E is electrical energy and t is time.

Execution speed of CPU algorithm shown in Figure 3. As seen in the Figure 3, the computation time of the blurring algorithm almost linearly decreases with the increasing number of cores. In the scalar CPU algorithm experiments Pthread library is used to create a multitasking executable to utilize the multicore CPU. Pthreads, also known as Posix Threads, is a standard programming interface to task creation, task managing and task synchronization [14]. When using all four cores of CPU, processor usage is increased to 100%. As a result of this, the processor becomes unresponsive for all other tasks.

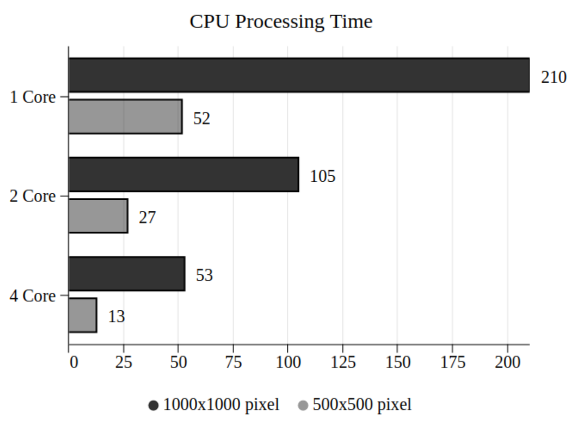


Figure 3. Processing time (Seconds) / Number of CPU core (Lower is better)

Figure 4 shows how much current the Raspberry Pi 3 board draws while the blurring algorithm is running on the CPU. As can be clearly seen from Figure 4, each CPU core draws approximately 50mA extra current in full usage.

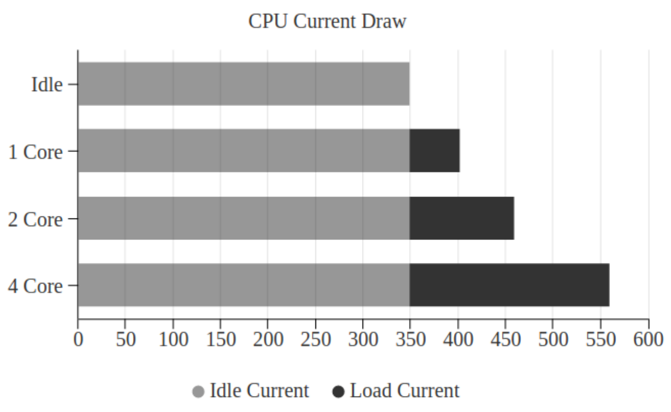


Figure 4. CPU Current Draw (Milliamps) / Number of CPU core (Lower is better)

As seen in Figure 5, GPU algorithm has an enormous speed advantage over CPU implementation. In all cases, GPU approach is much faster than the CPU approach. For example, while processing an image of size 1000x1000 takes around 210 seconds (see Figure 3), the processing of the same image in GPU with only 1 QPU takes only 5 seconds (see Figure 5). This shows that we have a speedup of 42 for this comparison.

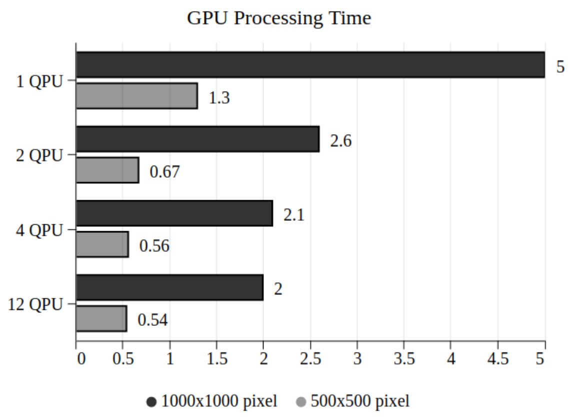


Figure 5. Processing time (Seconds) / Number of QPU core (Lower is better)

Figure 6 shows how much current the Raspberry Pi 3 board draws while the GPU blur algorithm is executing. As can be seen from Figure 6, while the GPU algorithm is executed, a maximum of 80mA current is drawn from the power supply.

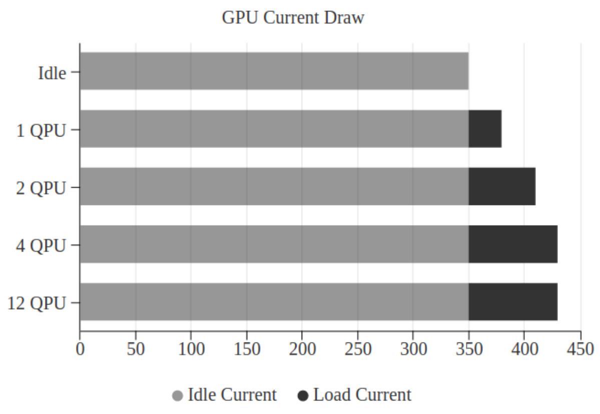


Figure 6. GPU Current Draw (Milliamps) / Number of CPU core (Lower is better)

Figure 7 shows how much speed up is obtained with GPU over CPU. As can be clearly understood from Figure 7, the performance of the GPU seems to be up to 105 times better than the performance of the CPU. The GPU implementation outperforms the CPU implementation for all QPU values.

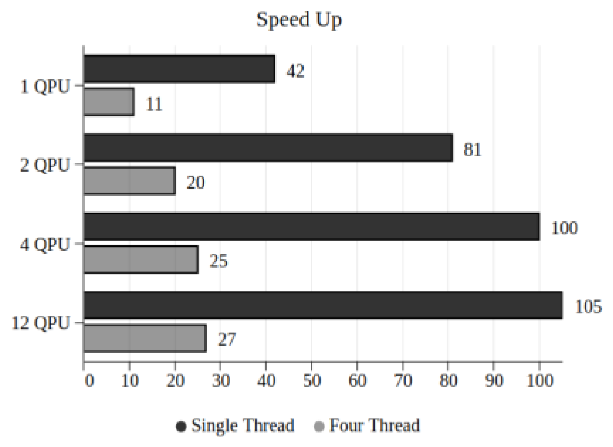


Figure 7. Speed Up Factor Against Single And Four Core CPU (Higher is better)

In Figure 8, energy consumptions of both CPU and GPU implementations are given. As can be seen from Figure 8, the energy consumption of the proposed GPU implementation is 35 times less than the CPU implementation. Such a huge energy consumption difference plays an important role in saving battery life in embedded applications, especially for mobile devices.

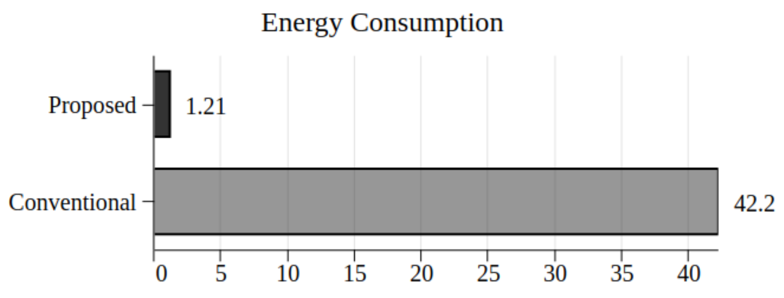


Figure 8. Energy Consumption Of Conventional And Proposed Algorithms (Lower is better)

5. Conclusion

In this paper, we have proposed a parallel processing approach to increase the execution speed of 2D convolution problem and decrease the energy consumption of hardware for embedded and mobile robotic applications, based on VideoCore IV graphic processor used on Raspberry Pi 3 single-board computer. Our approach is using vector operations instead of scalar operations to use SIMD machines (QPUs) in the GPU. In our work, we used a Raspberry Pi board based on a Broadcom System-on-Chip but our approach can be implemented on a similar System-on-Chip that has video accelerators. As seen in Figure 7, our proposed implementation is 105 times faster than the single-threaded implementation and 27 times faster than the four-threaded implementation. Also as seen in Figure 8, our implementation is 35 times more electrically efficient than the most efficient result of CPU implementation. On the other hand, the conventional CPU algorithm consumes all the processor resources on the machine while our algorithm requires under 10% of the total CPU utilization. In this way, the CPU can work on other tasks without any lag.

References

[1] Shin et al.(2013). 28nm High-k Metal-Gate Heterogeneous Quad-Core CPUs for High-Performance and Energy-Efficient Mobile Application Processor. *In 2013 International SoC Design Conference (ISOCC)*, Busan, Korea, 198-201.

[2] Hsu, C. H., Kremer, U., Compiler-Directed Dynamic Voltage and Frequency Scaling for CPU Power and Energy Reduction, Ph. D. thesis, Rutgers University, New Jersey, USA, 2003

[3] Hennessy, J. L., Patterson, D. A., *Computer Architecture: A Quantitative Approach*, Elsevier, Waltham, USA, 2011.

[4] Chhugani et al. (2008). Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture. *Proceedings of the VLDB Endowment*, Auckland, New Zealand, 1313-1324.

[5] Blake et al. (2009). A Survey of Multicore Processors. *IEEE Signal Processing Magazine*, 26(6), 26-37.

[6] Ou et al. (2012). Energy-and cost-efficiency analysis of arm-based clusters. *In 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Ottawa, Canada, 115-123.

[7] Yang et al. (2008). Parallel Image Processing Based on CUDA. *International Conference on Computer Science and Software Engineering*, Wuhan, China, 198-201.

[8] Gelke et al. (2016). Using mobile processors for general purpose industrial signal processing. *In Embedded World Exhibition and Conference*, Nuremberg, Germany.

[9] Peroni et al. (2019). ARGA: Approximate reuse for GPGPU acceleration. *In 2019 56th ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, NV, USA, (1-6).

[10] Hecht, V., Ronner, K. (1991). An Advanced Programmable 2D-Convolution Chip for, Real Time Image Processing. *In 1991., IEEE International Sympoisum on Circuits and Systems*, Singapore, Singapore, 1897–1900.

[11] ***, VideoCore® IV 3D Architecture Reference Guide, https://usermanual.wiki/Pdf/VideoCoreC2AE20IV203D20Architecture20Reference20Guide.1278100948/view

[12] Cypher, R., Sanz, J. L. (1989). SIMD architectures and algorithms for image processing and computer vision. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12), 2158-2174.

[13] Moreland et al. (2011). Dax toolkit: A proposed framework for data analysis and visualization at extreme scale. *In 2011 IEEE Symposium on Large Data Analysis and Visualization*, Providence, RI, USA, 97–104.

[14] Buttlar et al. *PThreads Programming: a POSIX Standard for better multiprocessing*, O'Reilly Media, Inc, Sebastopol, CA, USA, 1996.