Implementation of Image Processing Algorithms on the Graphics Processing Units

Natalia Papulovskaya, Kirill Breslavskiy, and Valentin Kashitsin

Department of Information Technologies of the Ural Federal University,
620002, 19 Mira street, Ekaterinburg, Russia
pani28@yandex.ru,

Abstract. The paper describes features of the multithreaded algorithms implementation on contemporary CPU and GPU. The features of access of a graphics processing unit (GPU) memory are reviewed. The bottlenecks have been identified, in which there is a loss of speed in image processing. Recommendations are made for optimization of algorithms for processing image of various size. Examples of implementation of the algorithms are given in the software and hardware architecture CUDA, which is well suited for a wide range of applications with high parallelism.

Keywords: Image processing, Graphics processing unit, Parallel computing, CUDA

1 Introduction

Restricted computing power is one of the problems for solving many tasks. Especially, it appears in processing huge arrays or real-time problems. Processor performance growth of processors has always been associated with increasing the frequency and number of transistors. However, according to the Moore's Law [1], the lost of power is because the laws of physics cannot be changed. Namely, increasing the frequency and amount of transistors can not be infinite. Recent ten years have been devoted to search for other solution: despite projected increase in frequency, there were optimizations of structure and growth of cores quantity. But cores quantity growth is not an absolute solution. New opportunities arise for parallel computing after introducing General Purpose Computing on Graphics Processing Units (GPGPU) [2] and Compute Unified Device Architecture (CUDA) by NVIDIA company [3,4,5].

Usually, the Central Processing Unit (CPU) has only 4 or 8 computing threads. But the Graphics Processing Unit (GPU) has hundreds and thousands computing threads. It provides significant acceleration for algorithms with high parallelism degree. Nevertheless, sometimes the CPU wins in competition with the GPU on well-paralleled tasks.

Mostly, image processing algorithms are easy for parallelization. However, under certain conditions, an algorithm implementation on the central processing unit (CPU) is faster. For proper use of GPU, it is necessary to identity its bottlenecks and describe capabilities of computing resources in tasks of image processing and analysis.

2 Differences between the CPU and GPU

Any program running on CPU uses the operating memory, processor cache, and processor itself. Being run on GPU, the working process of a program is more complex: the algorithm and data should be previously uploaded to GPU before use. The majority of graphics adapters are implemented with the PCI-Express bus, which has limited bandwidth. This is one of the reasons why the central processing unit can perform calculations faster. Obviously, the value of the data and memory uploading time is the crucial parameter for calculation speed.

The CUDA memory model includes the host DRAM memory (regular operating memory), device DRAM (graphics memory), and registered and shared memory that are located on every multiprocessor of the graphics chip. The CUDA threads have access to all kinds of the GPU memory, which differ by scope, speed, and size (Table 1). The registered memory is used for local variables. Other variables that exceed the registered memory are placed in the local memory, which is suited outside of the chip and has low access speed. The shared memory is used for interaction between the threads. All threads have read/write access to the global memory. The scope for global memory is the whole application, and contents of the global memory doesn't change while starting different cores. Moreover, the central processor has also access to the global memory; this is why the global memory is used for data exchange between the CPU and GPU.

Types of memory available for the CUDA applications

Type	Scope	Speed	Size	Applying
Registered	Thread	High	16384 registers per SM	Local variables
Local	Thread	Low	Up to global memory size	Local variables, exceeding registered memory
Shared	Block	High	16 Kb per SM	Threads inter operation
Global	Application	Low	Up to 4 Gb	Data storage with CPU

Thus, the most data operations and exchange with the CPU are implemented by the global memory, which has low speed, as seen from Table 1. The memory delay problem in the CPU is solved by caching and code predicting. But the GPU goes in the other way: in the case of waiting the data access in some thread, the video chip attaches to another thread that has already got all necessary data. Moreover, the video memory mostly has wider bandwidth than the common memory. Instead of active cache memory used by the CPU, the graphics cards have only 128-256 Kb of the cache memory, which is applied to bandwidth extension and of reduction delays.

Multithreading is supported in the GPU on the hardware level. Vendors achieve instant switching between threads (by 1 clock cycle) and support up to 1024 threads on every core.

Large number of threads in the CPU is not advisable due to significant time loss while switching, which may consume up to several hundreds of clock-cycles. In addition, one CPU core is able to process only 1 or 2 threads simultaneously.

3 Benchmarks

3.1 Baseline

Performance tests were done on image processing algorithms software implementation. Four different resolutions of image (50x50, 150x150, 300x300, and 512x512) were used with the quality of JPEG Quality 25 and JPEG Quality 100. As a reference image, the well-known file Lena.jpg [6], was used. Lenna or Lena is the name given to a standard test image widely used in the field of image processing since 1973.

The images were processed by application of Sarcis [7] developed by the Department of Information Technologies of the Ural Federal University. This program is designed for image processing of color and gray scale images and can be used for calculation both on the CPU and GPU. For the tests, four algorithms were chosen: the weighted linear filter, reverse gradient filter, edge detection Canny's filter, and morphology processing filter erosion. An example of application of the Canny's filter is shown in Fig.1. Workstation configurations

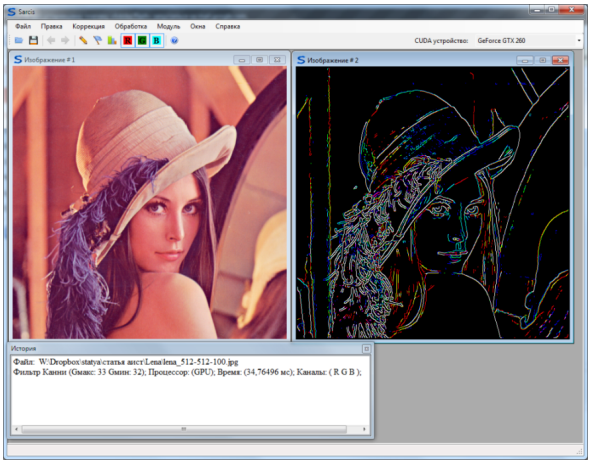


Fig. 1. Image processing by the Sarcis program

used for benchmarks:

CPU Intel Core i5-2450 (4 threads, 2.5Ghz), GPU NVIDIA GT630M (96 CUDA cores, 800Mhz);

CPU Intel Core i7-860 (8 threads, 2.8Ghz), GPU NVIDIA GTX260 (216 CUDA cores, 576Mhz);
CPU Intel Core i5-3470 (4 threads, 3.2Ghz), GPU NVIDIA GTX760Ti (1344 CUDA cores, 915Mhz).

3.2 Theoretical description of filtering algorithms

Weighted linear filtering. The linear smoothing filters are good ones for removing the Gaussian noise and, also, the other types of noise. A linear filter is implemented using the weighted sum of the pixels in the successive windows. Typically, the same pattern of weights is used in each window; this means that the linear filter is spatially invariant and can be implemented using a convolution mask. If different filter weights are used for different parts of the image (but the filter is still implemented as a weighted sum), then the linear filter is spatially varied [8].

One of the simplest linear filters is implemented by a local averaging operation where the value of each pixel is replaced by the average of all the values in the local neighborhood. The weighted linear filtering mask is given by the mask $M = \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$. The main strategy is to set the largest weight at the central pixel and inversely proportional to distance values to other pixels:

Reverse gradient filter. The idea of the reverse gradient filter is in choosing the mask weights. The greater bright difference between the central and the next point, the less weight is prescribed to the next point.

The mean gradient module value calculated for each aperture brightness level is:

$$H^G(l) = \frac{1}{h(l)} \sum_{m=1}^{h(l)} G_l^m(i,j)$$

Where $H^G(l)$ is the mean brightness gradient function value for l; $h(l)$ is the number of points with brightness l; $G_l(i,j)$ is the gradient value in the mth point with brightness l at position (i,j). The Laplace operator $\bigtriangledown^2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ is used to calculate the gradient value

Canny filter. The Canny filter is considered to be the optimal choice for any task, which requires edge extraction or edge enhancement of elements with different shapes and characteristics. This is due to its easy implementation and ability to be adapted to different circumstances [9].

There are the following three criteria for the algorithm:

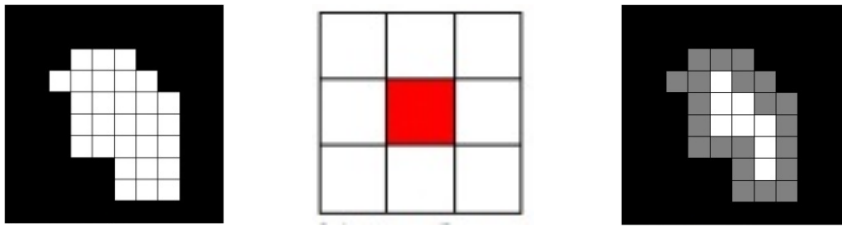
- smart detection (increasing the signal/noise ratio);
- good localization (correct definition of boundary position);
- only one response to the border.

The Canny filter differs from other classic edge detecting filters because it considers directionality of the element and permits personalization of the filter behavior by choosing the value of parameters of the algorithm. This provides changes in direction of the filter polarization and sensibility of the edge extraction.

The algorithm runs in 5 separate steps [10].

1. Smoothing: blurring of the image to remove noise.
2. Finding gradients: the edges should be marked where the gradients of the image have large magnitudes.
3. Non-maximum suppression: only local maxima should be marked as edges.
4. Using the double thresholds: potential edges are determined by using the thresholds.
5. Edge tracking by hysteresis: the final edges are determined by suppressing all edges that are not connected to a strongly determined edge.

Morphological Image Processing. The morphological image processing is a collection of non-linear operations related to the shape or morphology the image features. Erosion of images is generally used for getting rid of the image of random insertions. The idea is that the inclusions are eliminated by blurring, while large and, thus, more visually important regions remain.



a) b) c)

Fig. 2. Morphological image processing example; a) source image; b) kernel; c) erosion result

Erosion (morphological narrowing) is a convolution of the image or its area selected with some kernel. The kernel may be of arbitrary shape and size. Here, in calculating the convolution, only one leading position is selected in core, which is aligned with the current pixel. In many cases, the core is selected as a square or circle with a leading position at the center (Fig. 2b). The core can be regarded as a pattern or a mask.

Applications of dilation is reduced to the passage pattern throughout the image and use some operator to search for a local minimum intensities of the pixels in the image, which covers the template. The gray color filled pixels appear black due to erosion (Fig. 2c).

3.3 Benchmark results

Weighted linear filtering. As seen from graphs in Fig. 3, the images with size 50×50 are processed for the same time by both the CPU and GPU, excluding NVIDIA GTX260, which works a little better. The larger images are processed faster by the GPU.

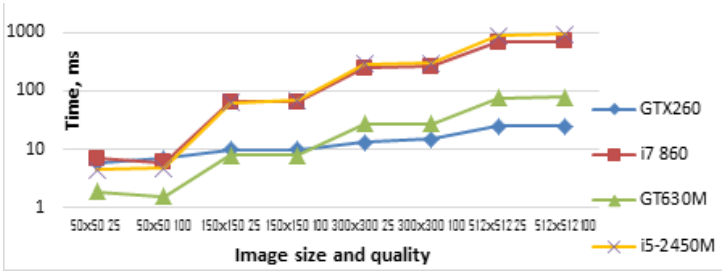


Fig. 3. Weighted linear filter benchmarks

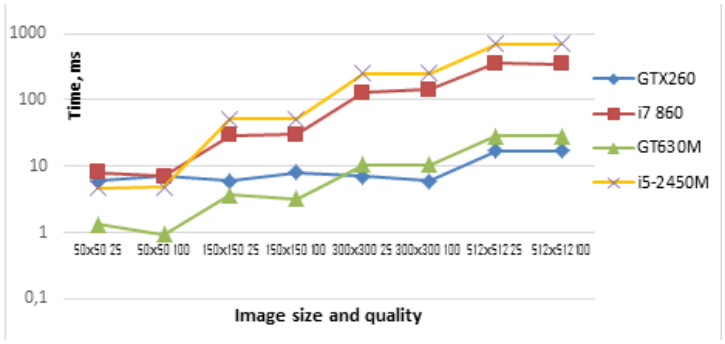


Fig. 4. Reverse gradient filter benchmarks

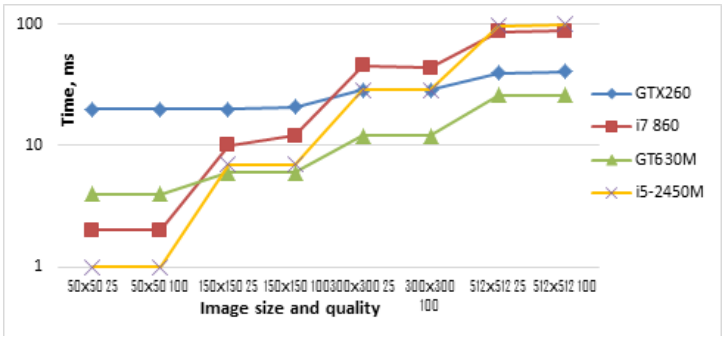


Fig. 5. Canny filter benchmarks

Reverse gradient filter. As seen from graphs in Fig. 4, application of the reverse gradient filter gives comparable results w.r.t. the weighted linear filter.

Canny filter. It is obvious from Fig. 5 that the CPU runs faster on images up to 150×150 quality 25, but the GPU NVIDIA GT630M takes advantage in times on larger images. The GPU NVIDIA GTX260 works better when the CPU starts from images 300×300 regardless of image quality.

Morphological Image Processing. Analysis of graphs in Fig. 6 shows that in using the morphological processing similar to Canny's filter, the GPU NVIDIA GT630M starts process faster than CPU from images size 50×50; and the GPU NVIDIA GTX260 is faster from size 300×300.

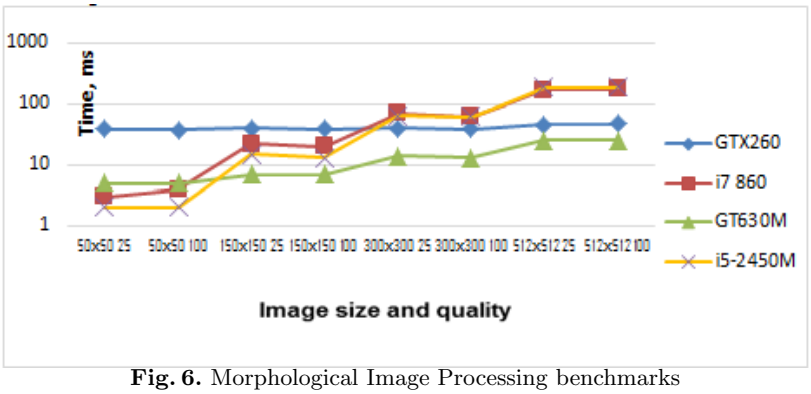


Fig. 6. Morphological Image Processing benchmarks

4 Conclusion

Results of this research show a rise in the efficiency of the GPU computing while image size grows. Nowadays, the images with resolution less than 1024×768 can be found rarely. From resolutions 300×300, the GPU starts to process images faster than the CPU; so, it is advisable to use image processing algorithms applying the GPU computing. The CPU computing is suitable in small image processing or hardware systems without the GPU.

References

1. Moore's Law, `http://www.mooreslaw.org/` csl783/canny.pdf (2009)
2. GPGPU. General-Purpose Computation on Graphics Hardware, `http://gpgpu.org`.
3. Halfhill, T. R.: Parallel processing with CUDA. Micriprocessor report, `http://www.nvidia.com/docs/IO/55972/220401-Reprint.pdf` (2008)
4. NVIDIA. CUDA in Action, `http://www.nvidia.com/object/cuda-home-new.html`
5. NVIDIA CUDA: Compute Unified Device Architecture, NVIDIA Corp. (2007).
6. The Lenna Story, `http://www.cs.cmu.edu/~chuck/lennapg/lenna.shtml`
7. Dorosinsky, L.G., Kruglov, V.N., Papulovskaya, N.V., Chiryshev, A.V.: CUDA technology in digital image processing, Ekateringurg, 192 p.,(2011)

8. Jain, R., Kasturi, R., Schunck, B.G.: Machine vision, http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision-Chapter4.pdf, 118-122
9. Pirotti, F., Vettore, A.: Canny filter applications and implementation with grass, http://www.academia.edu/313972/
10. Canny Edge Detection, http://www.cse.iitd.ernet.in/~pkalra/