Image Processing Methods Optimization by Means of GPU Computing

SKORPIL, V.*, ZIDEK, K.**, KOUBEK, T.**, LANDA, J.**, ENDRLE, P.*
*Faculty of Electrical Engineering and Communication
Brno University of Technology,
Purkynova 118, 612 00 Brno,
**Faculty of Business and Economics
Mendel University in Brno
Zemedelska 1, 61300, Brno
CZECH REPUBLIC
skorpil@feec.vutbr.cz, tomas.koubek@mendelu.cz, xlanda@node.mendelu.cz,
xendrl00@stud.feec.vutbr.cz, http:/www.vutbr.cz/, http:/www.mendelu.cz/

Abstract:- Optimization of computer resources allocation is crucial for real-time image processing applications. One of the optimization possibilities is the use of GPU (Graphics Processing Unit) instead of CPU for computing. The article explains augmented reality application (AuRel) and the possibility of speeding-up the image preprocessing phase. Further, two image processing algorithms – threshold and convolution are tested. These algorithms are tested in various implementations from naïve code performed on CPU to optimized code running on GPU.

Key-Words:- Image processing, GPGPU, CUDA, OpenCL, Augmented Reality

1 Introduction

Image processing is widely used in many applications from industry and medical research to consumer and entertainment electrotechnics such as mobile phones and tablets. Depending on the application, the image processing can be a lengthy process. Appropriate optimization is needed to shorten the process, particularly in case of real time applications [10], [12].

One of the optimization possibilities is the use of GPU (Graphics Processing Unit) instead of CPU for computing. Actually last generation of mobile and desktop graphic cards are powered by graphics chips that provide sufficient performance capabilities. Because of the technologies such as NVidia CUDA and OpenCL, it is possible to transfer image processing calculations from CPU to GPU. This transfer provides better performance capabilities, especially for real time applications.

Firstly, the augmented reality (AR) application AuRel is presented. This application uses a specific image processing algorithms. These algorithms are discussed in relationship to how much computing power they require. Secondly, the use of GPU is discussed with two most common frameworks, CUDA and OpenCL. Finally, two common image processing algorithms are tested using different CPU and GPU implementations. These algorithms are the basis of AuRel image preprocessing phase.

2 AR Application

One of the uses of image processing is in Augmented Reality [1]. Augmented reality is widely used in many applications; one of them is augmented prototyping. In augmented reality prototyping tool AuRel, more described [1] and [2], specific image preprocessing in marker recognition phase [14] is used.

AuRel is developed in C++ and OpenCV library [3] that implements various computer vision algorithms. Our application is based on [4] and uses morphology operations to detect markers. The first step is the camera image acquisition. A VGA web camera with 640x480 pixels resolution is usually used, but the application can use any USB camera or video file with high resolution as a source. The next step is the gray-scale conversion and image smoothing by Gaussian blur for noise and details removal.

The following step is image adaptive thresholding. Based on blurred image, it creates the thresholded value for each pixel. Finally it detects contours, from contours derives polygons and tries to identify potential markers by searching for quadrangles. These potential markers are then identified with the use of Golay error correction code. If the marker is identified, the transformation

ISBN: 978-1-61804-109-8

matrix is computed and the appropriate 3D model is displayed at the marker location.

In case of VGA camera usage (640x480 pixels resolution), the application runs in almost real time, but when using higher resolution, the time needed to detect and identify marker rises. Image preprocessing is area with one of the highest computational needs. These needs are relatively constant in time, for VGA and HD (1280x720 pixels) resolution take approx. 30 % of processor time. For FullHD (1920x1080 pixels) resolution this value grows, it is 40 % of processor time. However, all demands grow for high resolution images, so saving of resources is requested.

As described earlier, the image preprocessing uses convolution (in case of blurring) and also the threshold is used, one in case of adaptive threshold and also when using a Golay error correction code. These two operations have high computational demands. Because of this, we decided to test the capability of speeding these processes by using of GPGPU.

3 General-purpose computing on GPU

The idea of General-purpose computing on graphics processing units (GPGPU) is to use GPU not only for rendering, but also to use the potential of the parallel computing architecture for general computations. The goal is to transfer calculations usually done on CPU to GPU.

The GPU architecture was designed for rasterization and later for specific graphic operations (3D graphics pipeline). But with the rise of more demanding graphic effects, the architecture was extended by programmable pixel and shader units. Even though these units are primarily for image data, they are widely used for general computations on GPU. Nowadays, two platforms for GPGPU - CUDA and OpenCL are massively used.

GPGPU are used in many applications. [5] describes a framework for implementation of hyperspectral image processing algorithms which takes advantage of multiple levels of parallelism found in modern GPU. They show that using GPU is superior to using common CPU or high performance clusters.

The reference [6] uses GPU to speed up the registration process of 3D medical images using commodity GPU. They show 50 times the improvement over the standard CPU implementation. Because of this enhancement, the process is almost real-time. Nevertheless, they clearly state that to use GPU, one must be prepared to rethink existing methods and adapt them to parallel processing environment.

Another use of GPU shows [7] and [8]. They use GPU to increase the efficiency of well-known Lucas-Kanade registration algorithm. Their implementation shows that using GPU increases performance significantly.

Although the GPU is mostly used for image processing, the examples of different applications can be found, e.g. KinectFusion [9]. KinectFusion is a system for surface reconstruction that uses only Microsoft Kinect and any commodity graphics card. They present a novel GPU pipeline that allows accurate camera tracking and surface reconstruction in interactive real-time rates.

The ability to use GPU is not limited to consumer or professional computers. Transferring computations from CPU to GPU is also used in mobile devices. [11] shows a GPU-accelerated face annotation system for smartphones. They show that the performance can be increased significantly (4x) making the response time within the real-time requirements. More applications of GPGPU can be found in [13].

3.1 CUDA

CUDA (Compute Unified Device Architecture) is an architecture developed by California company NVidia. Every new graphic card from NVidia contains given number of CUDA cores. These cores replaced the need for separate pixel and vertex shaders and are able to perform necessary computations regardless of the operation type. Through CUDA, the programmer has the ability to use the potential of all available CUDA cores.

Because the CUDA architecture is available only for NVidia graphic cards, any code for CUDA can be used only with NVidia card. CUDA is supported by all graphic cards based on G80 architecture or higher. Nevertheless, it is clear that professional Quadro card will have better results than basic consumer GeForce card. As the development of new graphic cards progresses, also the development of CUDA goes forward and even though there is a back compatibility between versions, the new functions are not supported with old cards.

ISBN: 978-1-61804-109-8

3.2 OpenCL

OpenCL (Open Computing Language) is a standard extension of C and C++ of data types, data structures and functions [15] that facilitates parallel programing not only for GPU, but also for common CPU. OpenCL was created by Apple for the purpose of creating unified interface for hardware programing.

4 Testing

There are two methods tested in this article, threshold and convolution. Threshold is tested on several different implementations of optimized and unoptimized code and convolution is tested on CUDA and OpenCL. These two algorithms represent part of AuRel image preprocessing phase. The preprocessed image is then used for detecting markers in AuRel.

The testing was performed on HP8710p with NVidia Quadro 320M. As a operating system was used Ubuntu 11.10 with a proprietal driver NVidia 280.13.

4.1 Threshold

Thresholding is a basic image processing operation and can be performed on any type of image. Because of the simplicity of the thresholding algorithm, it is very easy to compare it in different implementations. The threshold was tested because threshold and other operations of this kind are key parts of the application.

4.1.1 CPU

Threshold implementation, with only a use of CPU, demonstrates the common approach used to deal with thresholding without any optimization. To simulate the naive code implementation, OpenCV library was used. Camera image is stored and processed in cv::Mat data type. The algorithm accesses individual image pixel and computes the thresholded value.

4.1.2 OpenCV

The second test was performed using standard functions implemented in OpenCV. Because OpenCV is primarily used for image processing, the algorithms are optimized. Threshold is performed using cv::Threshold method. OpenCV is mostly used for computations on CPU, but allows user to use many functions on both CPU and GPU. The need to load image data into graphic card for GPU is the significant difference between CPU and GPU implementations. After being processed, the data must be copied back to host device. The need for image transfers from and to GPU is a key part of any GPU implementation. Because of this, the data type used for storing image data is different from basic CPU version. Determining the computational speed of GPU implementation is done in the third test.

4.1.3 CUDA and OpenCL

The implementation on CUDA platform is more difficult than previously discussed examples. The application contains the code for graphic card initialization and implementation of method that sends data to graphic chip, processed data are sent back and, very important, launches CUDA Kernel. Kernel is a code that species activities performed on graphic card with supplied data. Our thresholding kernel is below.

```
#include ”kernel cuda .h” // CUDA kernel
__global__ void gpukernel(int rows, int cols , unsigned char∗ src data , unsigned char∗ dst data , int threshold){
  int n = blockIdx.x ∗ blockDim.x + threadIdx.x;
  if (n > cols∗rows) return ;
  if (src data[n] > threshold)
    dst data[n] = 255;
  else dst data [n] = 0;
}
```

Inside the kernel, first the position inside the array of input value is computed and checked if the position points to specific data inside the input array. After that the thresholding is performed and the output is stored inside output data array. The graphic card memory allocation is more complex then when using the basic functions implemented in OpenCV.

OpenCL implementation is similar to CUDA. The key principal, the transfer of image data to and from graphic card, is analogous. The kernel itself is not different from CUDA (except the header and data types), however, the use of kernel is different. The main difference between CUDA and OpenCL is that OpenCL is defined for use on different hardware platforms, whereas CUDA is limited only to Nvidia graphic cards. Because of this, OpenCL must firstly detect the platform on which the kernel code is being used. Platform detection is a complex problem because the large spectrum of possible platforms must be taken into consideration.

Therefore poorly written implementation can result in coding the possibility of code functioning only on specific type of devices.

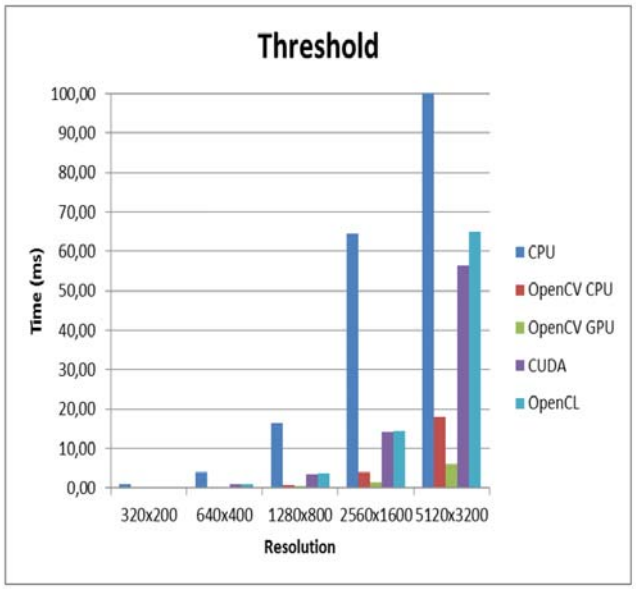

Fig.1: Duration of threshold algorithm performing in various implementation (time in ms)

4.1.4 Threshold testing results

The results clearly show the ineffectivity of the naive code implementation on CPU. When comparing the classic CPU implementation with OpenCV CPU implementation, the OpenCV version is clearly superior. This shows that the optimization of algorithms can make very large difference even when using only the CPU and not GPU. Even when using very large images, the OpenCV CPU implementation performs better then CUDA and OpenCL.

Table 1: Duration of threshold algorithm performance in various implementations (ms)

Resolution	CPU	OpenCV		CUDA	OpenCL
		CPU	GPU		
320x200	1,11	0,05	0,07	0,25	0,25
640x400	4,14	0,30	0,12	0,89	0,93
1280x800	16,42	0,85	0,39	3,52	3,67
2560x1600	64,58	4,08	1,50	14,31	14,58
5120x3200	253,72	18,15	6,00	56,45	65,06

The best results were measured with OpenCV GPU. With small images, the difference is negligible, but with larger images, the difference is in 10x better then with CUDA and OpenCL.

4.2 Convolution

To further test both CUDA and OpenCL platforms, the implementation of basic image processing algorithms was tested. The implementation of convolution with 5x5 matrix was chosen. Convolution enables the use of many image processing algorithms, e.g. Gaussian Blur. The key principles of working with both platforms remain unchanged; however the number of input parameters has increased. Besides the source image, the matrix representing convolution kernel was added.

Table 2: Duration of convolution algorithm performance in various implementations (ms)

Resolution	CUDA	OpenCL
320x200	23,68	23,69
640x400	94,11	94,02
1280x800	375,93	376,39
2560x1600	1508,90	1512,30
5120x3200	6061,98	6053,82

As results demonstrate the times needed to perform convolution are almost same. This means that the difference in performance time of both platforms is negligible. The slight differences can be credited to the different approaches used in performance measuring.

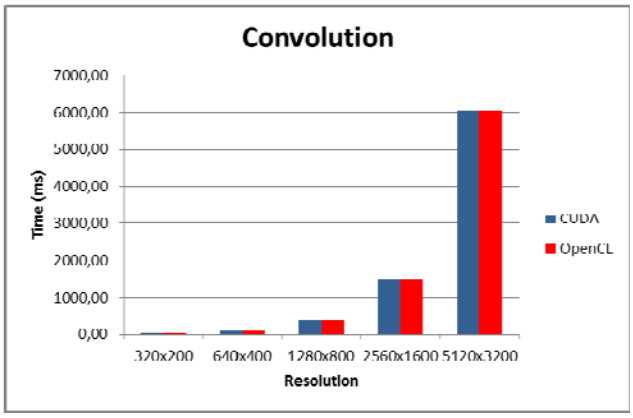

Fig.2: Duration of convolution algorithm performance in various implementations (ms)

ISBN: 978-1-61804-109-8

5 Conclusion

The test of different CPU and GPU implementations clearly shows that the transfer from CPU to GPU significantly reduces the time needed for operations. For augmented reality and other real-time applications, the use GPU acceleration is possible solution in a relationship to rising image resolution requirements. It is important to be able to use image in high resolution (1920x1080), which is not yet possible in our application. Nevertheless, according to our results, even processing speed of low resolution images in real-time applications (e.g. 640x480) is substantially improved by usage of GPU. Therefore, the GPU acceleration is not suitable only for processing of huge amounts of data.

A substantial advantage of the OpenCL technology is its support by mobile GPUs in tablets and smartphones. Especially these devices must deal with limited resources and must be also effectively designed with regard to the limited battery capacity.
The results also show that simple transfer from CPU to GPU provides only a partial computation speed-up. Other factor in computation speed-up is the code optimization. Comparison is shown in Table 1, where optimized code, represented by OpenCV library, reaches better result than naive code, regardless if it is performance on GPU or CPU. Accordingly, besides hardware enhancement of image processing method, there must be an algorithm and code optimization for results improvement.

Acknowledgment:
This work has been supported by the grants:
MSM 0021630529 Intelligent systems in automation (Research of Brno University of Technology), FSI-S-11-31 Application of artificial intelligence methods (Research design of Brno University of Technology) and MSM 6215648904/03 (Research design of Mendel University in Brno), OPVK No CZ.1.07/2.2.00/28.0062, Joint activities of BUT and TUO while creating the content of accredited technical courses in ICT, No. ED 2.1.00/03.0072, Centre of sensor information and communication systems (SIX) and No. FEKT-S-11-15 Research of electronic communication systems.

References:

[1] Prochazka, D. Stencl, M. Popelka, O. Stastny, J., Mobile Augmented Reality Applications. *Mendel 2011: 17th International Conference on Soft Computing. Brno University of Technology*, 2011, pp. 469-476.

[2] Stastny, J., Prochazka, D., Koubek, T., Landa, J., Augmented reality usage for prototyping speed up, *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*. 2011. VOL. LIX, No. 2, pp. 353--359.

[3] Laganiere, R.: *OpenCV 2 Computer Vision Application Programming Cookbook.* Packt Publishing, 2011.

[4] Kato, H. Tachibana, K. Billinghurst, M. and Grafe, M. A registration method based on texture tracking using ARToolKit, *In Augmented Reality Toolkit Workshop*, pp. 77-85, 2003.

[5] Setoain, J. et al. Parallel Hyperspectral Image Processing on Commodity Graphics Hardware, *International Conference Workshops on Parallel Processing*, 2006, pp. 465-472.

[6] Shams, R. et al. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images, *Computer Methods and Programs in Biomedicine*, Vol. 99, 2010, pp. 133-146.

[7] Duvenhage, B., Delport, J. P., Villiers, J. Implementation of the Lucas-Kanade image registration algorithm on a GPU for 3D computational platform stabilization, *7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pp. 83-90.

[8] Marzat, J., Dumortier, Y., Ducrot, A. Real-Time Dense and Accurate Parallel Optical Flow using, *WSCG 2009*, 2009, pp. 105-111.

[9] Izadi, S. et al. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera, *ACM Symposium on User Interface Software and Technology*, 2011, pp. 559-568.

[10] Stastny, J., Skorpil, V. New Methods for Face Recognition. *TSP 2010: 33rd International Conference on Telecommunication and Signal Processing.* Vienna, Austria, 2010, pp. 156-159. ISBN 978-963-88981-0-4.

[11] Wang, Y., Pang, S., Cheng, K. A GPU-accelerated face annotation system for smartphones, *International conference on Multimedia*, 2010, pp. 1667-1668.

[12] Minarik, M., Stastny, J. Recognition of Randomly Deformed Objects. *MENDEL 2008: 14th International Conference on Soft Computing.* Brno University of Technology, 2008, pp. 275-280. ISBN 978-80-214-3675-6.

[13] Owens, J. D. et al. A Survey of General-Purpose Computation on Graphics Hardware, *Eurographics 2005, State of the Art Reports*, 2005, pp. 21-51.

[14] Skorpil, V., Stastny, J. Comparison Methods for Object Recognition. *Proceedings of the 13th WSEAS International Conference on Systems.* Rhodos, Greece, 2009. pp. 607-610. ISBN 978-960-474-097- 0.

[15] Scarpino, M. OpenCL in action: how to accelerate graphics and computation. Shelter Island: Manning, 2012, 434 s. ISBN 978-1-617290-17-6.