Tomasz MARCINIAK, Sławomir BUJNOWSKI, Zbigniew LUTOWSKI, Dariusz BOROŃSKI
University of Technology and Life Sciences, Bydgoszcz, Poland
Tomasz GIESKO
Institute for Sustainable Technologies – National Research Institute, Radom, Poland

DIGITAL IMAGE CORRELATION – UNIVERSAL TOOLS VERSUS CUSTOM SOLUTIONS

Key words

Correlation methods, DIC, GPU, CUDA, multithreading.

Summary

The development of optoelectronics and increasing the processing speed of processors encourage the use of different measurement methods with the use of data obtained from the digital cameras. The article presents the test results of the time consumption of the normalised 2D-correlation function. The correlation is one of most commonly used functions in image processing. It searches for a correlation between the two images and requires quite a lot of calculation operations. The speed of this function was examined in a variety of tools, such as MatLab, MatLab with tools Parallel Computing, library OpenCV and OpenCV wer.2.2 with the CUDA platform extension. During tests, the speeds of image processing were determined, especially for images acquired with high-resolution cameras. Conclusions presented concern with application possibilities of correlation methods, including multithreaded processing.

Introduction

Fast development of optoelectronics has resulted in more widespread use of optical measurement methods. The basic measurement system based on these methods contains a digital camera and a digital image processing block. High resolutions of digital cameras enable one to increase the measurement accuracy. This results in transmission of more data and growing requirements for calculation capacity. Aiming at on-line camera image processing is a challenge for software designers; therefore, the choice of a calculation platform optimal in terms of efficiency is one of the key element of the designed system. The most popular platforms are as follows:

– signal processors (DSP),
– programmable systems FPGA and VLSI,
– CPU – calculation with the use of a standard processor,
– GPU – calculation with the use of graphic cards processors.

VLSI systems are a technology that enables the building of the most efficient systems [1]; however, this technology is the most expensive and the least available. It involves designing an integrated circuit capable of performing assigned tasks. Additionally, the very small scalability of such a solution is the drawback of this technology.

Application of FPGA systems ensures the highest of all possible calculation rates [2, 4]. However, this calculation platform has its disadvantages, including the high costs of FPGA circuits with appropriate resources (enabling implementation of suggested algorithms), the costs of special software and a complex method of the realisation of interfaces used for outside communication (gathering data and visualisation of calculation results).

The DSP processor is a solution that is better adjusted to floating point calculation [3]. However, like in the case of FPGA circuits, they are characterised by a relatively high solution cost, including a full set of interfaces.

In the stage of the device prototyping, the designers often choose to realise the calculation platform based on PC computer. This is justified by a relatively low cost of the prototype realisation, the easy application of modifications, the visualisation of results, and the possibility of monitoring transitional states of realised algorithms and the wide variety of tools available for image processing algorithm execution.

In the article, the most popular and available, for PC users, methods for image processing calculation execution have been compared. The comparison includes the use of Matlab and OpenCV, CUDA, IPP, and TBB libraries.

1. Libraries and technologies

1.1. Computer Unified Device Architecture (CUDA)

One of the popular ways to improve the performance of complex calculation tasks is their division into sub-tasks which, in turn, are solved using many nodes yielding partial results. Based on these results, the final result is calculated. Such an approach is used both for purely hardware solutions, that is, calculations performed with the use of FPGA circuits, and for software solutions, when the CPU possesses more than one core. However, the architecture of a traditional processor (CPU) optimised for data processing is not necessarily the architecture ideal for performing calculation. This problem was first noticed in the case of graphic processing units, which initiated a new branch of processors – graphic processors (GPU). While creating their architecture, the fact that the recurrence of performed operations is a characteristic feature of computer graphics was taken into consideration. In this way, GPUs were oriented to simultaneous and very effective performance of the calculation. In 2007, the Nvidia company has provided technology enabling the creation of calculation software, at the level of C language, which is performed directly according to GPU – CUDA technology.

1.2. Intel Integrated Performance Primitives (IPP)

Intel Integrated Performance Primitives is a library which provides the possibility of creating multimedia applications. Thanks to functions implemented in it, it is possible to have access to additional instructions on processors (e.g. MMX technology, Streaming SIMD Extension – SSE, Streaming SIMD Extensions 2-SSE2), aiding image and sound processing, such as, encoding and decoding JPG files and the encoding of MPEG4 and H263, as well as arithmetic, statistic functions and in the range of signal and image processing (DSP) [5]. The library handles many types of data providing flexibility for the creation of multimedia applications.

The library main features are as follows:
– multithreaded processing,
– functions optimised for operation on multi-core processors,
– exemplary source codes optimised for operation with multi-core processors,
– thread-safe processing,
– improved encoding of samples of vision sequences,
– improved encoding of sound samples,
– availability for Windows, Linux, and, Macintosh platforms.

1.3. Intel Threading Building Blocks (TBB)

Intel Threading Building Blocks is a perfect supplement to the IPP library. It allows improving the application efficiency, and from the point of view of a programmer, to facilitate the creation of the program multithreaded structures through the use of parallel processing predefined structures. It contains parallel algorithms and co-divided structures that eliminate parallel, multithreaded programming at the lowest level. The set of procedures included in the library provides the basis for multithreaded structures, thanks to which the created application can fully utilise multi-core processors [6]. The programmer can apply automatic techniques for calculation core load balancing. Moreover, such an application can be easily moved between Windows, Mac, and Linux platforms. It is compatible with applications created both on 32-bit and 64-bit processors, supporting multithreaded technologies Win32, POSIX, and Open MP. Being compatible with many numerous compilers is also of great benefit, these include Intel, Microsoft, and GNU. Another important feature is the library scalability that provides capability of being applied for operation on computer systems with different numbers of processors.

1.4. Open CV

OpenCV provides the user with efficient operation in the field of image digital processing, putting special emphasis on on-line operation. It provides tools, so far available only in specialist research laboratories, supporting the accomplishment of projects and research connected with digital processing of images.

2. Test results

Tests were performed on the object being a flexible membrane bearing a random pattern (Fig. 1), which was subjected to deformation.

The hardware platform was a PC with the following parameters:

- CPU – Intel I7 M620 2,67GHz,
- GPU – NVIDIA Quadro FX 2800M,
- RAM – 8GB.

Normalised 2D correlation has been tested in different environments: MatLab, MatLab with Parallel Computing and through custom application using OpenCV library with TBB performed on a computer processor (CPU) and uses the graphics processor called GPU. Tests were performed on greyscale images of size 2448x2050. Grid markers with distances between markers equal 50 pixels were used in all tests (ca. 1500–2000 markers), and displacements were sought in range equal to three times of size of the template; for example, for the template of size 30x30 pixels, the determined displacement was in the range of ± 45 pixels from the base point. An example of the displacement map obtained during the tests is shown in Fig. 2.

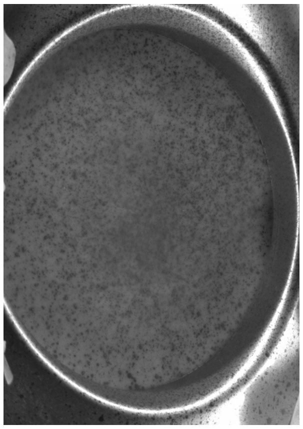


Fig. 1. Tested object

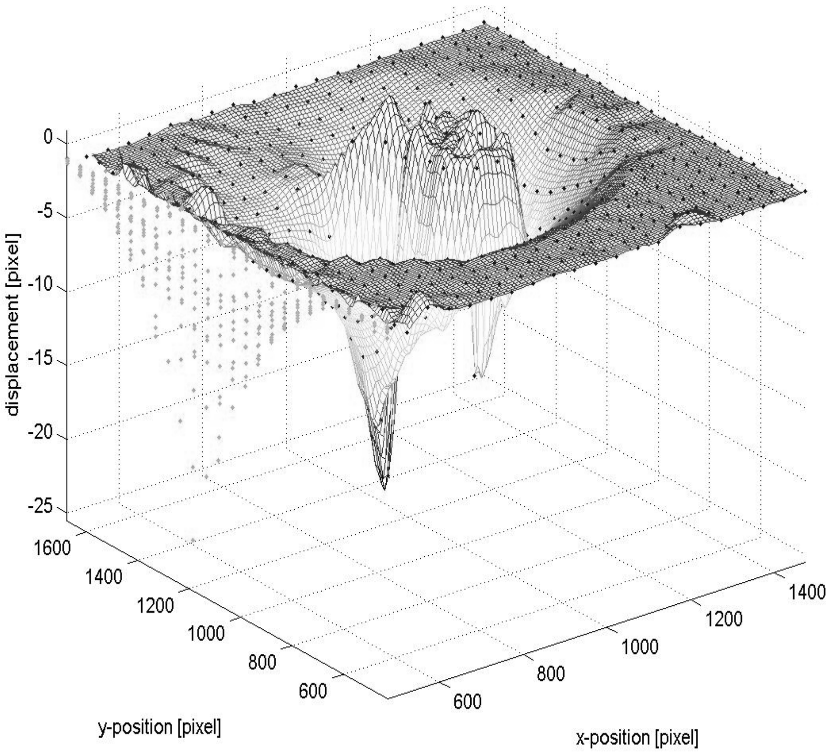
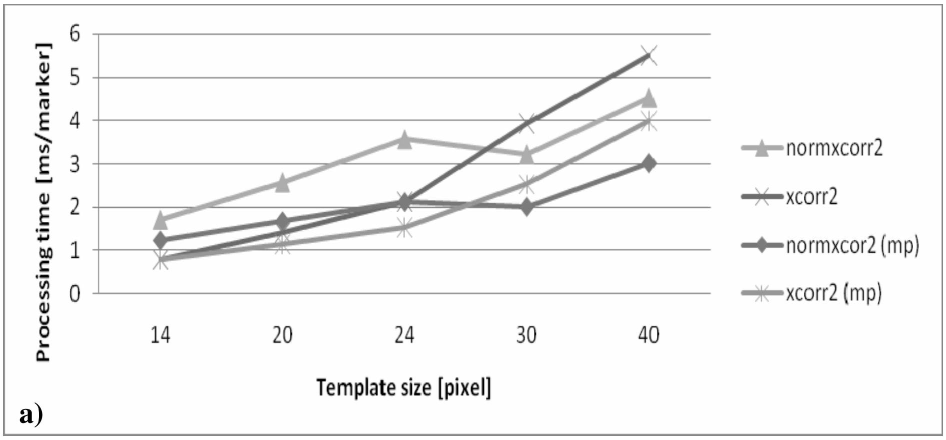


Fig. 2. Displacement versus x-y position

The first test consisted of DIC execution speed comparison between Matlab's scripts using *xcorr2* and *normxcorr2* functions. Additionally, a Parallel Computing toolbox impact on the calculation speed was tested. Achieved results are shown in Fig. 3.



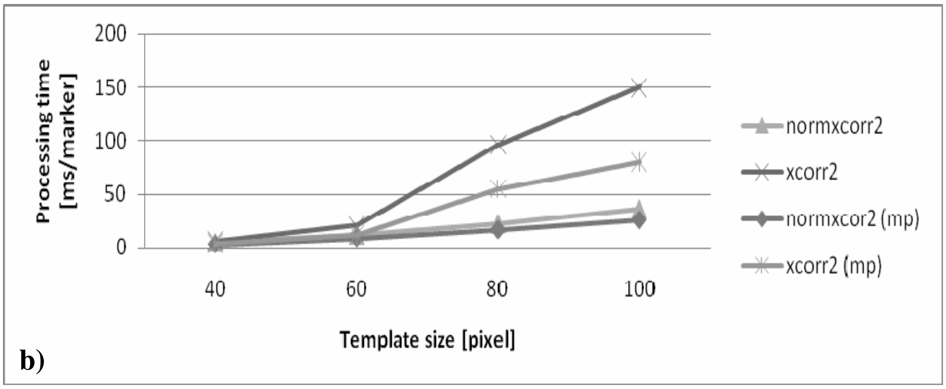


Fig. 3. Time consumption for single marker versus size of template in two cases, a) for small size of template, b) for big size of template

Measurement of both functions execution time showed that the normalised correlation is calculated slower, then its non-normalised counterpart for template sizes are up to 40x40 pixels. Most likely, it is due to the means of inner implementation of these functions. The *xcorr2* function is realised by the aid of conjunction and the *normcorr2* by the FFT-IFFT use. In addition, as is shown in Fig. 2b), the use of Parallel Computing Toolbox significantly accelerates the *xcorr2* function calculation (up to 2 times). The *normxcorr2* function toolbox gain is smaller and does not exceed 30% for 100x100 template sizes.

In the next test normalised 2D correlation was calculated in the custom "C++" application in the two ways. The first used a OpenCV library with IPP and TBB add-ons. Since the tests proved that the use of TBB library does not bring any increase of speed (low CPU load), it was decided to use native Windows API to spread calculations into threads. The second case used the newest OpenCV library ver. 2.2 with integrated CUDA technology. In the both cases, calculation time of one marker shift was measured, and the results are shown in Table 1.

Tab. 1. Normalised correlation processing time for different template size with OpenCV multi-threads and OpenCV - CUDA realisations

	Template size [pixel]	Processing time [ms/marker]						
		1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	GPU
1	14x14	0.289	0.178	0.145	0.136	0.144	0.144	4.587
2	20x20	0.442	0.263	0.229	0.221	0.212	0.22	4.61
3	24x24	0.570	0.365	0.297	0.289	0.289	0.289	4.58
4	30x30	1.159	0.668	0.588	0.588	0.57	0.57	4.81
5	40x40	1.650	1.01	0.86	0.85	0.86	0.86	4.84
6	60x60	4.390	2.49	2.33	2.31	2.3	2.34	6.86
7	80x80	6.770	3.99	3.6	3.39	3.32	3.56	11.41
8	100x100	13.460	8.28	7.22	6.8	7.23	7.02	15.22
-	CPU Utilisation							GPU Utilisation
9	-	27%	55%	100%	100%	100%	100%	5%

Our test shows that splitting calculations into separate threads brings the biggest gains when using up to 4 threads. Further increasing the number of threads does not improve the calculation speed and can even introduce minor speed degradation. This observation was also confirmed by the CPU load shown in Table 1 – row 9. The use of four threads forces 100% CPU load; thus, further increasing the number of threads will not shorten the computation time. The highest performance at four threads is due to the physical structure of CPU – the i7 620M processor has 4 logical cores, so at the best case is that each thread has its own core. Similar measurements were made for applications that use the OpenCV library in the recently introduced 2.2 version. This version gives one the possibility to execute part of the OpenCV functions as GPU kernels. The results were presented in the last column of Table 1. GPU processor utilisation was measured by NVidia CUDA Visual Profiler software. Achieved computation times can compete with CPU single threaded times only at largest template sizes (more than 100x100 pixels). That poor result is due to the lack of the optimal use of the GPU resources – maximum measured GPU resources utilisation was 5%. Figure 4 shows the best single marker calculation times at three test setups.

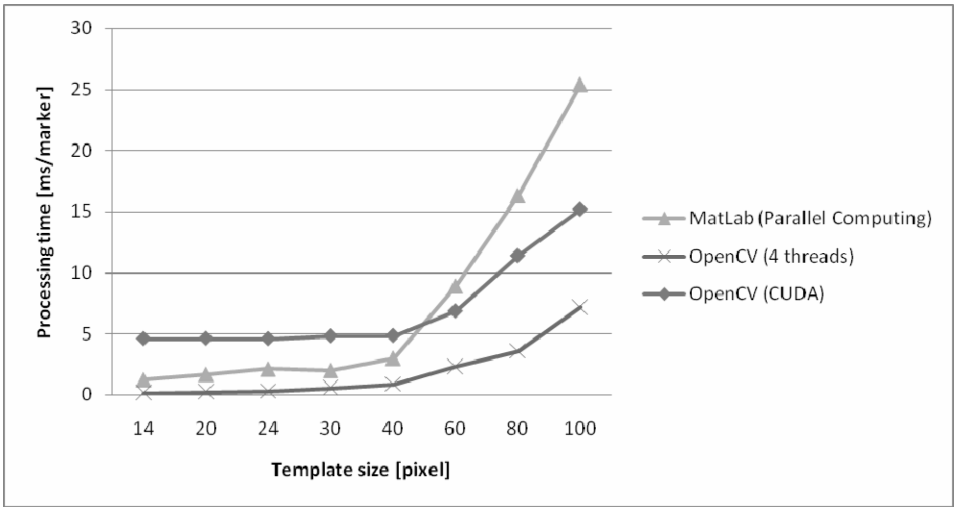


Fig. 4. Time consumption versus size of template in different realisations

Best calculation speed of the normalised 2D correlation function was achieved with the use of the OpenCV library functions calls executed in 4 and more parallel threads. This solution outperforms Matlab environment calculations (multithreaded version) by 300% at template size 100x100 pixels, up to 1000% for the smaller templates.

Conclusions

The tests indicate the following:

– The use of OpenCV library permits better calculation efficiency than the universal tool; however, with increasing computational complexity (larger template size), the gap between those two tools diminishes.

– The use of computational procedures taking advantage of GPU power does not necessarily lead to faster algorithm execution. To maximise the use of GPU potential, it is necessary to optimise the overall computational problem rather than its constituent elements. In this particular case, a change is needed that allows simultaneous determination of several marker shifts.

– Currently available PC's have performance when coupled with appropriate image analysis algorithms to sufficiently carry out the calculations in real time.

Scientific work executed within the Strategic Programme "Innovative Systems of Technical Support for Sustainable Development of Economy" within Innovative Economy Operational Programme.

References

1. Sunwoo Myung H. and Oh Seong K.: A Multiplierless 2-D Convolver Chip for Real-Time Image Processing. Journal of VLSI Signal Processing, 2004, 38, p. 63–71.
2. Ren Y.J., Zhu J.G., Yang X.Y. and Ye S.H.: The Application of Virtex-Pro FPGA in High-Speed Image Processing Technology of Robot Vision Sensor. Journal of Physics: 2006, Conference Series 48, p. 373–378.
3. Darabiha A., MacLean W.J., RoseMachine J.: Reconfigurable hardware implementation of a phase-correlation stereo algorithm. Vision and Applications, 2006, 17(2), p. 116–132.
4. Strenski D.: FPGA Floating Point Performance – a pencil and paper evaluation. HPC Wire, 2007.
5. Opis biblioteki Intel Integrated Performance Primitives 7.0. http://www.gambit.nazwa.pl/intel/?intel-integrated-performace-primitives-(ipp)-6.1,130
6. Informacje producenta. http://software.intel.com/en-us/articles/intel-ipp/
7. Informacje producenta. http://www.nvidia.com

Reviewer:
Jens Myrup PEDERSEN

Cyfrowa Korelacja Obrazu – Uniwersalne narzędzia a rozwiązania użytkownika

Słowa kluczowe

Metody korelacji, DIC, GPU, CUDA, wielowątkowość.

Streszczenie

Rozwój optoelektroniki i rosnąca szybkość przetwarzania procesorów zachęca do stosowania różnorodnych metod pomiarowych wykorzystujących dane pozyskane z kamer cyfrowych. W artykule zaprezentowano wyniki badań szybkości działania funkcji korelacji wykorzystywanych przy obróbce obrazów. Jedną z najczęściej wykorzystywanych funkcji w przetwarzaniu obrazów jest funkcja korelacji. Polega ona na poszukiwaniu współzależności pomiędzy dwoma obrazami i wymaga stosunkowo dużego nakładu obliczeniowego. Szybkość działania funkcji sprawdzono podczas wykorzystania różnych narzędziach takich, jak MatLab, MatLab z biblioteką Parallel Computing, biblioteka OpenCV oraz OpenCV wer.2.2 z możliwością wykorzystania technologii CUDA. W trakcie testów wyznaczono uzyskiwane prędkości przetwarzania obrazów, zwłaszcza w przypadku obrazów rejestrowanych za pomocą kamer wysokiej rozdzielczości. Przedstawiono wnioski dotyczące możliwości zastosowania funkcji korelacji, w tym z wykorzystaniem mechanizmu wielowątkowości.