Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 3, 571–580
DOI: 10.2478/v10006-010-0042-2

PARALLEL IMPLEMENTATION OF LOCAL THRESHOLDING IN MITRION–C

TOMASZ KRYJAK, MAREK GORGOŃ

Laboratory of Biocybernetics, Department of Automatics
AGH University of Science and Technology, Al. Mickiewicza 30, 30–059 Cracow, Poland
e-mail: {kryjak,mago}@agh.edu.pl

Mitrion-C based implementations of three image processing algorithms: a look-up table operation, simple local thresholding and Sauvola's local thresholding are described. Implementation results, performance of the design and FPGA logic utilization are discussed.

Keywords: image processing, FPGA, Mitrion-C, local thresholding, reconfigurable systems.

1. Introduction

Image processing algorithms are often implemented in reconfigurable devices (FPGA) (Wiatr, 2003; Lai *et al*., 2007), which facilitate higher implementation performance and enable real-time data processing. Currently, several main development directions of image processing, analysis and recognition systems based on reconfigurable hardware can be indicated. Some designs focus on implementing image processing operations which have not been accomplished on the FPGA platform yet (Kokufuta and Maruyama, 2009). New design methodologies of image processing algorithms are proposed (Plavec *et al*., 2009), while existing algorithms are accelerated by implementing computing intensive routines in FPGA resources (Sotiropoulos and Papaefstathiou, 2009). Comparison of speed-up factor for various implementation platforms, i.e., GPU, CPU (GPP) and the FPGA, is also considered (Asano *et al*., 2009; Claus *et al*., 2009). Stereovision (Ibarra-Manzano *et al*., 2009), self-organizing (Wildermann *et al*., 2009) and other complex algorithms are also implemented on the FPGA. Furthermore, power reduction and power saving techniques in image processing systems are discussed (Kalaycioglu *et al*., 2009). An interesting image processing and analysis system which takes advantage of partial run-time reconfiguration is presented by Canto *et al.* (2009).

Most of the algorithms are designed in hardware description languages: the VHDL and Verilog (Cho *et al*., 2007). However, in recent years several tools have been developed which enable creation of higher abstraction level designs and their subsequent transformation into hardware circuitry descriptions. Examples include Handle-C (Vitabile *et al*., 2004), Impulse-C (ImpulseC, 2009) and graphical tools like System Generator (Murthy *et al*., 2008), PICO Extreme (Denolf *et al*., 2009) and PixelStreams (Jabłoński *et al*., 2006). A broad overview of high-level tools for circuit design can be found in the works of Araby *et al.* (2007) and Edwards (2006).

This work's primarily focus is to verify the usefulness of the Mitrion-C language in image processing operations. The only former example of implementing image processing algorithms in Mitrion-C is the Sobel edge detection, as presented in MitrionUserGuide (2008). Originally, only simple video channel and LUT (look-up table) operations were implemented (Sections 3 and 4), followed by implementation of two variants of the local thresholding algorithm—simple and Sauvola's (Section 5), the latter requiring real number computations. Fixed-point and floating-point implementations were compared (Section 5.4). Additionally, a modification to Sauvola's algorithm was proposed, enabling a reduction of FPGA resource usage.

2. Mitrion-C

Mitrion-C is a high-level language used to develop reconfigurable computing applications. Its syntax is similar to that of ANSI C. Mitrion-C describes data flow and data dependencies, rather than instruction execution order. The code is compiled into a configuration of the so-called Mitrion Virtual Processor (MVP) (Mitrion-C, 2009). The processor's architecture provides for instruction level parallelism (all instructions can be executed concurrently),

loop-level parallelism (many loop iterations can be executed concurrently) and loop-level pipelining (for loops with data dependency). The MVP lacks an instruction stream. Instead, data flow is used through fine-grain processing elements, ordered according to the user-designed configuration of the FPGA. MVP design is transformed into Processor Circuit Design (IP Core, VHDL), which is then synthesized, placed, routed and used to configure the FPGA device. The Mitrion-C programming flow is shown in Fig. 3.

Mitrion-C was designed to support high-performance reconfigurable computing platforms, such as SGI RASC RC 100, Cray XD1, Nallatech BenDATA, Scan Engineering Telecom SAMC-707 and XtremeData XD2000F (i). SGI RC 100 was chosen as a target platform of implementation, due to its availability at the Academic Computer Centre *Cyfronet AGH*. The RC 100 board contains two Virtex 4 LX 200 devices and cooperates with the SGI Altix 4700 supercomputer. The abstraction layer provided with Mitrion SDK allows single instruction access to memory resources.

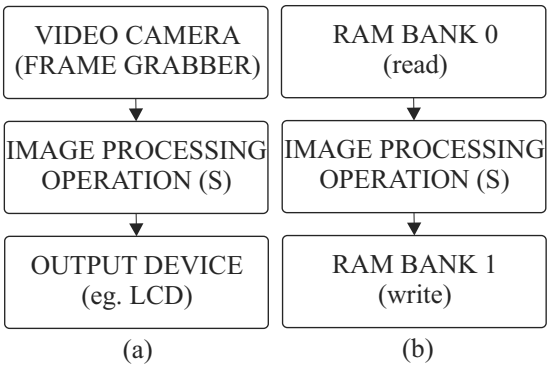

Fig. 1. Video channel.

3. Video channel and LUT implementations

The video channel is a basic image processing application (Fig. 1). It consists of a video source (i.e., a camera and frame grabber) and an output device (e.g., an LCD screen). When used in real time, the video channel typically consists of a camera, an image processing module and an output device (Fig. 1(a)). Alternatively, it may consist of input and output memory banks that accumulate video frames before and after processing, respectively (Fig. 1(b)). Pixels are read from the first RAM bank, processed consecutively and stored in the second RAM bank. Since SGI RC 100 does not support video input and output devices, the other video channel scheme can only be implemented.

The processed image consists of 256×256 pixels. The image could be transmitted in 4096 cycles (a single 128-bit wide data word is transmitted in one clock cycle), due to a 128-bit wide data bus of RC 100's RAM. However, simulation results prove that 4127 cycles are required to move the image from input to output memory. The difference results from the latency of SRAM access operation. Implantation results of the video channel (Table 1) serve as a reference for other image processing applications.

A look-up table (LUT) was implemented as the first image processing operation. It is widely used in IP applications (thresholding, gamma-correction, affine-operations, etc.). In the Mitrion-C language the operation is described as reading values from a one-dimensional array of constants. The results (Table 1) indicate that the compiler of Mitrion-C assigns BlockRAM FPGA resources to perform the LUT operation. The implementation flow of this algorithm is presented in Fig. 2. It is worth noticing that the LUT operation only slightly increases FPGA resource usage (Table 1, LUT-VC column).

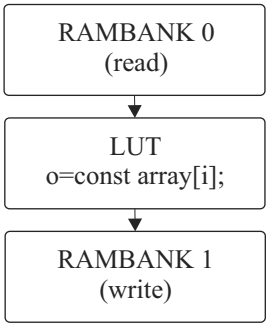

Fig. 2. Implementation flow of the LUT.

4. Local thresholding

The most straightforward strategy for image analysis uses the brightness of regions in the image as a means of identification. It is assumed that the same type of feature will have the same brightness throughout the whole image. Uniform illumination, which allows easier and more reliable image segmentation, can be achieved mainly indoors by using artificial light sources and isolating the subject from sunlight, light reflexes, etc. However, real-life and real-time image processing tasks are performed in non-uniform lightning conditions.

The non-uniformity of light can be minimized or even eliminated by using image processing algorithms. In some cases it is possible to separate a "background" image and use it to "level" subsequent images. This approach is often called background generation (subtraction). Fitting a background function is another possibility. A function $B(x, y)$, which approximates the background, is created using brightness values of selected points in the image and least-squares fitting (Russ, 2002). Furthermore, local segmentation algorithms (i.e., local thersholding), which operate on parts of the original image (ROI—Region of

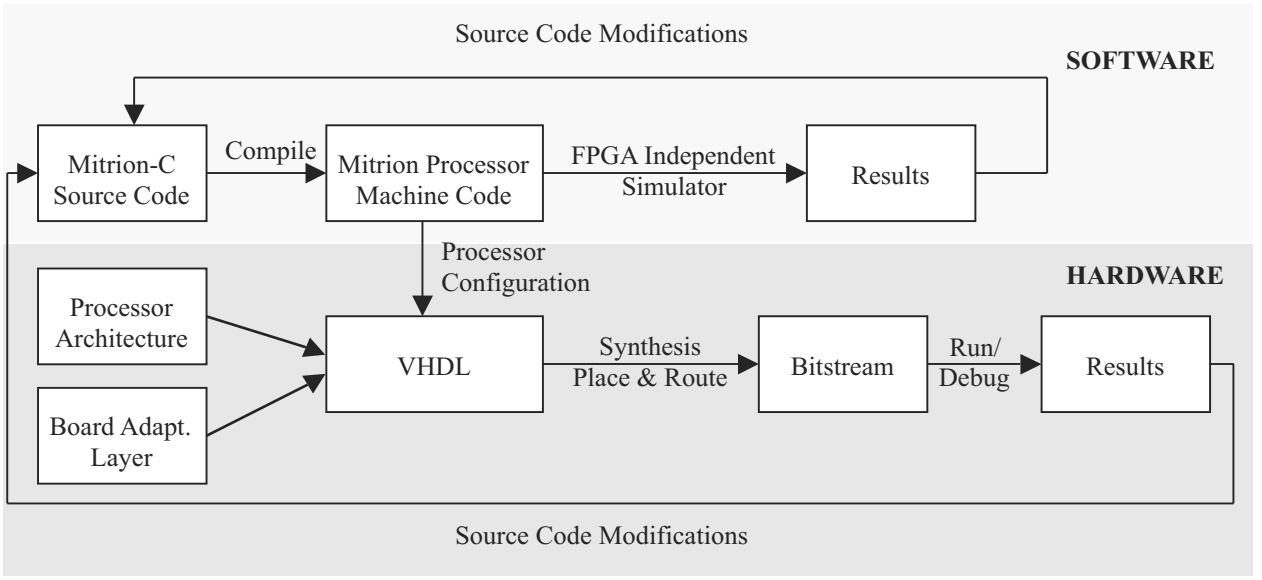

Fig. 3. Mitrion-C design flow.

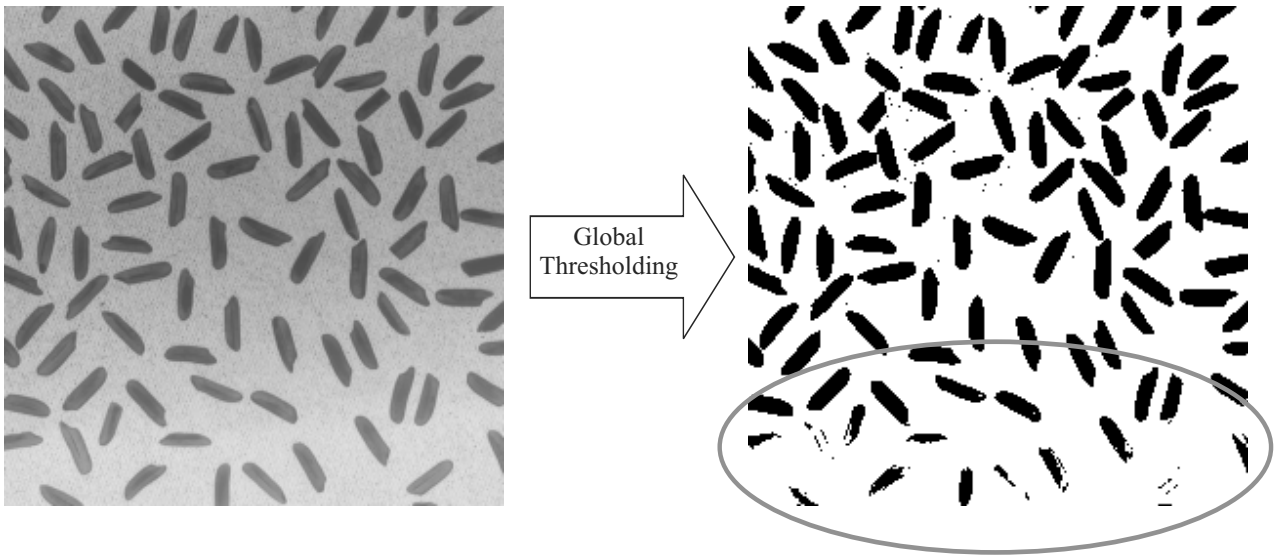

Fig. 4. Global thresholding (Otsu's method) for a picture with non-uniform illumination.

Interest), can be used (Gocławski *et al*., 2009). An interesting approach to local image processing has also been presented by (Rafajłowicz *et al*., 2008).

The paper introduces a local thresholding algorithm implementation on an FPGA device. Thresholding is one of the basic image processing operations. It is the simplest method of segmentation, allowing extraction of meaningful objects from the background. A standard thresholding procedure uses a single parameter—a threshold value. In order to determine the threshold, the image's histogram is usually analyzed (either manually or automatically). Several automatic thresholding methods have been described (Sezgin and Sankur, 2004). Among the most commonly used ones is Otsu's method (Otsu, 1979), which computes the threshold based on statistical parameters of the image's histogram. Global thresholding (establishing a single threshold for the whole image) proves to be very good for simple images with relatively uniform illumination. Images with non-uniform illumination can be problematic, as indicated in Fig. 4.

In order to achieve reliable thresholding results, a more complex method is required, e.g., either local or

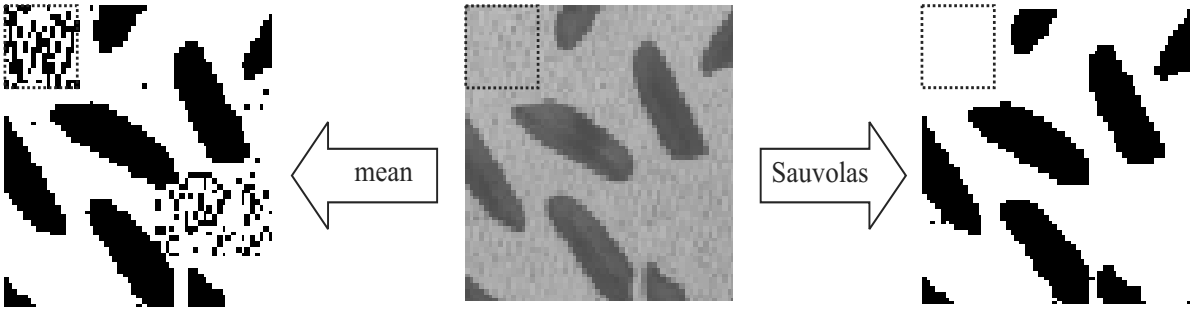

Fig. 5. Results comparison of simple and Sauvola's local thresholding methods.

adaptive thresholding (Shafait *et al*., 2008). The paper discusses two variants of local thresholding (LT): simple (based on the mean) and Sauvola's (Sauvola and Pietikainen, 2000) (based on the mean and standard deviation). The input image is divided into square windows (16×16 pixels). For each window a threshold is then determined and thresholding is performed. As opposed to regular thresholding, local thresholding is an example of a contextual operation (using a relatively large window). In the case of the simplest of LT methods, the threshold is calculated as the mean of pixel values in a single window:

$$t(a,b) = m(a,b) = \frac{\sum L(x,y)}{W^2}, \tag{1}$$

where $t(a,b)$ stands for the threshold for a given window, $m(a,b)$ stands for the mean, $L(x,y)$ are input image pixels in a given window and W stands for the size of a window (here it is set as 16).

The method does not perform well in windows where there are no foreground elements (a single window, without rice grains, is marked out in Fig. 5). In order to achieve better thresholding results, Sauvola proposed a method that calculates the threshold based on the mean and standard deviation (Sauvola and Pietikainen, 2000):

$$t(a,b) = m(a,b) \cdot \left[1 \pm k \cdot \left(\frac{s(a,b)}{R} - 1\right)\right], \tag{2}$$

where $t(a,b)$ stands for the threshold for a given window, $m(a,b)$ stands for the mean, $s(a,b)$ stands for standard deviation, k stands for a parameter (0.15), and R stands for a parameter (128). The '+' sign in (2) is used when objects are darker than background and the '−' sign when objects are brighter than background. The standard deviation is defined as

$$s(a,b) = \sqrt{v(a,b)}, \tag{3}$$

where $v(a,b)$ stands for the variance for a given window defined as

$$v(a,b) = \frac{\sum (L(x,y) - m(a,b))^2}{W^2}. \tag{4}$$

Both methods were implemented in the Matlab environment as a reference and then in the Mitrion-C language.

5. Implementing local thresholding in Mitrion-C

Local thresholding, as implemented on a sequential computer (e.g., in Matlab, C), is divided into two parts. The thresholds in each window are calculated first, and then the actual thresholding is applied. Such a procedure requires that each pixel in the RAM bank be read twice. Optimization of the method, using integral images, is described by Shafait *et al.* (2008). LT on a parallel computer (Mitrion Virtual Processor) should run in a pipeline manner and take advantage of FPGA parallelism.

There are two known FPGA implementations of local thresholding algorithms: by Jin *et al.* (2009) and Gorgon and Tadeusiewicz (2000). In the first one a simplified version of the LT algorithm, which is a part of an passive auto-focusing system, was implemented in the FPGA. The second article describes an LT system with a relatively small window size (8×8 pixels). The results of local thresholding with this window were quite similar to those of edge detection. Today, due the increase in logical resources available in FPGA devices, in particular lager on-chip memories, analysis of larger windows is possible. The presented LT implementation uses a 16×16 window and such advancement allows seeing local thresholding as an effective means of improving image quality, by eliminating non-uniform illumination in particular.

5.1. First LT implementation (LT1). In the first experiment the LT algorithm was directly ported from Matlab to Mitrion-C. Only the necessary modifications that resulted from the Mitrion-C syntax and the programming model were made. As has been mentioned, the RAM data word of the target platform (SGI RC 100) is 128-bit, which means that 16 pixel values are available at each clock cycle. These pixels were summed up (using the

Mitrion-C `for` loop), data were divided into blocks (using Mitrion-C `reshape` and `reformat` functions) and another summation was made. Finally, the result was divided by 256 (using a bit shift) to calculate the mean value in each window. Mean values were then used in the thresholding and the resulting image was written into the RAM bank. The implementation flow diagram is presented in Fig. 6.

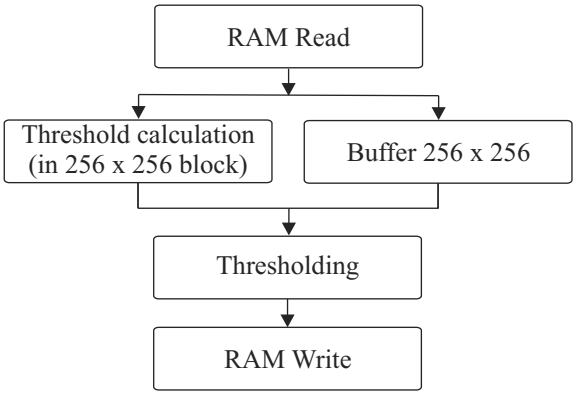

Fig. 6. First local thresholding implementation flow diagram.

A major disadvantage of such implementation is that the whole image (256×256) needs to be buffered. The thresholding operation can only start when all thresholds are available. This leads to high resource usage and more than 8000 algorithm steps needed to complete the operation (Table 1).

5.2. Modified LT implementation (LT2). The first LT implementation does not take advantage of the pipelining potential. To calculate the threshold in the first window, only 16 lines are needed (precisely, 15 lines and one 16 pixel block from line 16). The result can then be transferred to the thresholding module, where the actual thresholding takes place. Meanwhile, the threshold calculation module processes the next window, and so on. Such a scheme allows reducing the buffer size from 256×256 to 256×16.

In order to implement pipelining in Mitrion-C, a minor modification is needed. One instruction converts the input data from a 4096×16 pixel data block into a 16×256×16 pixel data block, enabling the Mitrion-C compiler to create the above mentioned pipeline. The implementation flow diagram is presented in Fig. 7.

Such a modification of the data flow allows reducing the amount of logical resources (Flip-Flops) used in the design by 42% (81% with VC excluded) and to decrease the number of algorithm steps needed to complete the operation by 47% (93% with VC excluded). It is worth noticing that, although the modification is rather simple, knowing how to divide the input data requires experience in designing pipelined and parallel algorithms.

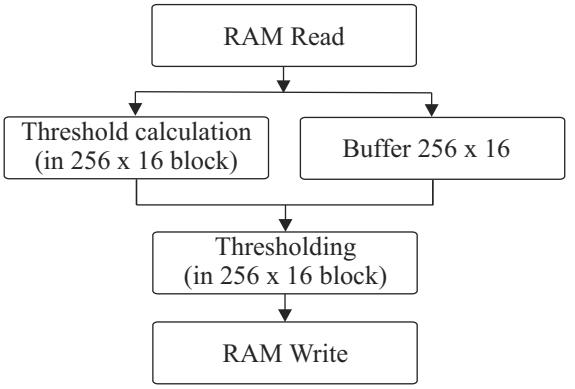

Fig. 7. Flow diagram of modified local thresholding implementation.

5.3. Sauvola's LT implementation. The simple local thresholding method can be implemented using only unsigned integer numbers. Sauvola's method incorporates arithmetic operations, which require real numbers: the square root (used to calculate the standard deviation (3)), division by the constant R (128) and multiplication by the constant k (0.15). Therefore Sauvola's algorithm can be implemented by

- using floating-point operations (`float`),
- using fixed-point operations,
- substituting the standard deviation (3) and (4) with absolute deviation:

$$ad(a,b) = \sum |L(x,y) - m(a,b)|. \qquad (5)$$

Implementing Sauvola's LT algorithm requires two additional modules: variance and threshold calculation. All computations in the variance calculation module are made using 24-bit unsigned integers. The result (the variance for a given window) is transferred to the threshold calculation module, where it is divided by W^2 (a bit shift by 8) and computed based on the formulas (3) and (2). The implementation flow diagram is presented in Fig. 8.

The Mitrion-C language supports floating-point calculations with a definable number of bits for both the mantissa and the exponent. From the designer's point of view, using Mitrion-C's native square root function (*sqrt*) and basic arithmetic operators is the simplest way to execute the operations described by Eqns. (3) and (2). The algorithm was implemented using different computational precisions: `float 6.6`, `float 7.6`, `float 8.6` and `float 9.6` (as in `float` *m.e*, where *m* stands for a number of mantissa bits and *e* stands for the number of exponent bits).

In order to validate the Mitrion-C results of Sauvola's LT module, they were compared with a Matlab implementation of the algorithm. All the calculations in the Matlab

Table 1. Summary and comparison of implementations results: VC, LUT and Simple LT (Mitrion-C simulator).

	Video Channel	Look - Up Table (LUT)		Simple Local Thresholding			
				LT1 (sequential)		LT2 (modified)	
	VC	LUT	LUT-VC*	LT1	LT1-VC*	LT2	LT2-VC*
No. of algorithm steps	4127	4129	2	8236	4109	4392	265
Flip-Flops used	18966	19512	546	39237	20271	22816	3850
Block RAM used	11	27	16	59	48	59	48
$18x18$ Multipliers used	0	0	0	0	0	0	0

*denotes difference between two methods

Table 2. Summary and comparison of implementations results: Sauvola's LT—floating-point (Mitrion-C simulator).

	Sauvola's Local Thresholding							
	F6.6	F6.6-VC*	F7.6	F7.6-VC*	F8.6	F8.6-VC*	F9.6	F9.6-VC*
No. of algorithm steps	4751	624	4756	629	4763	636	4768	641
Flip-Flops used	63760	44794	65936	46970	67065	48099	70769	51803
Block RAM used	108	97	108	97	108	97	108	97
18x18 multipliers used	16	16	48	48	48	48	48	48

*denotes difference between two methods

Table 3. Summary and comparison of implementations results: Sauvola's LT—fixed-point (Mitrion-C simulator).

	Sauvola's local thresholding					
	Fixed	Fixed - VC*	Fixed&Float	Fixed&Float - VC*	AD	AD-VC*
No. of algorithm steps	4740	613	4720	593	4678	551
Flip-Flops used	54174	35208	53603	34637	34679	15713
Block RAM used	108	97	108	97	108	97
18x18 Multipliers used	48	48	48	48	32	32

*denotes difference between two methods

Table 4. Comparison of Matlab and Mitrion-C results.

	Sauvola's local thresholding							
	F 6.6	F7.6	F8.6	F9.6	F10.6	Fixed	Fixed&Float	AD
DP	98	46	23	24	30	54	53	113
% of all	0.15%	0.07%	0.04%	0.04%	0.05%	0.08%	0.08%	0.17%
MDT	4	2	1	1	1	2	2	3
NoDT	177	120	72	71	82	148	145	234
% of all	69.14%	46.88%	28.13%	27.73%	32.03%	57.81%	56.64%	91.41%

environment were made using double precision (64-bit) floating-point arithmetic. The Matlab local thresholding results served as a reference to the Mitrion-C implementation results. The outcome of this comparison is presented in Table 4. Three quality indicators were considered: the number of pixels by which the reference image differs from the image obtained by executing the Mitrion-C code on a Mitrion virtual processor simulator (DP—different pixels), the maximal difference of threshold values (MDT) and a number of different thresholds (NoDT). All tests were done for both "rice" and "catalog" images, although the results presented in Table 4 relate to "rice" only . In each experiment, the results for the "catalog" image were analogous.

Analysis of data presented in Table 4 indicates that `float 8.6` precision is sufficient for the examined algorithm. The MDT value is 1 and results from a rounding error in the mean calculation module (integer division by 256). The number of different pixels (DP) is less than 0.1% of all pixels in the image and has no influence on the quality of thresholding. Table 2 presents FPGA resources usage. Higher computational precision leads to higher usage of logical resources (especially flip-flops) and a slight increase in the amount of algorithm execution steps.

5.4. Sauvola's fixed-point LT implementation. Floating-point operations implemented in the FPGA consume a lot of logical resources. The next stage of the research concentrated on using fixed-point calculations in Sauvola's algorithm. The first problem that had to be solved was a fixed-point implementation of the square root operation. The *sqrt* function available in Mitrion-C is designed for floating-point calculations only, and therefore a fixed-point square root module had to be

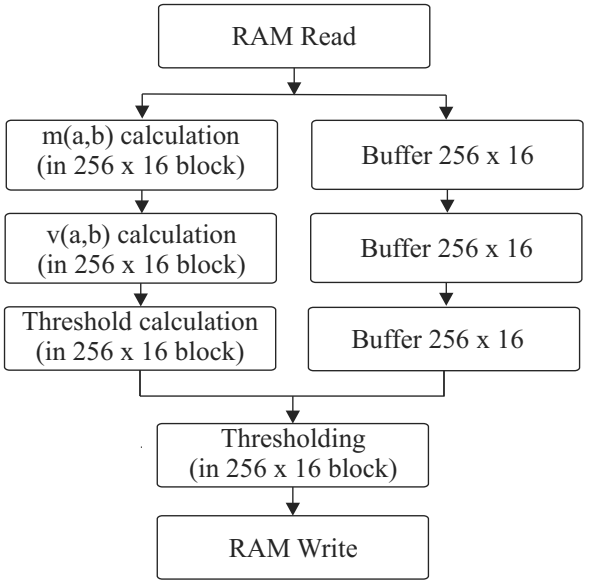

Fig. 8. Flow diagram of Sauvola's LT implementation.

implemented. The non-restoring square root algorithm described by Piromsopa *et al.* (2001) was used. It was decided to round the result of square root calculation to an integer value. Subsequently, a fixed-point calculation of the threshold was implemented (2). Based on preliminary research, it was decided to use eight bits to store the fractional part of a number. Table 4 presents a comparison with the reference image and Table 3 presents FPGA resource usage. The method proves very effective (MDT value is 2, less than 0.1% of pixels differ), providing substantial FPGA resource savings (12891 FF less than `float 8.6`). It also needs fewer algorithm steps to complete the calculations.

In the course of additional experiments it was established that the fixed-point implementation of the square root, using the non-restoring method, requires more FPGA resources than the Mitrion-C native floating-point *sqrt* function. This is likely to result from the highly effective native implementation of the square root module. Based on this observation, a new implementation of Sauvola's algorithm was created, in which the square root calculation was made using floating point arithmetic and subsequent calculations were made using fixed-point one. The results (Table 4) are very close to those of the previously mentioned fixed-point method. Resource usage (as per Table 3, Fixed&Float column) is slightly lower, while the number of algorithm steps is the same.

The last stage of the research concentrated on trying to replace standard deviation with absolute deviation. A preliminary research done in the Matlab environment showed that such an approach does not lead to significant deterioration of the thresholded image. Using absolute deviation eliminates the need for square root calculation. Therefore, all operations can be easily performed using fixed-point arithmetic. The results, presented in Table 4, indicate a slight deterioration of thresholded image quality. The MDT factor is 3 and less than 0.2% of all pixels differ. It is worth pointing out that in the case of thresholding a difference of this order does not influence the subsequent image analysis—the reference image and the Mitrion-C image are almost indistinguishable, especially for a human. Eliminating the square root resulted in a considerable reduction in usage of FPGA logical resources—by half in the case of flip-flops and 18×18 multipliers in comparison with the `float 8.6` version (Table 3, AD column). Furthermore, it takes 85 fewer algorithm steps to complete the computation.

It is worth mentioning that reprogrammable devices offer the ability of adapting to the required type of computations and the size of the computational element (i.e., data type and data bus bit width), based on the precision required by a given algorithm. The above-mentioned modifications are possible in working systems. This kind of adjustment is not available in the case of general purpose processors (GPP). Thus, where flexibility is required on one hand (not offered by ASICs, which cannot be modified after manufacturing) along with processing speed and low power consumption due to the adjustment of the architecture to a particular task on the other (not possible with GPPs), FPGA devices can be an alternative to other computational environments.

Results of local thresholding for selected images with non-uniform illumination are presented in Fig. 9.

6. Conclusion

The paper presents modern development trends in image processing, analysis and recognition systems, based on reconfigurable devices. It indicates that new possibilities arise to create such systems, due to an increased availability of FPGA resources and development of high-level tools for circuit design. The work's focus was to verify the usefulness of the Mitrion-C language and the Mitrion Virtual Processor environment in image processing operations.

The research indicates that, althougy Mitrion-C claims to abstract from the underlying hardware, it is crucial to take specific characteristics of the target resource into consideration. The programmer should know the basic structure of the FPGA and should have experience in designing parallel algorithms—being able to think of data flow rather than of the instruction execution sequence.

Implementation of the local thresholding algorithm, which took advantage of the parallelism available in the FPGA device after a data flow modification (as described in Subsection 5.2), allowed reducing logic usage by half. The computational resources analysis presented in Table 1 is worth reading. Data in columns 3, 5 and 7 indicate to-

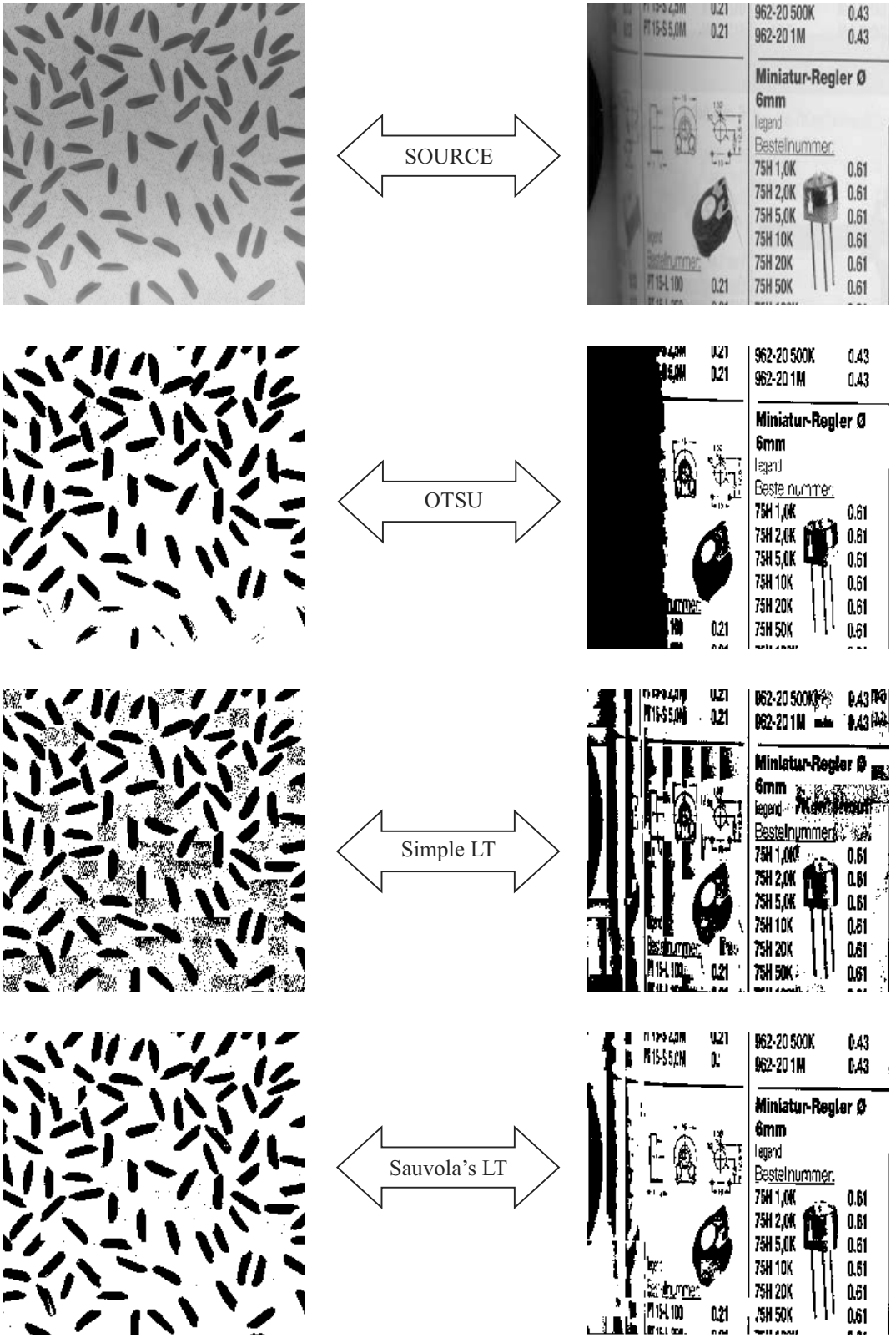

Fig. 9. Local thresholding examples.

tal resource usage differences by particular operation and video channel logic.

Sauvola's local thresholding, which requires real number operations, was successfully implemented in Mitrion-C. Both floating-point and fixed-point implementations were realized. Furthermore, a modification of the algorithm was proposed, enabling reduction of FPGA resource usage, which has virtually no effect on the final quality of local thresholding.

Floating-point operations are easily implemented in Mitrion-C. Logic resources usage strongly depends on the chosen number of mantissa and exponent bits. The proper selection of computational precision is very important to assure adequate results, while using the least FPGA resources.

The usefulness of Mitrion-C (Mitrion version 1.0) in image processing applications is limited to algorithms which operate on a still image due to the execution model: input data is read from one RAM bank and results are stored in another RAM bank. Implementation of a video processing application (Fig. 1(a)) is impossible because in the examined Mitrion-C version no input or output data stream are supported. On the other hand, describing quite complex algorithms, which require floating-point operations, is easy and does not take much time (in comparison to HDL description). Furthermore, the provided (with Mitrion-C) SDK graphical simulator allows easy debugging.

Concluding, Mitrion-C and the Mitron Virtual Processor where designed for a very specific group of applications—high performance reconfigurable computing. It supports FPGA boards which cooperates with supercomputers—SGI RASC RC 100 and SGI Altix 4700, both available at the AGH ACC *Cyfronet*. When an image processing algorithm is suitable for a hardware-software implementation and operates only on still images, a Mitrion-C implementation would be advisable. In other cases, especially for real-time video processing applications,a different design flow (i.e., HDL) is advisable.

Acknowledgment

The authors are grateful to the Academic Computer Centre *Cyfronet AGH* for partial support of this work.

This work was supported by the AGH University of Science and Technology under AGH Grant No. 11.11.120.612.

References

Asano, S., Maruyama, T. and Yamaguchi, Y. (2009). Performance comparison of FPGA, GPU and CPU in image processing, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 126–131.

Canto, E., Fons, M., Lopez, M. and Ramos, R. (2009). Acceleration of complex algorithms on a fast reconfigurable embedded system on Spartan-3, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 429–434.

Cho, J., Jin, S., Pham, X., Kim, D. and Jeon, J. (2007). FPGA-based real-time visual tracking system using adaptive color histograms, *Proceedings of the IEEE International Conference on Robotics and Biomimetics, ROBIO 2007, Sanya, China*, pp. 172–177.

Claus, C., Huitl, R., Rausch, J. and Stechele, W. (2009). Optimizing the SUSAN corner detection algorithm for a high speed FPGA implementation, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 138–145.

Denolf, K., Neuendorffer, S. and Vissers, K. (2009). Using C-to-gates to program streaming image processing kernels efficiently on FPGAs, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 626–630.

Edwards, S. (2006). The challenges of synthesizing hardware from C-like languages, *IEEE Design & Test of Computers* **23**(5): 375–386.

El-Araby, E., Nosum, P. and El-Ghazawi, T. (2007). Productivity of high-level languages on reconfigurable computers: An HPC perspective, *International Conference on Field-Programmable Technology, ICFPT 2007, Kitakyushu, Japan*, pp. 257–260.

Gocławski, J., Sekulska-Nalewajko, J., Gajewska, E. and Wielanek, M. (2009). An automatic segmentation method for scanned images of wheat root systems with dark discolourations, *International Journal of Applied Mathematics and Computer Science* **19**(4): 679–689, DOI: 10.2478/v10006-009-0055-x.

Gorgon, M. and Tadeusiewicz, R. (2000). Hardware-based image processing library for Virtex FPGA, *in* J. Schewel, P.M. Athanas, C.H. Dick and J.T. McHenrz (Eds.) *Reconfigurable Technology: FPGAs for Computing and Appplications II*, Proceedings of SPIE, Vol. 4212, pp. 1–10.

Ibarra-Manzano, M., Devy, M., Boizard, J.-L., Lacroix, P. and Fourniols, J.-Y. (2009). An efficient reconfigurable architecture to implement dense stereo vision algorithm using high-level synthesis, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 444–447.

ImpulseC (2009). Impulse accelerated technologies website, `www.impulseaccelerated.com`.

Jabłoński, M., Przybyło, J. and Gorgon, M. (2006). Real-time implementation of motion detection algorithm based on Pixelstreams, *Proceedings of the IFAC Workshop on Programmable Devices and Embedded Systems, PDeS 2006, Napa, CA, USA*, pp. 186–190.

Jin, S., Cho, J., Kwon, K. and Jeon, J. (2009). A dedicated hardware architecture for real-time auto-focusing using an FPGA, *Machine Vision and Applications* **21**(5): 727–734.

Kalaycioglu, C., Ulusel, O. and Hamzaoglu, I. (2009). Low power techniques for motion estimation hardware, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 180–185.

Kokufuta, K. and Maruyama, T. (2009). Real-time processing of local contrast enhancement on FPGA, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 288–293.

Lai, H.-C., Savvides, M. and Chen, T. (2007). Proposed FPGA hardware architecture for high frame rate (100 fps) face detection using feature cascade classifiers, *Proceedings of the First IEEE International Conference on Biometrics: Theory, Applications and Systems, BTAS 2007, Crystal City, VA, USA*, pp. 1–6.

Mitrion-C (2009). Mitrion website, `www.mitrionics.com`.

MitrionUserGuide (2008). Mitrion user guide—Image processing using Sobel convolution, Mitrionics AB, Lund.

Murthy, S., Alvis, W., Shirodkar, R., Valavanis, K. and Moreno, W. (2008). Methodology for implementation of unmanned vehicle control on FPGA using system generator, *Proceedings of the 7th International Caribbean Conference on Devices, Circuits and Systems, ICCDCS 2008, Cancun, Mexico*, pp. 1–6.

Otsu, N. (1979). A threshold selection method from gray level histograms, *IEEE Transactions on Systems, Man and Cybernetics* **9**(1): 62–66.

Piromsopa, K., Aporntewan, C. and Chogsatitvataa, P. (2001). An FPGA implementation of a fixed-point square root operation, *International Symposium on Communications and Information Technology, ISCIT 2001, Bangkok, Thailand,* pp. 587–689.

Plavec, F., Vranesic, Z. and Brown, S. (2009). Enhancements to FPGA design methodology using streaming, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 294–301.

Rafajłowicz, E., Wnuk, M. and Rafajłowicz, W. (2008). Local detection of defects from image sequences, *International Journal of Applied Mathematics and Computer Science* **18**(4): 581–592, DOI: 10.2478/v10006-008-0051-6.

Russ, J.C. (2002). *Image Processing Handbook*, 4th Edn., CRC Press, Inc., Boca Raton, FL.

Sauvola, J. and Pietikainen, M. (2000). Adaptive document image binarization, *Pattern Recognition* **33**(2): 225–236.

Sezgin, M. and Sankur, B. (2004). Survey over image thresholding techniques and quantitative performance evaluation, *Journal of Electronic Imaging* **13**(1): 146–165.

Shafait, F., Keysers, D. and Breuel, T.M. (2008). Efficient implementation of local adaptive thresholding techniques using integral images, *Proceedings of the 15th Document Recognition and Retrieval Conference (DRR-2008), Part of the IS&T/SPIE International Symposium on Electronic Imaging, San Jose, CA, USA,* pp. 681510–681510-6.

Sotiropoulos, I. and Papaefstathiou, I. (2009). A fast parallel matrix multiplication reconfigurable unit utilized in face recognitions systems, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 276–281.

Vitabile, S., Gentile, A., Siniscalchi, S. and Sorbello, F. (2004). Efficient rapid prototyping of image and video processing algorithms, *Euromicro Symposium on Digital System Design, DSD 2004, Rennes, France*, pp. 452–458.

Wiatr, K. (2003). *Acceleration of Computations in Vision Systems*, WNT, Warsaw, (in Polish).

Wildermann, S., Walla, G., Ziermann, T. and Teich, J. (2009). Self-organizing multi-cue fusion for FPGA-based embedded imaging, *International Conference on Field Programmable Logic and Applications, FPL 2009, Prague, Czech Republic*, pp. 132–137.

Tomasz Kryjak earned the M.Sc. degree in automatics and robotics in 2008 from the AGH University of Science and Technology in Cracow, Poland. Since 2008 he has been a research assistant at the Institute of Automatics of the same university. His current research focuses on image processing and recognition, biometrics, reconfigurable FPGA systems, hardware algorithm acceleration, and software/hardware co-design.

Marek Gorgoń earned the M.Sc. degree in electronics and control engineering in 1988, Ph.D. in automatic control and robotics in 1995, and D.Sc. (habilitation) in 2007, all three from the AGH University of Science and Technology in Cracow, Poland. Since 1994 he has been working at the Institute of Automatics of the same university, currently as an assistant professor. His research interests include image processing, reconfigurable devices and systems architecture, software-hardware co-design, DSP and FPGA devices and applications. He is a member of the IEEE Computer Society (since 2002) and the IEEE (since 2006). He is a member of IPCs of many international conferences. He is an author of 50 scientific papers and technical reports. He earned the awards of the Rector of his home university in the years 2002, 2004 and 2008. He has participated in 11 scientific and industrial research projects. He regularly reviews for international conferences, journals, and the Polish State Committee for Scientific Research.

Received: 20 June 2009
Revised: 18 November 2009
Re-revised: 1 March 2010