Computing and Informatics, Vol. 31, 2012, 727–741

PIPELINE IMPLEMENTATION OF PEER GROUP FILTERING IN FPGA

Tomasz Kryjak, Marek Gorgon

Laboratory of Biocybernetics, Department of Automatics
AGH University of Science and Technology
Al. Mickiewicza 30, 30-065 Krakow, Poland
e-mail: {kryjak, mago}@agh.edu.pl

Communicated by Jacek Kitowski

Abstract. In the paper a parallel FPGA implementation of the Peer Group Filtering algorithm is described. Implementation details, results, performance of the design and FPGA logic resources are discussed. The PGF algorithm customized for FPGA is compared with the original one and Vector Median Filtering.

Keywords: Colour image processing, reconfigurable systems, FPGA, parallel algorithms

1 INTRODUCTION

Image smoothing and noise removal algorithms are the basis of many image processing applications. Linear filtering with Gaussian mask and median filtering are two often used methods. Both are well known and tested for gray-scale images [1]. For colour images a common approach to remove impulse noise is Vector Median Filtering (VMF) [2] and its modifications [3, 4, 5, 6].

In this paper a parallel and pipeline implementation of the Peer Group Filtering algorithm (PGF) in FPGA is presented. The PGF algorithm for colour images was originally introduced in [7] and some modifications are presented in [8]. According to [7] Peer Group Filtering effectively removes impulse noise and smoothes colour images without blurring edges and other details. Its main drawback is a high computational complexity, therefore a parallel implementation is advisable.

Reconfigurable systems are a good platform for many image processing algorithms [9, 10] although there are some limitations in direct porting algorithms from software to FPGA implementation, i.e. real number operations. Floating point operations are possible to implement in FPGA resources, but may require the use of significantly more logical resources and sophisticated methods of design [11]. Therefore, usually some change or simplification of the algorithm is required. In this paper several modifications to the original PGF algorithm are proposed (Section 3). Their influence on the filtration result is examined and discussed (Section 4). Finally, PGF algorithm and its implementation in FPGA is compared with Vector Median Filtering.

2 PGF – PEER GROUP FILTERING

A peer group for a given point of the image (pixel) is a set composed of its neighbors, which are similar to it on the basis of calculating the appropriate distance and similarity metrics. The basic PGF approach for colour images was described in the paper [7]. PGF can be divided into two basic steps: classification and replacement. In the first step, classification, the peer group is chosen. In the second phase, replacement, the original pixel value is substituted by a pixel value obtained from the weighted average of the peer group members. The main purpose of averaging over the peer group, instead of the entire window, is to avoid edge blurring. The article describes how to customize the software version of the algorithm, so that an effective implementation in reprogrammable FPGA hardware resources is possible.

Consider an input colour image size $X \times Y$ (720×576 in the described implementation) in RGB colour space. Every pixel $x(n)$ at position n is a vector consisting of three values $\{R, G, B\}$. Let $x_0(n)$ denote a pixel vector centered in a $w \times w$ window ($w = 5$). In the first step of the PGF, distances $d_i(n)$ between the central and other pixels in the window are calculated:

$$d_i(n) = ||x_0(n) - x_i(n)||, \quad i = 0, \dots, k, \quad k = w^2 - 1. \tag{1}$$

Originally, the use of the Euclidean norm has been proposed, although there are some other possibilities: from the simple L1 distance (so called Manhattan) to more complex, described in detail in [12]. The second step of PGF is sorting the calculated $d_i(n)$ values (in ascending order):

$$d_0(n) \leq d_1(n) \leq \dots \leq d_k(n). \tag{2}$$

The next stage is impulse noise removal. First order differences between subsequent distances are calculated:

$$f_i(n) = d_{i+1}(n) - d_i(n), \quad i = 0, \dots, k-1. \tag{3}$$

Subsequently the first and last M values from the series $f_i(n)$ are tested to check if they belong to impulse noise (M was originally proposed as $M = \mathit{floor}(w/2)$):

$$f_i(n) > \alpha : i \leq M \vee i \geq k - M. \tag{4}$$

The α threshold is set large for highly corrupted images and small for slightly corrupted ones. When the $f_i(n)$ value is greater than the threshold, the pixel is considered as impulse noise and excluded from further processing. The remaining values are used to estimate the true peer group. It is worth noticing that the central pixel $x_0(n)$ could also be classified as impulse noise. In this case the true peer group is estimated by the remaining pixels from the window.

In the consecutive stage the actual peer group for the central pixel is determined. A modification of the Fisher's linear discriminant [13] is used. The criterion to be maximised is:

$$J(i) = \frac{|a_1(i) - a_2(i)|^2}{s_1^2(i) + s_2^2(i)}, \quad i = 1, \dots, k' \tag{5}$$

where:

- k' – number of pixels after impulse noise removal

$$a_1(i) = \frac{1}{i}\sum_{j=0}^{i-1} d_j(n) \text{ and } a_2(i) = \frac{1}{(k+1-i)}\sum_{j=i}^{k} d_j(n) \tag{6}$$

$$s_1^2(i) = \sum_{j=0}^{i-1} |d_j(n) - a_1(n)|^2 \text{ and } s_2^2(i) = \sum_{j=i}^{k} |d_j(n) - a_2(n)|^2. \tag{7}$$

The criterion $J(i)$ is calculated for each i and then the i value, where $J(i)$ is the maximum, is found:

$$m(n) = \arg\max_i J(i). \tag{8}$$

The peer group $P(n)$ of the size $m(n)$ for the central pixel $x_0(n)$ is defined as

$$P(n) = \{x_i(n) \mid i = 0, \dots, m(n) - 1\}. \tag{9}$$

The final stage of the PGF algorithm is Gaussian filtering:

$$x_{new}(n) = \frac{\sum_{i=0}^{m(n)-1} g_i \cdot p_i(n)}{\sum_{i=0}^{m(n)-1} g_i}, \quad p_i(n) \in P(n) \tag{10}$$

where g_i are the standard Gaussian weights depending on the relative position of $p_i(n)$ with respect to $x_0(n)$.

3 FPGA IMPLEMENTATION OF THE PGF ALGORITHM

The PGF algorithm is characterized by high computational complexity and long calculation times in case of implementation on a general purpose processor (GPP). A distance sort operation, for the context of 5×5 size and the need to calculate the index $J(i)$, have a major impact on the computational complexity of this algorithm. These operations are relatively simple and repeated for a large number of pixels per unit of time, so the algorithm can be described as data-dominated. Such algorithms are well suited to fine-grained parallelization and implementation in FPGA. The results of PGF filtration for colour images are usually better than those obtained by other methods (cf. Section 4). For this reason, an attempt to speed up the algorithm by describing it as a parallel and implement it in FPGA resources is well founded.

The PGF filtration module was designed in VHDL. The entry was a colour image, represented by the $\{R, G, B\}$ components and the output was represented by new $\{R', G', B'\}$ values. Each colour component was described as an 8-bit unsigned integer number ranging from 0 to 255. The module was designed to process images and video with a resolution of 720×576 pixels. The size of the context was established as in [7], 5×5 pixels. The module was implemented on Xilinx Virtex 4 LX 200 device. Most of the logic was described in VHDL, partly by generating code using scripts written in Matlab. Xilinx Core Generator Logic was used to implement some of the modules, i.e. delay lines and modules implementing the division operation.

Here, implementation of each stage of the PGF algorithm in FPGA is described and discussed in detail. Schematic diagram of the PGF module is shown in Figure 1.

3.1 Context Generation

The task of the first module was to determine the context (size 5×5) for the selected pixel. A known solution, based on the use of delay lines, allows collecting the local context [14, 15]. The scheme is shown in Figure 1 (in its upper part). Each pixel is represented by 24 bits. The module consisted of three identical subsystems that generate the context for each of 8 bits wide colour components $\{R, G, B\}$. The subsystem was constructed of four delay lines, 720 pixels long, and 25 1 pixel delay registers. Delay lines were implemented using Xilinx Logic Core Module RAM-based Shift Register, which had a much shorter synthesis time and made better use of FPGA logic resources than VHDL implementation.

To make a proper design of neighbourhood imaging operation, it is necessary to support implementation of pixels nearer to the edge of the image, for which it is impossible to determine all neighbours. Various approaches are used, including special rules along edges and corners, assuming that the image wraps around, mirroring of the image edges or rewriting the edge pixels of the original image. In the proposed module the last approach has been used.

3.2 Distance Calculating

When designing a module for calculating the distances, an assumption was made that the distances between the central pixel $x_0(n)$ and its 24 neighbors will be computed in parallel. Originally [7], the Euclidean (L2) measure of distance was proposed, which requires the calculation of the square, square root and operations on real numbers. Implementation of these operations in FPGA is possible; however, it requires considerable logic resources. In the present case it was necessary to execute $24 \times 3 = 72$ square operations and 24 square root calculations in parallel.

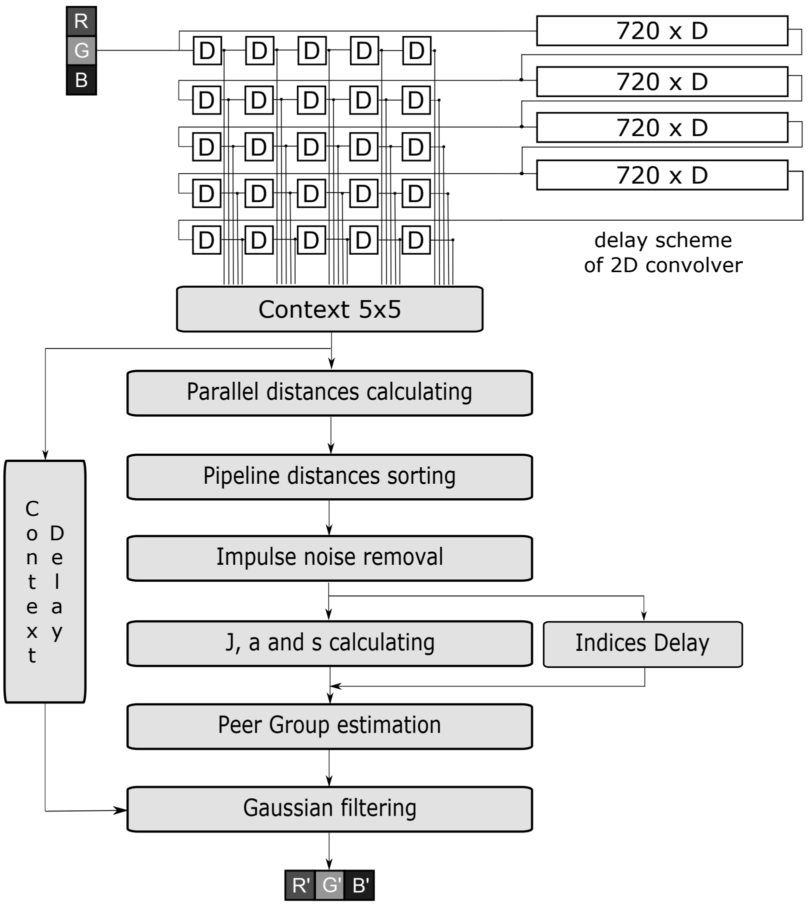

Fig. 1. PGF module scheme

It was decided therefore to make a simplification and replace the proposed L2 norm by L1 norm (the Manhattan distance):

$$d_i(n) = |x_0(n) - x_i(n)| \quad i = 1, \ldots, k, \quad k = 24. \tag{11}$$

3.3 Distance Sorting

Sorting of the $d_i(n)$ distances is a key stage of the impulse noise removal method introduced in the PGF algorithm. FPGA implementations of sorting algorithms are described in several papers [16, 17]. In this work the bitonic merge sort (BMS) algorithm was used [18]. This method is an example of a sorting network, which does not require control logic and is straightforward to parallelize. Therefore it can be implemented in FPGA very efficiently. A detailed description of the BMS algorithm can be found in [19].

In the described module 24 distances $d_i(n)$ were sorted. During the distance sorting, it was necessary to preserve the pixel index for each pixel. For this purpose pixels were indexed from $i = 1$ to 25 and the resultant indices i were sorted simultaneously with the distances. Most basic elements of the sorting network were two types of comparators:

type A – input values were switched, when first input value was greater than the second,

type B – input values were switched, when first input value was less than or equal to the second.

Only the two types of comparators described above and additional synchronization registers were used to implement the BMS module.

3.4 Impulse Noise Removal

The impulse noise removal method proposed in [7] was based on comparing M first and last values from a sorted series $f_i(n)$ with a threshold α and considering as impulse noise those pixels, whose distance to the central pixel exceeded the given threshold. Originally, for a 5×5 context, the M parameter value was proposed as $M = 2$. It was observed during preliminary tests that this value was too low and the filtration results were often unsatisfactory. It was noticed that the M value should be set depending on the level of noise in the image. It appeared that a modification of the original PGF algorithm, which allowed changes in the M value, was necessary. It was decided to implement the module, so that it would be possible to adjust the value as a parameter ranging from 2 to 7.

The input data for the module were: sorted distances $d_i(n)$, indices i, threshold α and the M value. In the first stage distances between subsequent $d_i(n)$ values were calculated (Equation (3)) and then they were compared with the threshold α (Equation (4)). Pixels, considered as impulse noise, were excluded from further processing – they had no influence on the new pixel value.

3.5 Parallel Calculating of the a_1 and a_2 Values

Parallel calculating of the a_1 and a_2 values turned out to be a complex issue during hardware implementation of the PGF algorithm. Simultaneous access to sums

$$\text{from } a_1(1) = d(0), a_2(1) = \sum_{(i=1)}^{24} d(i) \text{ to } a_1(24) = \sum_{(i=0)}^{23} d(i), a_2(1) = d(24) \qquad (12)$$

was required. The described sums are illustrated in Figure 2.

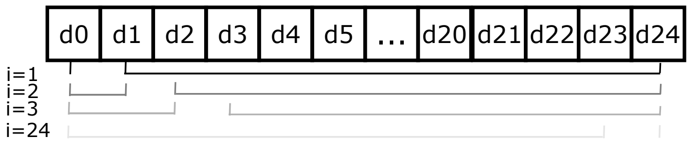

Fig. 2. Illustration of Equation (6)

The module was implemented as a summation tree. In the first stage sums of the adjacent values were calculated (i.e. $d(1)+d(2)$, $d(3)+d(4)$, described as *sum2*). In the subsequent stages, using the results from previous stages, sums of four, eight and sixteen values were calculated (i.e. $d(1)+d(2)+d(3)+d(4)$, described as *sum4*). Between each summation stage synchronization registers $R(j)$ were placed. In order to ensure pipeline operation it was necessary to store all summation results – shown schematically in Figure 3.

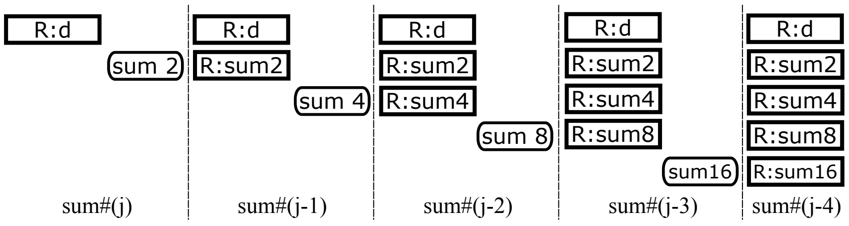

Fig. 3. Summation tree for a_1 and a_2 calculation. R – register, j – number of a discrete sample

Finally, the necessary sums, calculated according to Equation (6), were created using the registered partial sums (*R:d, R:sum2, R:sum4, R:sum8 and R:sum16*) and subsequently the results were divided as integer using Xilinx Logic Core Divider Generator.

3.6 Parallel Calculating of the s_1 and s_2 Values

Parallel calculation of s_1 and s_2 values required large logical resources. In the first step, it was necessary to calculate 600 absolute values of differences (Equation (7)). Implementing 600 square operations in parallel would be almost impossible in the considered FPGA device (Virtex 4 LX 200). Therefore it was decided to simplify the calculations by ignoring this operation. Summations described in Equation (7) were realised by means of a summation tree, like in Section 3.5.

3.7 The $J(i)$ Criterion

The 24 $J(i)$ criterion values were calculated in parallel. A modification to Equation (5) was required because of the square operation omitted in Equation (7). A new form of the criterion was proposed:

$$J(i) = \frac{s_1(i) + s_2(i)}{|a_1(i) - a_2(i)|}, \quad i = 1, \ldots, k'. \tag{13}$$

The change in the formula defining $J(i)$ caused a modification of Equation (8). Searching for the maximum was replaced by searching for the minimum. The division was implemented using Xilinx Logic Core Divider Generator and the search for the $J(i)$ minimum was implemented as a binary search tree.

3.8 Peer Group Estimation and Gaussian Filtering

In the peer group estimation module, using the minimum of the $J(i)$ criterion, pixels which form the true peer group were determined. Subsequently, for those pixels a filtration with a Gaussian mask was performed (Equation (12)). The filtration consisted of integer multiplication, a summation tree and division. Multiplication was implemented using the "*" VHDL operator and resulted in FPGA DSP48 block utilization (75 modules, 25 for each colour component). The division was implemented using Xilinx Logic Core Divider Generator. The Gaussian mask coefficients were predefined integers ranging from 0 to 63. The module's output was represented by new pixel values $\{R', G', B'\}$.

3.9 Additional Modules

In order to implement the PGF algorithm in pipeline two additional modules were necessary: a 59 clock cycles context delay (3 sets of 25 values, each set for a colour component) and a 46 clock cycles indices i delay (25 5-bit values). Both delays are presented in Figure 1.

3.10 Resource Usage

The estimated resource usage for each of the described modules is presented in Table 1. The results were obtained using the XST Synthesis tool included in Xilinx ISE 9.2 SP 4 software. Each of the modules was synthesized separately. As the target device the Xilinx Virtex 4 LX 200 was chosen.

After the place and route process, for the integrated PGF module, the following resource usage was obtained: 35 202 FF (19 % of Virtex 4 LX 200 logical resources), 57 152 LUTs (32 %), Slices 35 424 (39 %), DSP48 75 (78 %). An analysis of the results indicates that the module uses less than half of logical resources available in Virtex 4 LX 200 device. Therefore additional logic (i.e. communication with the host computer or image processing modules) could be implemented. Static timing analysis after the place and route indicates that the module should work at 100 MHz clock frequency.

Module	**LUTs**	**FF**	**DSP48**	**Slices**	**Estimated Max. Frequency**
Context Generator	31 320	8 352	–	2 160	321.337 MHz
Distances Calculating	2 160	241	–	1 210	132.327 MHz
Distances Sorting	6 361	2 761	–	3 350	173.214 MHz
Impulse Noise Removal	770	380	–	390	126.984 MHz
a_1 and a_2 calculating	9 137	11 695	–	6 622	214.362 MHz
s_1 and s_2 calculating	23 450	9 150	–	13 010	260.484 MHz
$J(i)$ calculating	6 649	9 852	–	5 677	123.265 MHz
Idices Delay	375	125	–	188	321.337 MHz
Context Delay	2 400	600	–	1 200	321.337 MHz
Peer Group Estimation	848	151	–	505	313.676 MHz
Gaussian Filtering	1 458	1 770	75	1 101	108.483 MHz

Table 1. FPGA resource usage – XST Synthesis tool (each module synthesized separately)

4 PGF IMPLEMENTATION EVALUATION

During the parallel implementation of the PGF algorithm some of its parts were modified: the Euclidean norm was substituted by the L1 (Manhattan) norm, all the divisions were integer and square operations in the variance (s_1, s_2) calculation were omitted. Furthermore the $J(i)$ criterion calculation was modified.

This simplification led to a difference between the results of the implementation of the algorithm: software, described in Section 2, and hardware, customized for FPGA, as described in Section 3. Therefore some tests were performed to examine this differences. The main aim of the Peer Group Filtering algorithm is impulse noise removal. For that reason, several test images were artificially corrupted with random impulse noise. An example of a pair of original a) and corrupted b) image is illustrated in Figure 4.

A popular method of removing noise from an image is Median Filtering. A simple Vector Median Filtering (VMF) algorithm, using the Euclidean norm in RGB colour space, was implemented in software and the filtration results are presented for comparison as an additional reference. Figure 5 shows a magnified selection of the Mandrill's eye (Figure 4). Figure 5 a) shows the original image, Figure 5 b) the noisy image, Figure 5 c) result of the VMF filter, Figure 5 d) image filtered by the floating point PGF version and Figure 5 e) image filtered by the customized for FPGA PGF version.

VMF removed all impulse noise although the result is blurred (despite of the rather small 3×3 context window). PGF gave better results than VMF. The results of both versions of PGF algorithm are very similar. The edges are preserved quite well and almost all impulse noise is removed. There is no difference in perceiving by a human observer between the filtered images generated by the PGF methods.

Next to the subjective, visual assessment of the obtained images, their comparisons were made using objective quality measures i.e. mean absolute error (MAE)

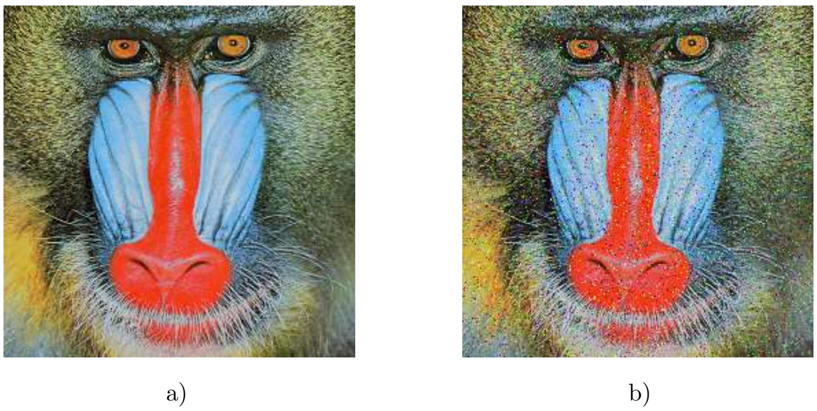

Fig. 4. Mandrill test image. a) original, b) corrupted with impulse noise – 5 % density

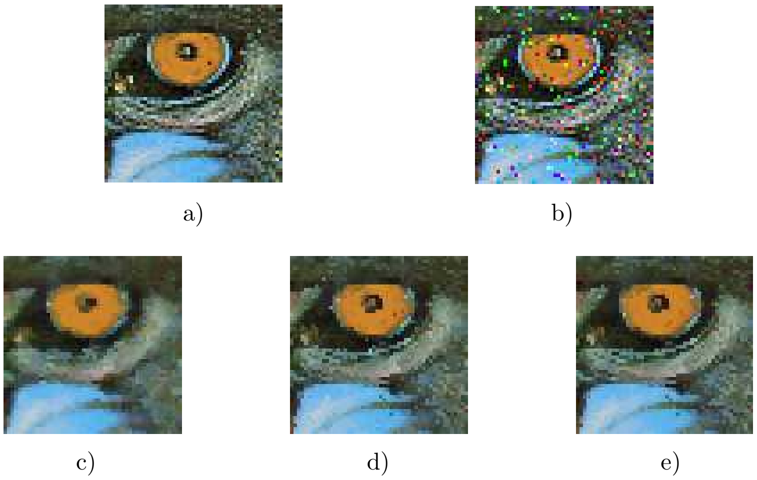

Fig. 5. Filtration results. a) original, b) noisy image, c) VMF 3×3 window, d) floating point PGF, e) PGF customized for FPGA.

and peak signal to noise ratio (PSNR). The test was performed on seven diverse colour test images corrupted by noise of various types and different levels: impulse (IN: 5 %, 2.5 %, 1 % of pixels modified), mixed impulse (5 %) and Gaussian (MIG: standard deviation 0.01 and 0.02). The results are presented in Table 2. Analysis of MAE results leads to the conclusion that the described hardware PGF algorithm should be considered as a new, filtering algorithm, particularly useful for FPGA implementation but not identical to the original floating point version. The PSNR ratio, in all cases greater than 40 dB, indicates that both filtration results are very similar.

	IN 5 %		IN 2.5 %		IN 1 %	
Image	MAE	PSNR [dB]	MAE	PSNR [dB]	MAE	PSNR [dB]
1	4.32	43.66	3.64	44.41	2.88	45.41
2	1.55	48.04	1.21	49.10	0.85	50.55
3	1.51	48.22	1.09	49.57	0.73	51.16
4	1.70	47.64	1.30	48.78	0.93	50.13
5	1.43	48.17	1.15	49.14	0.84	50.38
6	3.00	45.36	2.22	46.65	1.47	48.37
7	2.47	46.23	1.79	47.63	1.08	49.73

	MIG 0.01		MIG 0.02	
Image	MAE	PSNR [dB]	MAE	PSNR [dB]
1	7.47	41.17	9.34	40.18
2	5.60	42.35	7.52	41.08
3	5.91	42.14	7.92	40.88
4	6.02	42.07	8.06	40.81
5	5.69	42.28	7.59	41.04
6	6.52	41.74	8.53	40.57
7	6.10	42.04	8.09	40.81

Table 2. Comparison of floating point and hardware implementations of the PGF algorithm applied to 7 different noisy images with different type and level of noise

Visual assessment of images c), d) and e) presented in Figure 5 reveals that VMF blurs the image. To evaluate how the filtrations preserve the quality of edges, a sharpness metric based on the energy of the Laplacian of the image [20] was applied:

$$E_L = \iint \left[\frac{\delta^2 I(x,y)}{\delta x^2} + \frac{\delta^2 I(x,y)}{\delta y^2}\right]^2 \mathrm{d}x\,\mathrm{d}y; \tag{14}$$

where $I(x,y)$ – image.

Each of the images was converted to grayscale and the sharpness metric 14) was calculated. The results presented in Table 3 indicate that the energy of Laplacian in the PGF methods is greater than for the VMF method in all cases. The simplifications that were introduced in the hardware PGF method caused a slight loss of sharpness. The hardware method, however, provides significantly better results than VMF filtration.

5 TARGET PLATFORM AND APPLICATION TEST

As a target platform the SGI RC 100 with two Xilinx Virtex 4 LX 200 devices was chosen, mainly due to fast data bus between the FPGA and the host supercomputer (Altix 4700) and the amount of available logic resources in Virtex 4 LX 200 device. The platform is available at ACK Cyfronet AGH Krakow.

Image	Original	VMF 3×3	VMF 5×35	PGF Software	PGF Hardware
1	6.06	4.17	2.82	5.09	4.95
2	2.73	1.50	0.91	2.17	1.92
3	2.55	1.94	1.37	2.43	2.16
4	3.07	2.02	1.47	2.63	2.46
5	1.60	1.33	1.01	1.63	1.40
6	5.57	3.71	2.56	4.72	4.46
7	4.07	2.53	1.54	3.33	3.14

Table 3. Energy of the Laplacian for different images and filtering methods

The PGF FPGA implementation was tested in a behavioral simulation using the ModelSim 6.2 SE software. The results were consistent with a software model of the algorithm (implemented in Matlab). The estimated maximal clock rate is approximately 100 MHz which indicates that a single 720×576 colour image will be processed in approximately 0.0041 s (240 frames/s). The estimated computational power of the presented hardware processor is 248.8 GOPS (additions, subtractions, multiplications, divisions and comparisons).

6 CONCLUSION

In this paper a parallel implementation of the Peer Group Filtering algorithm was described. The design details of each PGF stage were presented. During porting the algorithm from software to hardware some simplification was done: all operations were performed on integer numbers, the L1 instead of the Euclidean norm was used and square calculations were omitted. This modification allowed to implement the PGF algorithm in the pipeline manner – new pixel values $\{R, G, B\}$ were generated every clock cycle (after an constant initial delay); however, the hardware results were slightly different from those generated by the original floating point version. Therefore some tests were performed to examine the influence of this modification on the quality of the filtering results. In was shown that although the software and hardware results were not the same there was no difference between those images for a human observer and only a slight difference of the filtering quality measured using MAE and PSNR. Additionally it was shown that both implementations of PGF algorithm preserve edges much better than Vector Median Filtering.

The main contribution of this paper is a parallel implementation of the Peer Group Filtering algorithm which differs from the original software version but provides a very similar filtering quality, utilizing the FPGA parallelism. This is the first described FPGA implementation of PGF algorithm.

PGF is the first stage of a colour image segmentation algorithm – JSEG [21]. Further research will focus on examining the influence of the simplifications on the final segmentation result and accelerating the whole JSEG algorithm using the hardware-software platform – SGI RC 100 and Altix 4700 supercomputer. In addition, after a few modifications, the PGF module could be used in an real-time colour image vision system.

Acknowledgments

The authors are grateful to the Academic Computer Center Cyfronet AGH for support of this work and access to hardware resources.

This work was supported by the AGH University of Science and Technology under Grant AGH 10.10.120.783.

REFERENCES

[1] Gonzalez, R.—Woods, R.: Digital Image Processing. Prentice Hall, New Jersey 2008.

[2] Astola, J.—Haavisto, P.—Neuvo, Y.: Vector Median Filters. Proc. of IEEE. Vol. 78, 1990, pp. 678–89.

[3] Trahanias, P. E.—Venetsanopoulos, A. N.: Vector Directional Filters – A New Class of Multichannel Image Process Filters. IEEE Trans. on Image Processing, Vol. 2, 1993, No. 4, pp. 528–34.

[4] Karakos, D. G.—Trahanias, P. E.: Combining Vector Median and Vector Directional Filters: The Directional-Distance Filters. Proc. of ICIP, Vol. 1, 1995, pp. 171–174.

[5] Morillas, S.—Gregori, V.—Peris-Fajarns, G.—Latorre, P.: A Fast Impulsive Noise Color Image Filter Using Fuzzy Metrics. Real-Time Imaging, Vol. 11, 2005, No. 5–6, pp. 417–428.

[6] Morillas, S.—Gregori, V.—Peris-Fajarns, G.—Sapena, A.: New Adaptive Vector Filter Using Fuzzy Metrics. Journal of Electronic Imaging, Vol. 16, 2007, No. 3.

[7] Deng, Y.—Kenney, S.—Moore, M. S.—Manjunath, B. S.: Peer Group Filtering and Perceptual Color Image Quantization. Proc. IEEE International Symposium on Circuits and Systems (VLSI), USA 1999, Vol. 4, pp. 21–4.

[8] Camarena, J.-G.—Gregori, V.—Morillas, S.—Sapena, A.: Some Improvements for Image Filtering Using Peer Group Techniques. Image and Vision Computing, Vol. 28, 2010, pp. 188–201.

[9] Wiatr, K.: Acceleration of Calculations for the Image System. WNT, Warsaw 2003 (in Polish).

[10] Gorgon, M.: Reconfigurable Architecture for Image Processing, Analysis and Digital Video Decoding. Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH, Krakow 2007 (in Polish).

[11] Wielgosz, M.—Jamro, E.—Wiatr, K.: Accelerating Calculations on the Rasc Platform: A Case Study of the Exponential Function. Reconfigurable computing: Architectures, tools and applications: 5th International Workshop: ARC 2009 Karlsruhe, Germany.

[12] Lukac, R.—Smolka, B.—Martin, K.—Plataniotis, K. N.—Venetsanopoulos, A. N.: Vector Filtering for Color Imaging. IEEE Signal Processing Magazine, Vol. 22, 2005, No. 1, pp. 74–86.

[13] Duda, R. O.—Hart, P. E.: Pattern Classification and Scene Analysis. John Wiley & Sons, New York 1970.

[14] Kruse, B.: A Parallel Picture Processing Machine. IEEE Transactions on Computers, Vol. C-22, 1973, No. 12, pp. 1075–1087.

[15] Stenberg, S. R.: Pipeline Architecture for Image Processing. Multicomputers and Image Processing, Academic Press 1982, pp. 291–305.

[16] Dong, S.—Wang, X.—Wang, X.: A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA. 2nd International Congress on Image and Signal Processing (CISP '09), pp. 1–4.

[17] Ratnayake, K.—Amer, A.: An FPGA Architecture of Stable-Sorting on a Large Data Volume: Application to Video Signal. 41st Annual Conference on Information Sciences and Systems (CISS '07), pp. 431–436.

[18] Mueller, R.—Teubner, J.—Alonso, G.: Data Processing on FPGA. Very Large Data Bases Conference, Lyon 2009.

[19] Batcher, K. E.: Sorting Networks and Their Applications. In: AFIPS Spring Joint Computer Conference 1968.

[20] Subbarao, M.—Choi, T.—Nikzad, A.: Focusing Techniques. Journal of Optical Engineering, Vol. 32, 1992, pp. 2824–2836.

[21] Deng, Y.—Manjunath, B. S.: Unsupervised Segmentation of Color-Texture Regions in Images and Video. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, 2001, No. 8, pp. 800–810.

Tomasz Kryjak received his M. Sc. degree in automatics and robotics in 2008 from AGH University of Science and Technology in Krakow, Poland. From 2008 on permanent position of Research Assistant at the Institute of Automatics, AGH-UST. His current research is focused on image processing and recognition, biometrics, reconfigurable FPGA systems, hardware algorithm acceleration and software/hardware co-design.

Marek Gorgon received his M. Sc. degree in electronics and control engineering in 1988, his Ph. D. in automatic control and robotics in 1995 and his Dr. Sc. (habilitation) in 2007, all from AGH University of Science and Technology in Krakow, Poland. From 1994, on permanent position at the Institute of Automatics AGH-UST, currently 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 IEEE Computer Society from 2002, and a member of IEEE from 2006. He is a member of IPC of many international conferences. He is the author of 50 technical papers and reports. In 2002, 2004 and 2008 he won the Awards of the Rector of AGH-UST University. He participated in 11 scientific and industrial research projects. He regularly reviews for international conferences, journals, and State Committee for Scientific Research.