AUTOMATIC DESIGN OF IMAGE OPERATORS USING EVOLVABLE HARDWARE

Lukáš Sekanina and Vladimír Drábek
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz, drabek@fit.vutbr.cz

Abstract. *An original approach to automatic design of image operators is presented in this paper. The proposed solution employs evolvable hardware at simplified functional level and produces image operators (digital circuits), which can compete against traditional designs in terms of quality and implementation cost in Xilinx's chips.*

1 Introduction

Image operators (like image filters or edge detectors) used mainly in the pre-processing phase, are an important part of the image recognition systems. Current industry calls for an automatic design of such image operators since (1) the recognition systems should adapt to changing environments automatically, and (2) it is expensive to pay designers when no standard solution can be easily adopted. Industrial applications (e.g. a traffic sign recognition or an automatic detection of correct/damaged components on the production line) also call for cheap but high performance solutions that can be effectively implemented in low-cost, commercial off-the-shelf hardware devices like FPGA (Field Programmable Gate Array).

In case of traditional image operator design, the designer has to (1) analyze an input image, (2) design an algorithm (an operator), which will perform a required operation, and (3) prepare a circuit implementation (e.g. a configuration of an FPGA). The design process is mostly based on experimental work with the images and it leads to a very time consuming job.

In this paper, we present an original approach to automatic design of image operators. The solution employs evolvable hardware and produces image operators (digital circuits), which can compete against traditional designs in terms of quality and implementation cost. The approach extends the previous paper [1], where it was shown that if an image is available both with and without noise, the whole design process of the image filter design can be done automatically without influence of a designer.

The next section briefly summarizes conventional image operators while an evolutionary approach and our experimental framework are introduced in Section 3. Evolved designs and their comparison to conventional approach are reported in Section 4. And finally, conclusions and problems for future work are given in Section 5.

2 Conventional image operators

The paper deals with gray-scale (8bits/pixel) images. Basic information on the theory of image operators is available e.g. in [2, 3] and it is summarized in the following subsections.

2.1 Image filters

Images usually acquired through modern cameras may be contaminated by a variety of noise sources. If a type of noise is known a priori, an efficient operator can be usually designed to suppress the noise.

Hardware implementation mostly operates in the spatial domain where the input image convolves with the filter function. In discrete convolution, the kernel (a small weight matrix) is shifted over the image and multiplies its values with the corresponding pixel values of the image.

Let us briefly describe mean (denoted as *FA1* in the paper), mean-2 (*FA2*), mean-4 (*FA4*) and median (*FME*) filters since these filters are usually used as the first attempts to solution, and in addition, our results will be compared with them in the next sections. Consider the kernel 3×3 in this paper. The idea of mean filtering is simply to replace each pixel value in an image with the mean (average) value of its neighbors, including itself. Mean-2 and mean-4 filters take some pixels in account several times and produce better results than mean filter for Gaussian noise since their coefficients are derived from the curve of Gaussian distribution. The kernels are defined as:

$$FA1 = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad FA2 = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad FA4 = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

In case of median filter, a pixel value is replaced with the median of neighboring values. A median filter is much better at preserving sharp edges than the mean filter since it does not create the new (potentially unrealistic) pixel values.

General hardware implementation of an image filter operating with a kernel 3×3 is based on a tree of adders with shifters while an implementation of the median filter employs the network of comparators and multiplexers [4].

2.2 Edge detectors

Sobel operator, which performs a 2D spatial gradient measurement on an image, is one of the most popular edge detectors. It can be considered as an image filter too. Its two convolution kernels are defined as:

$$p = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad q = \frac{1}{8}\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

A new pixel value is calculated according to the formula (1) in cheaper, mostly hardware implementations:

$$NewPixelValue = 128 + |p| + |q| \qquad (1)$$

3 Evolutionary image operator design

3.1 Evolutionary approach

In case of evolutionary image operator design, either the kernel or the whole function (i.e. a circuit) is automatically evolved. The resulting structure evolves from primitives instead of calculating coefficients for a general-purpose model. Evolved solutions (circuits) should be more efficient than conventional designs in terms of performance and

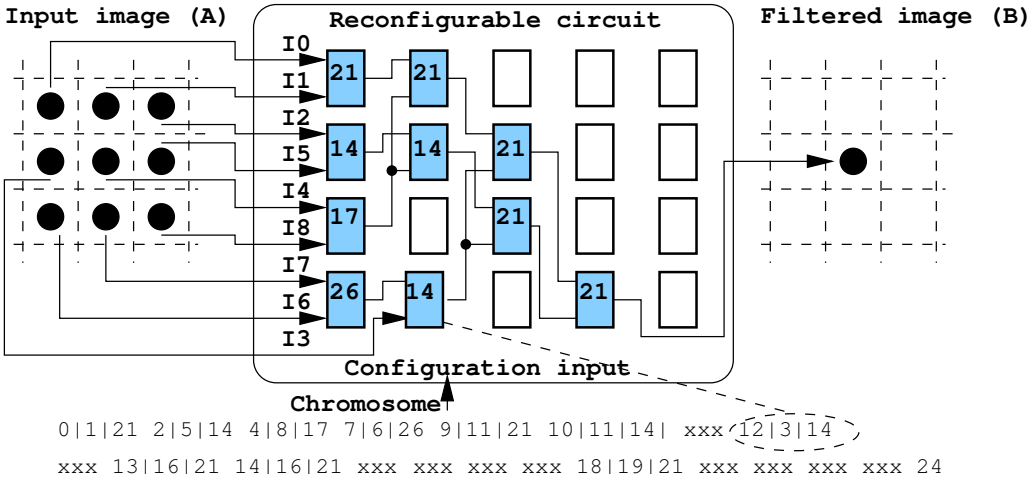

Figure 1: An example of the reconfigurable circuit and its configuration for an image operator. Nine inputs (pixel values) are used to calculate a new pixel value. Parameters: 9 inputs, 1 output, circuit topology 5×4, L-back=1. Only utilized programmable elements are marked.

implementation cost. Typical approaches to evolutionary image operator design published in past years are e.g. [5, 6, 7]. However, these approaches deal with the problem of a simple operator design. This paper tries to attack the problem of evolutionary design of a wide set of operators using a uniform approach.

3.2 Evolvable hardware

Evolvable hardware is a new method that was successfully applied for a design of digital circuits [8, 10]. In our case, the evolution is performed at simplified functional level, which requires only simple logical functions and adders (see Table 1). The method is based on a combination of *Cartesian Genetic Programming* (CGP) [9] and an evolution at functional level [11].

The reconfigurable circuit is modeled as an array of u (columns) × v (rows) programmable elements (gates). The number of circuit inputs and outputs is fixed. Feedback is not allowed. A gate input can be connected to the output of some gate in the previous columns or to some of circuit inputs. L-back parameter defines the level of connectivity and thus reduces/extends the search space. For example if L=1, only neighboring columns may be connected; if L=u, the full connectivity is enabled. For a given application, designer has to define: the number of inputs and outputs, L, u, v and a set of functions performed by programmable elements.

3.3 Experimental framework

We have employed evolvable hardware for the design of image operators according to Fig. 1. The new pixel value is calculated using 3 × 3 neighborhood and it is available on the 8bit output of the circuit. The circuit input consists of nine pixel values.

The approach presented in Fig. 1 can be exploited in two ways: (1) The evolutionary algorithm is used only in the design phase and the evolved operators are stored in a library to be reused instead of conventional operators. Quality as well as implementation cost of evolved operators placed into an FPGA is very important. (2) The evolutionary algorithm is a part of a target (e.g. embedded) system and the evolution is performed during the system execution to make the system adaptive to changing

Table 1: A list of functions implemented in a programmable element. The inputs a and b and the outputs operate over 8bits. Symbols used: $>>$ right shifter, $<<$ left shifter, $\wedge$ binary AND, $\vee$ binary OR, $\oplus$ binary exclusive-OR, $+$ 8bit adder, $\bar{a}$ is a binary negation of a. Constants are given in a hexadecimal system.

0	$a >> 1$	1	$a >> 2$	2	$a >> 4$
3	$\bar{a}$	4	$a << 1$	5	$a << 2$
6	$a << 4$	7	$(a << 4) \vee (a >> 4)$	8	0
10	FF	11	$a \oplus \bar{b}$	12	CC
13	$max(a, b)$	14	$(a + b + 1) >> 1$	15	$\bar{a} \vee b$
16	$\overline{a \wedge b}$	17	$(a \wedge 0F) \vee (b \wedge F0)$	18	$(a \wedge CC) \vee (b \wedge 33)$
19	$(a \wedge AA) \vee (b \wedge 55)$	20	$a + b$	21	$(a + b) >> 1$
22	$a \vee b$	23	$a \wedge b$	24	$a \wedge \bar{b}$
25	$\bar{a} \wedge b$	26	$a \oplus b$	27	$\overline{a \vee b}$
28	$\overline{a \oplus b}$	29	$a \vee \bar{b}$	30	$((a + b) >> 1) + 1$

operation conditions. Then the implementation cost is not so important since a number of gates available for evolution is constant and there is not usually any reason to distinguish between a small or a big circuit that produces the same outputs. Hardware implementation of the whole evolvable system can be based, e.g. on [12, 13].

Based on initial experiments, the following parameters were set up as default:

Reconfigurable circuit: Parameters of the reconfigurable circuit according to CGP are: 9 inputs (8bits), 1 output (8bits), $u = 10$ (columns), $v = 4$ (row), L-back $= 2$. A programmable element has two inputs and operates (inputs as well as outputs) over 8 bits. Table 1 lists functions supported in the programmable element.

Chromosome encoding: A chromosome is a fixed-size string of integers, containing $u \times v$ genes (corresponding to the programmable elements in the reconfigurable circuit) and one place devoted to the index of the element representing the circuit output (see a chromosome in Fig. 1). A gene is described by three values: the position of the first input, the position of the second input and an index of the function applied on inputs. Thus genotype is of fixed length while phenotype is variable length since all the programmable elements need not be used.

Population: Population size is 16. The evolution was typically stopped (1) when no improvement of the best fitness value occurs in the last 50000 generations, or (2) after 500000 generations.

Genetic operators: Mutation of two randomly selected gates is applied per circuit. A mutation always produces a correct circuit configuration. Crossover is not used. Four of the best individuals are utilized as parents and their mutated versions build the new population up (deterministic selection with elitism).

Fitness function: Let A denote an input image (e.g. an image corrupted by a noise), let B denote an image after application of the image operator (e.g. a filtered image) and let C denote an ideal version of the image A. The image C must be available in the training phase. The image size is $N \times N$ (N=256) pixels but only the area of 254 x 254 pixels is considered because the pixel values at the borders are ignored. The fitness value of a candidate chromosome is obtained as follows: (1) the circuit simulator is configured using a candidate chromosome, (2) so created circuit is used to calculate

pixel values in the image B, and (3) the differences between pixels of the images C and B are added and the sum is subtracted from a maximum value (representing the worst possible difference: #grey_levels × #pixels):

$$FitnessValue = 255.(N-2)^2 - \sum_{i=1}^{N-2}\sum_{j=1}^{N-2}|C(i,j) - B(i,j)| \qquad (2)$$

4 Results and discussion

Evolved and conventional operators are compared in this section. However, we know, for instance, the type of noise a priori and thus efficient conventional solutions may be prepared for comparison. It is not a case of real world situations, where we may not know anything about the noise and suitable conventional solution may not exist at all.

To compare hardware cost of evolved and conventional operators, VHDL implementations (at level of structural description) of these operators were designed and then synthesized using The Xilinx Integrated Software Environment tool to XC4028XLA chip. The results are given in the number of equivalent gates needed for implementation. All the circuits were described with the registers joined to the adders allowing pipeline execution.

4.1 Operators for smoothing images

We consider three types of noise denoted as: $G16$ (Gaussian with a mean of zero and a standard deviation of 16), $R32$ (uniform random with parameter 32) and $N1$ (block uniform random). Images with $G16$ and $R32$ noise were generated from originals using Adobe Photoshop program. We designed the $N1$ noise for testing purposes to model random defects in the image. $N1$ noise is generated as $R32$ noise but applied only for randomly selected blocks of the image. The rest of the image is left unchanged.

More than one hundred image filters were evolved and 6 of them are presented. *Avereage difference per pixel* (*dpp*) of the filtered and ideal images was measured over the test set to get general information about the filter quality. Lena image was used in the training phase (evolution). The test set contains the following images: Lena (popular for testing), Man (a man), Bld (a building), Cpt (a capacitor), and Rel (a relay). Cpt and Rel images were acquired through a camera and the system for automatic recognition of damaged/correct product on the production line.

Some interesting evolved designs as well as results of traditional filters are summarized in Table 2. It is evident that it is possible to evolve the filters with higher quality (less *dpp*) than conventional filters can reach. On the other hand, the number of equivalent gates is higher in some cases.

It was detected after analysis of the evolved designs that some filters do not employ all the functional elements effectively. For instance, the filter F57 uses a functional element with the same inputs. This element can be omitted (i.e. replaced by a direct connection) since it does not influence the output. The number of used functions in the evolved designs after manual optimization is listed in the column #O of the Table 2.

4.2 Shot noise

In case of shot noise (also called "salt and pepper" noise), some pixels (5% in our case) are randomly set up to maximum (255) or minimum (0) value. Conventional robust

Table 2: A list of evolved and conventional filters. Columns have these purposes: $G16, R32, N1$ – an average of *ddp* for all the test images and a given type of noise; $\#EqG$ – the number of equivalent gates needed after the synthesis; $\#E$ – the number of functional elements used in the evolved design; $\#O$ – the number of functional elements used after manual optimization; $functions\,used$ in the evolved designs (and their multiplicity); Gnr – the generation where the solution occured.

Filter	G16	R32	N1	#EqG	#E	#O	functions used	Gnr
F24	6.669	7.356	6.243	2128	21	14	8, 17, 22, 21(12), 14(6)	185168
F20	6.704	7.356	6.308	2626	17	15	17, 21(14), 30(2)	79369
F26	6.706	7.358	6.308	2128	19	14	17, 18(2), 21(12), 14(4)	24853
F21	6.708	7.348	6.322	3082	19	18	21(17), 30(2)	133224
F14	6.705	7.424	6.270	1790	14	12	17, 18, 21(9), 23, 30(2)	13678
F23	6.734	7.452	6.240	1368	10	9	18, 21(9)	42772
FA4	6.717	7.408	6.242	1397			Conventional designs	
FA2	6.872	7.486	6.511	1371				
FA1	7.172	7.723	6.847	1390				
FME	7.403	9.443	6.538	4740				

solution which suppresses shot noise requires the median filter. However a cheaper solution, which does not remove all the "salt and pepper", needs only a simple if-then-else function checking an occurence of 0 or 255. Very efficient filter F57 (see Fig. 2) is one of the filters evolved using the training image Lena. The filter F57 exhibits the same performance also for the test set. The only problem is that the F57 filter leaves about 9% of "pepper" pixels unchanged. On the other hand, the median filter leads to smoothing. Hardware implementation of the F57 filter is much cheaper than in the case of median filter (see Table 3).

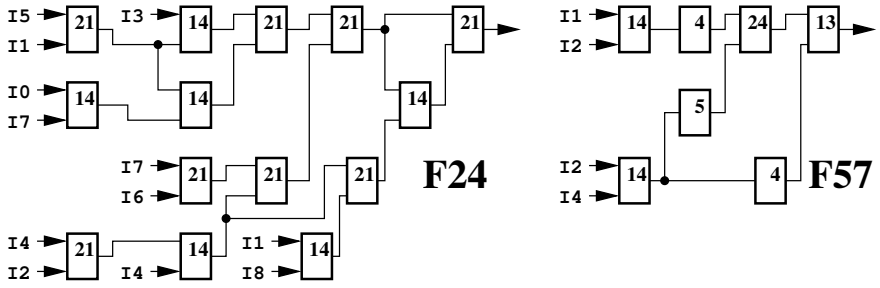

Figure 2: The best filters F24 and F57 evolved to suppress the Gaussian and the shot noise.

4.3 Sobel operator

We have tried to evolve a circuit operating as the Sobel operator. To do it, the proposed form of the fitness function has to be slightly modified. First, the conventional Sobel operator was applied on the source image to create an ideal image C according to the formula (1). Then the source image was also used as an input image A and it was processed by evolved operators to obtain the images B. The fitness value was calculated using the formula (2). The best resulting filter FS3 marks edges similarly to the Sobel

Table 3: The comparison of evolved and conventional operators. Columns have these purposes: *dpp* – *ddp* for the training image. $\#EqG$ – the number of equivalent gates needed after the synthesis; $\#E$ – the number of functional elements used in the evolved design; $\#O$ – the number of functional elements used after manual optimization; $functions\ used$ in the evolved designs; *Gnr* – the generation where the solution occured.

Operator	dpp	#EqG	#E	#O	functions used	Gnr
F57	1.703	441	8	7	4, 5, 24, 14, 13	63763
FME	2.954	4740			conventional design	
FS3	6.569	1350	21	17	0,1,22,15,23,24,26,28,14,13	29741
Sobel	-	1988			conventional design	
FL2	-	679	22	19	0,1,4,5,11,12,22,23,17,18,20,21,14,13	52972

operator for the test set, but it is more sensitive. Its hardware implementation is about 30% cheaper than for the conventional Sobel operator (see Table 3).

4.4 Illumination enhancement

The FL2 operator in Table 3 was evolved to improve illumination of the images with the linearly decreasing intensity. For testing purposes, the input image A was generated from an ideal image C as (for $i, j = 0 \ldots N-1$):

$$A(i,j) = C(i,j) - (i+j)\,/\,5 \qquad (if \quad A(i,j) < 0 \quad then \quad A(i,j) = 0)$$

It is important to note that FL2 operator uses only a local knowledge (i.e. 3×3 kernel) to enhance an image while a conventional solution will probably need the position of the currently processed pixel. The FL2 operator sufficiently improves illumination and since we do not know any conventional algorithm that uses a kernel 3×3, a comparison is not available. This FL2 is one of the operators, which was successfully evolved for an ad hoc designed non-traditional task.

5 Conclusions

Experiments performed with the proposed system allowed us to claim that is is possible to evolve general filters that exhibit in average less *dpp* than conventional filters for a given noise and test images (e.g. F24 for G16 and R32 noise, F23 for N1 noise, and F57 for shot noise). If an implementation cost is important, some evolved filters (e.g. F23, F57) require less equivalent gates than conventional filters (e.g. *FA4*, *FME*) and so outperform conventional filters totally. The same system can be used for automatic design of edge detectors and illumination enhancement operators. We can suppose that proposed approach will operate well in various other problems of image pre-processing.

It is also important to note that an 8bit adder is the most complicated function needed to evolve an image operator. Such simple function is commonly available or can be easily implemented in low cost devices like FPGA or simple microcontrollers. It takes in average about 70000 generations and so about 20 hours on Pentium III to evolve a sufficient operator.

The proposed approach is suitable mainly for automatic design of libraries of image operators. An evolutionary algorithm can be run several times and the best operator

taken after an automatic analysis of the evolved designs on the test set. The approach can be also efficient for on-line adapting systems. The question for future work is how to solve communication between evolvable hardware and the changing environment.

Acknowledgments

The research was performed with the Grant Agency of the Czech Republic under No. 102/01/1531 *Formal approach in digital circuit diagnostic – testable design verification* and the Research intention no. CEZ: J22/98: 262200012 – *Research in information and control systems.*

References

[1] Sekanina, L.: Image Filter Design with Evolvable Hardware. To appear In: Proc. of the 4th Evolutionary Image Analysis and Signal Processing Workshop EvoIASP'2002, LNCS, Springer-Verlag 2002, 12 pp.

[2] Šonka, M., Hlaváč, V., Boyle R.: Image Processing, Analysis and Machine Vision. Chapman & Hall, University Press, Cambridge 1993.

[3] Russ, J., C.: The Image Processing Handbook (third edition). CRC Press LLC 1999.

[4] Fučík, O.: Reconfigurable Embedded Systems. PhD thesis, Brno University of Technology, Czech Rep. 1997, 65 pp.

[5] Hollingworth, G., Tyrrell, A., Smith S.: Simulation of Evolvable Hardware to Solve Low Level Image Processing Tasks. In: Proc. of the Evolutionary Image Analysis, Signal Processing and Telecommunications Workshop EvoIASP99, LNCS 1596 Springer-Verlag, Berlin 1999, pp. 46–58.

[6] Dumoulin, J. et al.: Special Purpose Image Convolution with Evolvable Hardware. In: Proc. of the EvoIASP 2000 Workshop, Real-World Applications of Evolutionary Computing, LNCS 1803, Springer-Verlag, Berlin 2000, pp. 1–11.

[7] Erba, M. et al.: An Evolutionary Approach to Automatic Generation of VHDL Code for Low-Power Digital Filters. In: Proc. of the Genetic Programming European Conference EuroGP 2001, LNCS 2038, Springer-Verlag, Berlin 2001, pp. 36–50.

[8] Sanchez, E., Tomassini, M. (Eds.): Towards Evolvable Hardware: The Evolutionary Engineering Approach. LNCS 1062, Springer-Verlag, Berlin 1996.

[9] Miller, J., Thomson, P.: Cartesian Genetic Programming. In: Proc. of the Genetic Programming European Conference EuroGP 2000, LNCS 1802, Springer-Verlag, Berlin 2000, pp. 121–132.

[10] Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines, Vol. 1(1) 2000, pp. 8–35.

[11] Murakawa, M. et al.: Evolvable Hardware at Function Level. In: Proc. of the Parallel Problem Solving from Nature PPSN IV, LNCS 1141, Springer-Verlag Berlin 1996, pp. 62–72.

[12] Sekanina, L., Sllame, A.: Toward Uniform Approach to Design of Evolvable Hardware Based Systems. In: Proc. of the Field Programmable Logic And Applications FPL 2000, LNCS 1896, Springer-Verlag, Berlin 2000, pp. 814–817.

[13] Sekanina, L., Růžička, R.: Design of the Special Fast Reconfigurable Chip Using Common FPGA. In: Proc. of the Design and Diagnostic of Electronic Circuits and Systems IEEE DDECS'2000, Polygrafia SAF Bratislava, Slovakia 2000, pp. 161–168.