DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch062

Coevolutionary Cartesian Genetic Programming in FPGA

Radek Hrbáček and Michaela Šikulová

Brno University of Technology, Faculty of Information Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
xhrbac01@stud.fit.vutbr.cz, isikulova@fit.vutbr.cz

Abstract

In this paper, a hardware platform for coevolutionary cartesian genetic programming is proposed. The proposed two-population coevolutionary algorithm involves the implementation of search algoritms in two MicroBlaze soft processors (one for each population) interconnected by the AXI bus in Xilinx Virtex 6 FPGA. Candidate programs are evaluated in a domain-specific virtual reconfigurable circuit incorporated into custom MicroBlaze peripheral. Experimental results in the task of evolutionary image filter design show that we can achieve significant speed-up (up to 58) in comparison with highly optimized software implementation.

Introduction

Cartesian genetic programming (CGP) – a special variant of genetic programming (GP) – has been successfully applied to a number of challenging real-world problem domains (Miller, 2011). However, the computational power that evolutionary design based on CGP (as well as on standard GP) needs for obtaining innovative results is enormous for most applications. Often, the fitness in GP is calculated over a set of *fitness cases* (Vanneschi and Poli, 2012). A fitness case corresponds to a representative situation in which the ability of a program to solve a problem can be evaluated. Fitness case consists of potential program inputs and target values expected from a perfect solution as a response for these program inputs.

A set of fitness cases is typically a small sample of the entire domain space. The choice of how many fitness cases (and which ones) to use is often crucial since whether or not an evolved solution will generalize over the entire domain depends on this choice. However, in the case of digital circuit evolution, it is necessary to verify whether a candidate n-input circuit generates correct responses for all possible fitness cases (input combinations, i.e. 2^n assignments). It was shown that testing just a subset of 2^n fitness cases does not lead to correctly working circuits (Imamura et al., 2000). Recent work has indicated that this problem can partialy be eliminated in real-world applications by applying formal verification techniques (Vasicek and Sekanina, 2011).

Hillis (1990) introduced an approach that can automatically evolve subsets of fitness cases concurrently with problem solution. Hillis used a two-population coevolutionary algorithm (CoEA) applied to a test-based problem in the task of minimal sorting network design. Subsets of test cases used to evaluate sorting networks evolved simultaneously with the sorting networks. Evolved sorting networks were used to evaluate the test cases subsets. The fitness of each sorting network was measured by its ability to correctly solve fitness cases while the fitness of the fitness cases subsets was better for those that could not be solved well by currently evolved sorting networks.

Coevolutionary algorithms are traditionally used to evolve interactive behavior which is difficult to evolve with an absolute fitness function. The state of the art of coevolutionary algorithms has recently been summarized in (Popovici et al., 2012). A *test-based problem* is defined as a co-search or co-optimization problem with two populations – population of candidate solutions and population of *tests* (subsets of the fitness cases set).

In our previous work, inspired by *coevolution of fitness predictors* (Schmidt and Lipson, 2008) and the principles of the *competitive coevolution* introduced by Hillis (1990), we proposed a two-population coevolutionary CGP algorithm running on an ordinary processor in order to accelerate the task of symbolic regression (Sikulova and Sekanina, 2012b) and the evolutionary image filter design (Sikulova and Sekanina, 2012a). For our benchmark problems (5 symbolic regression problems and salt-and-pepper noise filter design) we have shown that the (median) execution time can be reduced 2-5 times in comparison with the standard CGP.

Despite the acceleration based on fitness cases coevolution, the CGP design is still computationally very intensive design method. Therefore an FPGA based acceleration platform has been designed. Modern FPGAs provide cheap, flexible and powerfull platform, often outperforming common workstations or even clusters of workstations in particular applications. Vasicek and Sekanina (2010) introduced a new FPGA accelerator of CGP with the aim to provide both high performance and low power. The architecture

contains multiple instances of *virtual reconfigurble circuit* (VRC, Sekanina (2003)) to evaluate several candidate solutions in parallel.

Inspired by the FPGA accelerator of CGP, we propose a hardware platform for parallel two-population CoEA and show that by using this platform, the execution time of evolutionary design using CGP can be significantly reduced. The proposed hardware accelerated coevolutionary CGP is compared with hardware-accelerated standard CGP and with a highly optimized software implementation of coevolutionary CGP in the task of evolutionary image filter design.

The paper is organized as follows. The next section introduces the idea of coevolution in cartesian genetic programming. In the following section the architecture of the proposed accelerator is presented. The remaining section is devoted to experimental evaluation of the accelerator in the benchmark problem – the image filter evolution. Conclusions are given in the last section.

Coevolution in Cartesian Genetic Programming

In standard CGP (Miller, 2011), a candidate program is represented in the form of directed acyclic graph, which is modelled as an array of $n_\mathrm{c} \times n_\mathrm{r}$ (columns $\times$ rows) programmable elements (nodes). The number of primary inputs, n_i, and outputs, n_o, of the program is defined for a particular task. Each node input can be connected either to the output of a node placed in previous l columns or to one of the program inputs. The l-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. Feedback is not allowed. Each node is programmed to perform one of n_a-input functions defined in the set Γ. Each node is encoded using $n_\mathrm{a} + 1$ integers where values $1 \ldots n_\mathrm{a}$ are the indexes of the input connections and the last value is the function code. Every individual is encoded using $n_\mathrm{c} \cdot n_\mathrm{r} \cdot (n_\mathrm{a} + 1) + n_\mathrm{o}$ integers.

A simple $(1+\lambda)$ evolutionary algorithm is used as a search mechanism. It means that CGP operates with the population of $1 + \lambda$ individuals (typically, λ is between 1 and 20). The initial population is constructed either randomly or by a heuristic procedure. Every new population consists of the best individual of the previous population (so-called parent) and its λ offspring. In each generation, an offspring with equal or better fitness than the parent's is chosen as the new parent. The offspring individuals are created using a point mutation operator which modifies up to h randomly selected genes of the chromosome, where h is a user-defined value. The algorithm is terminated when the maximum number of generations is exhausted or a sufficiently working solution is obtained.

There are two concurrently evolving populations in the proposed coevolutionary algorithm: (1) candidate programs evolving using CGP and (2) tests (*fitness cases subsets*, abb. FCSs) evolving using a simple genetic algorithm. Both populations evolve simultaneously and interact through the fitness function.

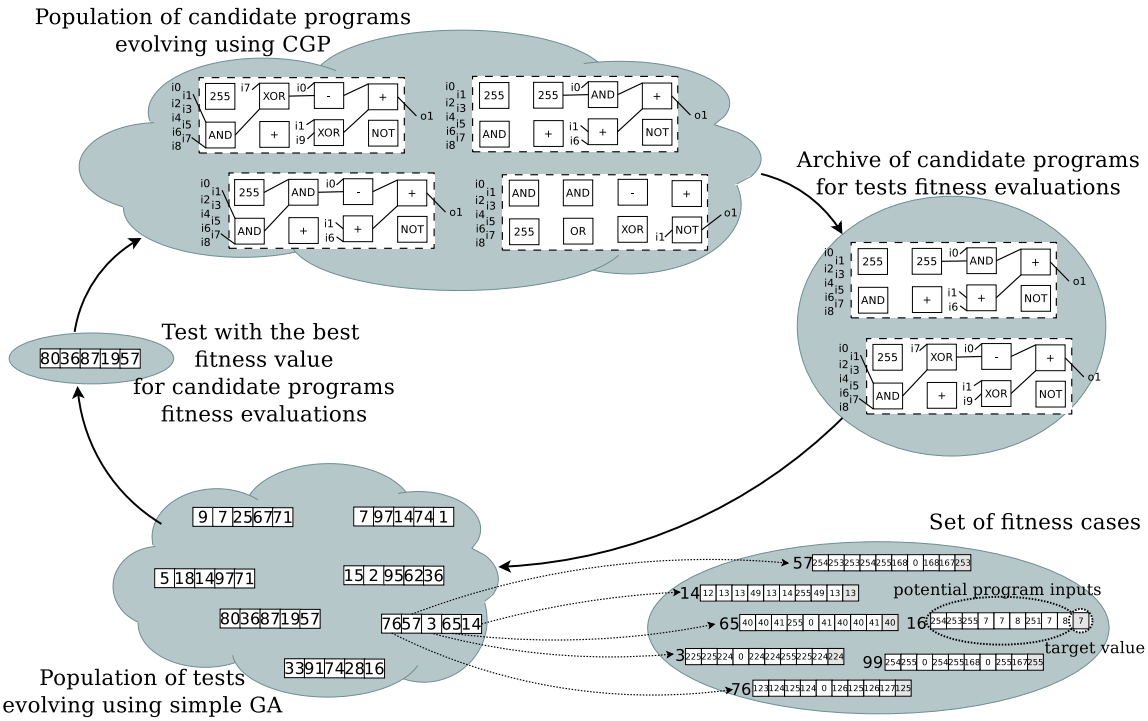


Figure 1: Populations in coevolutionary CGP – candidate programs and tests.

Test is a subset of the fitness cases set, therefore every test is encoded as a fixed-sized array of pointers to elements in the fitness cases set. In addition to one-point crossover and mutation, a randomly generated tests replacing the worst-scored tests in each generation has been used.

The aim of coevolving tests and candidate programs is to allow both candidate programs and tests to enhance each other automatically until a satisfactory problem solution has been found. Figure 1 shows the overall scheme of the proposed method. If the top-ranked candidate program fitness value (in the actual generation of candidate programs evolution) has changed against the previous generation, the top-ranked candidate program is copied to the archive of candidate programs. The archive of candidate programs is a circular list that is used for tests evaluation. Tests (in the tests evolution) are evaluated using candidate programs from the archive as follows. Each candidate program from the archive is executed for all fitness cases in the test. The test with the worst mean fitness value for candidate programs from the archive is selected as the top-ranked test in the actual generation. This test is then used to evaluate candidate programs in the candidate programs evolution. This fitness interaction approach allows to improve candidate programs using the fitness cases, which cannot be correctly solved by currently evolved candidate programs yet.

Hardware platform design

The evolutionary design includes two basic steps alternating in each generation – generation of new population and evaluation. Since the evaluation step consists in multiple running or simulating of candidate program and computing chosen fitness, a significant acceleration can be achieved by means of task or data parallelism, while the best throughput can be achieved using custom hardware.

On the contrary, the evolutionary process control is, by its nature, suitable rather for running on a universal processor, moreover in the case of CoEAs two evolutionary processes need to be executed in parallel with the ability to communicate with each other.

These requirements have been taken into account when choosing the target platform. Currently, two suitable alternatives are available – conventional FPGAs and a combination of a processor and programmable logic (e.g. Xilinx Zynq All Programmable SoC, Dobai and Sekanina (2013)). Table 1 compares several devices available in our institution as part of a development kit with respect to the configurable logic

Table 1: Target platforms comparison.

device	logic cells	block RAM
Virtex 6 XC6VLX240T	241,152	14,976 Kb
Virtex 7 XC7K325T	326,080	16,020 Kb
Zynq 7020 XC7Z020	85,000	4,480 Kb

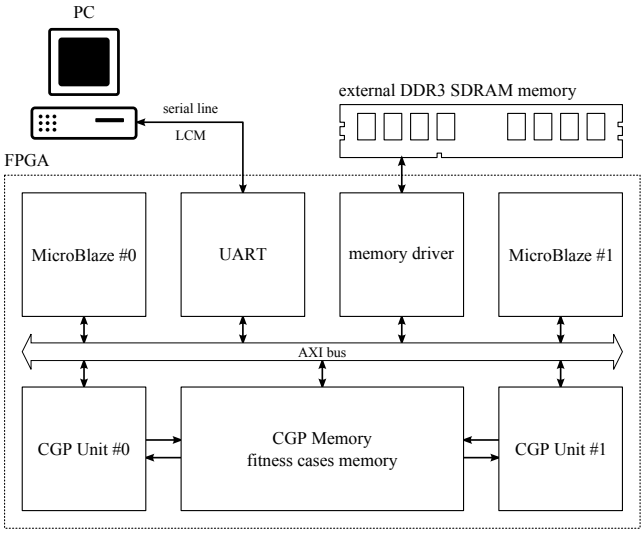


Figure 2: Hardware platform architecture.

cells count and the amount of block RAM. It is obvious, that the Zynq platform offers much less flexibility in terms of custom logic comparing to Virtex FPGA family. Therefore, a standard FPGA has been chosen as a more flexible option.

Despite the fact that standard FPGAs do not have hard processors, wide choice of soft processors under various licences are available. The most suitable choice for Xilinx devices is the MicroBlaze soft processor, offering sufficient performance while occupying a reasonable area. Figure 2 shows the proposed hardware platform architecture. The system consists of two MicroBlaze soft processors supplemented by two independent acceleration units (CGP Units) and fitness cases memory (CGP Memory). All components are interconnected by the AXI bus and additional memory channels are introduced for fitness cases transfers. Com-

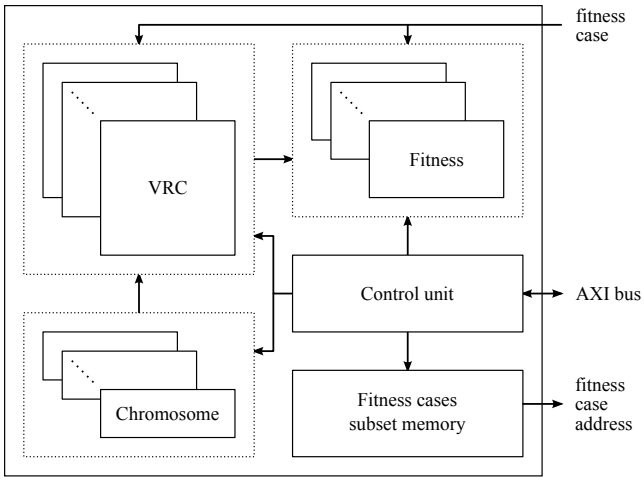


Figure 3: Detailed architecture of CGP Unit.

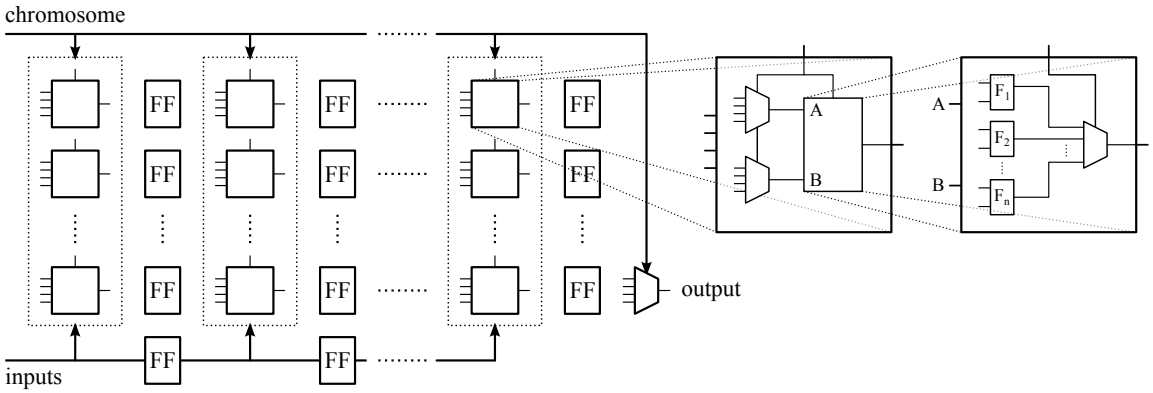


Figure 4: Virtual Reconfigurable Circuit (VRC).

munication with a service application running on a PC is performed through serial port (UART) and LCM communication library[1]. The dual MicroBlaze system utilizes AXI Mailbox component, which enables to pass simple messages between the processors (control and status messages, chromosomes, fitness values etc.).

CGP Unit (Figure 3) includes a set of subcomponents, each composed of one virtual reconfigurable circuit (VRC), fitness unit and chromosome register. Moreover, the CGP Unit includes a common control unit and FCS memory, each subcomponent is fed with the same data. The control unit is responsible for the communication between the MicroBlaze processor and the peripheral and for controlling the fitness computation. There are several configuration and status registers that are memory mapped on the AXI bus together with the test memory. By setting a specific bit in the control register, the fitness computation starts. Fitness cases are addressed indirectly using the test memory, which is addressed sequentially by the control unit. In the case of image filter design, each fitness case consists of chosen, e.g. 3×3 or 5×5, pixel neighbourhood from the noisy image and one pixel from the original image. The noisy pixels are processed in the VRCs and together with the clean pixel, properly delayed, come to the fitness unit. After a specified number of fitness cases is processed, the control unit saves the current fitness values and notifies the MicroBlaze processor by changing the status register value.

The VRC architecture is shown in Figure 4. According to the program representation in CGP, the VRC comprises a grid of nodes, called configurable function blocks (CFBs), interconnected in such a way that each block can access all other blocks in previous columns and the VRC inputs. Both VRC's inputs and CFB's outputs are registered and delayed, so that the VRC is fully pipelined while keeping the l-back parameter of arbitrary choice. Thanks to the pipelining, the VRC is able to process one fitness case per clock cycle.

[1]Lightweight Communications and Marshalling (LCM) is a set of libraries and tools for message passing and data marshalling originally designed by the MIT DARPA Urban Challenge Team.

Table 2: Functions implemented in CFBs according to Sekanina et al. (2011).

#	function	#	function
0	255	8	$i_1 \gg 1$
1	i_1	9	$i_1 \gg 2$
2	$\overline{i_2}$	10	$(i_1 \ll 4) \vee (i_2 \gg 4)$
3	$i_1 \vee i_2$	11	$i_1 + i_2$
4	$\overline{i_1} \vee i_2$	12	$i_1 +^s i_2$
5	$i_1 \wedge i_2$	13	$(i_1 + i_2) \gg 1$
6	$\overline{i_1 \wedge i_2}$	14	$\max(i_1, i_2)$
7	$i_1 \oplus i_2$	15	$\min(i_1, i_2)$

Each CFB has the same structure. The input data are selected using two multiplexers and forwarded to several functions (functions used for image filter design are listed in Table 2), the output value is selected by an output multiplexer. The configuration of the multiplexers is determined by specific genes of the chromosome.

The output of each VRC is connected to separate fitness unit (Figure 7). Two different fitness functions are computed simultaneously – squared and absolute error:

$$f_{\mathrm{sq}} = \sum_{i=1}^{N} (x_i - y_i)^2 \,, \qquad f_{\mathrm{abs}} = \sum_{i=1}^{N} |x_i - y_i| \,, \tag{1}$$

where x_i is the clean pixel, y_i the VRC output corresponding to the i-th fitness case and N the number of fitness cases. These fitness functions are very similar to the MSE and MDPP functions (commonly used for image filter design), except for normalization with the number of pixels N. Since division is a very demanding operation, its removal saves a lot of resources without any impact on the application in EAs. While performing experiments, one can choose which

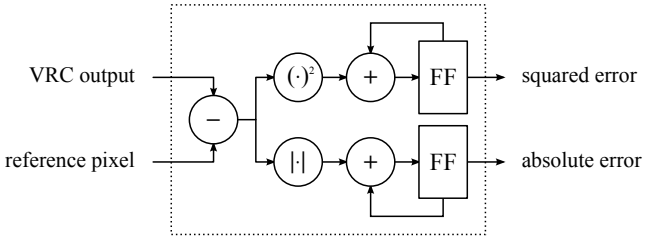


Figure 7: Fitness unit.

fitness function is used for the evaluation.

The CGP Memory component (Figure 8) is designed to achieve very high throughput. One write port (connected to the AXI bus) and two read ports enable to supply both CGP Units with data. These ports have different data widths because the AXI bus is 32 bit wide, but the width of the fitness case depends on the chosen pixel neighbourhood (80 bits for 3×3, 208 bits for 5×5). Therefore the memory has to be divided into 8 bit wide blocks and the read and write ports have to be treated in a different way. The fitness case (data output of the read port) is a concatenation of values from all these blocks from the same address. When writing from the AXI bus side, at most 4 blocks are updated at the same time. The total memory size depends on the chosen pixel neighbourhood and the maximum training image size we want to use. In our design, the number of fitness cases is limited to 65,536 due to fixed address width (16 bit), then the maximum memory capacity is $80 \cdot 65,536 \approx 5,243$ Kb for 3×3, respectively $208 \cdot 65,536 \approx 13,632$ Kb for 5×5 neighbourhood. Note that these sizes still fit into the Virtex devices, but not into the Zynq SoC (see Table 1).

Thanks to these hardware components, the fitness calculation is very efficient. The remaining steps of the evolutionary process (individuals manipulation, communication) take place on the MicroBlaze processors.

The evolutionary design is running as follows. At the beginning, original and noisy images are transfered to the external DDR3 memory, fitness cases are put together and copied to the CGP Memory. After that, the design process is initiated. Timing diagram in Figure 5 shows the steps of a single generation. The population is divided into N_{ch} chunks of P_{ch} individuals depending on the number of

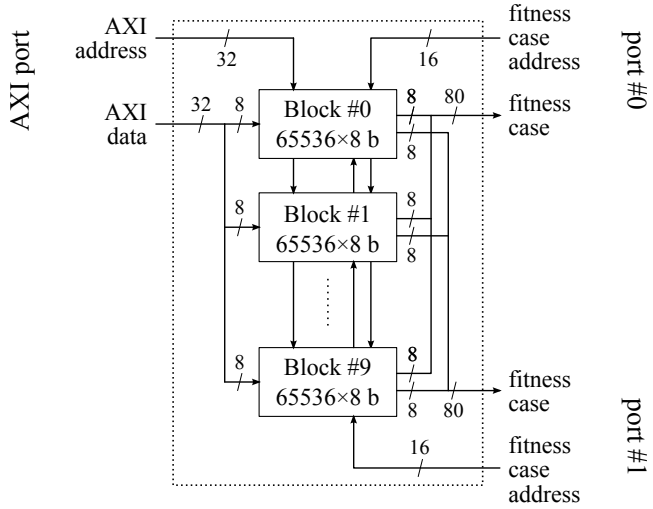


Figure 8: Architecture of CGP Memory.

VRCs N_{VRC} and the population size P:

$$N_{\mathrm{ch}} = \left\lceil \frac{P}{N_{\mathrm{VRC}}} \right\rceil, \quad P_{\mathrm{ch}} = \left\lceil \frac{P}{N_{\mathrm{ch}}} \right\rceil. \tag{2}$$

In each generation, individuals belonging to the first chunk need to be mutated and transfered to the CGP Unit before the fitness computation is executed. For every succeeding chunk (except for the last one including the last individuals of the population), the mutations and chromosome transfers can be overlapped with the fitness computation (all chromosome registers are shadowed). To achieve the best hardware utilization, the fitness computation time t_{f} has to be longer than the time t_{m} spent on the mutations and transfers. Ignoring some overhead, the total time per generation t_{g} is than:

$$t_{\mathrm{g}} = t_{\mathrm{m}} + (N_{\mathrm{ch}} - 1) \cdot \max(t_{\mathrm{m}}, t_{\mathrm{f}}) + t_{\mathrm{f}}. \tag{3}$$

Finally, when the evolution is completed, the best individual's chromosome is sent to the PC.

The coevolutionary design process is slightly more difficult, as it can be seen in Figure 6. The image filter evolution is running almost the same way except for the fitness cases subset, which is beeing evolved in parallel. For the purpose of FCSs evaluation, the best evolved filters are saved to an archive of candidate filters. The FCSs evolutionary process

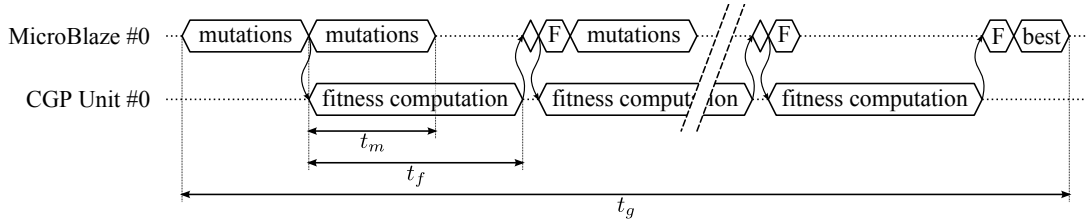


Figure 5: Timing diagram of the evolutionary process.

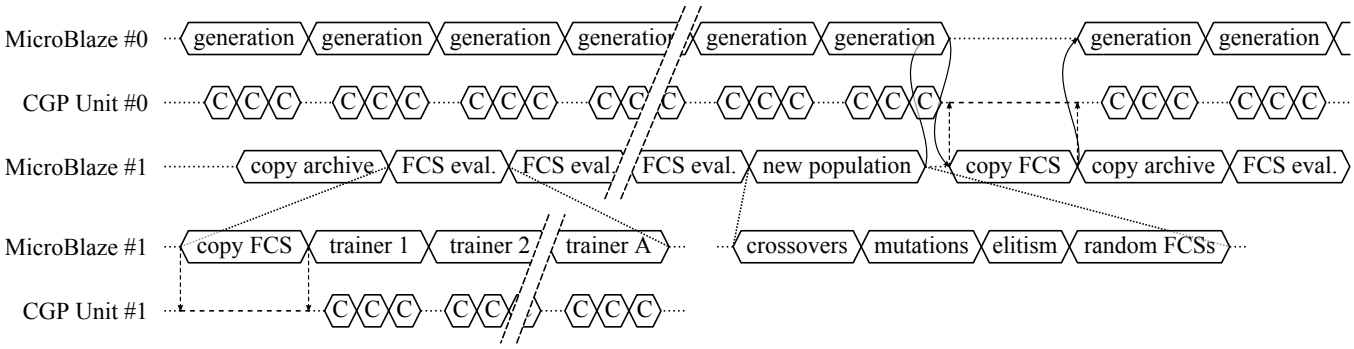


Figure 6: Timing diagram of the coevolutionary process.

is based on a simple genetic algorithm, the FCS chromosome is represented by a fixed-sized array of integers. In each generation, all FCS individuals are evaluated using all filters from the archive (trainers), the fitness value of the i-th individual is the mean value of the particular fitnesses:

$$f^i_{\mathrm{FCS}} = \frac{1}{A}\sum_{j=1}^{A} f(i,j), \tag{4}$$

where A is the archive size and $f(i,j)$ is the fitness (either squared error f_{sq} or absolute error f_{abs}) of the j-th trainer on the i-th FCS. After all FCSs are evaluated, new population is created using standard genetic operators. Specified number of individuals is obtained by one-point crossovers and the new individuals are mutated with some probability. In order to exert the selective pressure, elitism is introduced by keeping the best individual unchanged and making a few mutated clones. Finally, the rest of the population is generated randomly to preserve genetic variability. At the end of each generation, the filter evolutionary process is notified and at the right moment (after finishing the entire generation), the FCS is copied to the CGP Unit #0. No FCSs sharing between MicroBlaze processors is required.

Experimental results

This section presents benchmark problems, experimental setup and experimental evaluation of the proposed hardware accelerated approach and its comparison with the software approach.

In order to evaluate the proposed approach, salt-and-pepper noise filters were designed using standard CGP and coevolutionary CGP. This type of noise is characterized by noisy pixels with the value of either 0 or 255 (for 8-bit grayscaled images). The Lena training image with size 256×256 pixels was corrupted by 5%, 10%, 15% and 20% salt-and-pepper noise. The evolved filters were tested on 14 different images (Gonzalez et al., 2009) containing the same type of noise.

CGP was used according to Sekanina et al. (2011), i.e. $n_{\mathrm{c}}=8$, $n_{\mathrm{r}}=4$, $l=7$, $n_{\mathrm{i}}=9$, $n_{\mathrm{o}}=1$, $\lambda=19$, every node had two inputs, the number of mutations per new individual was $h=5$ and Γ contained the functions from Table 2. The archive of candidate programs had capacity of 20 elements.

FCSs were evolved using a simple GA, where 3-tournament selection, single point crossover and mutation up to 2 % of chromosome were used. Elitism and random individuals were used to exert selective pressure and preserve genetic variability. For the GA, various chromosome lengths were tested, particularly, 1.5625 %, 3.125 %, 6.25 %, 12.5 %, 25 % and 50 % of total number of fitness cases in the training set. For each FCS size, 100 independent runs were performed and the evolution/coevolution was terminated after 100,000 generations of CGP.

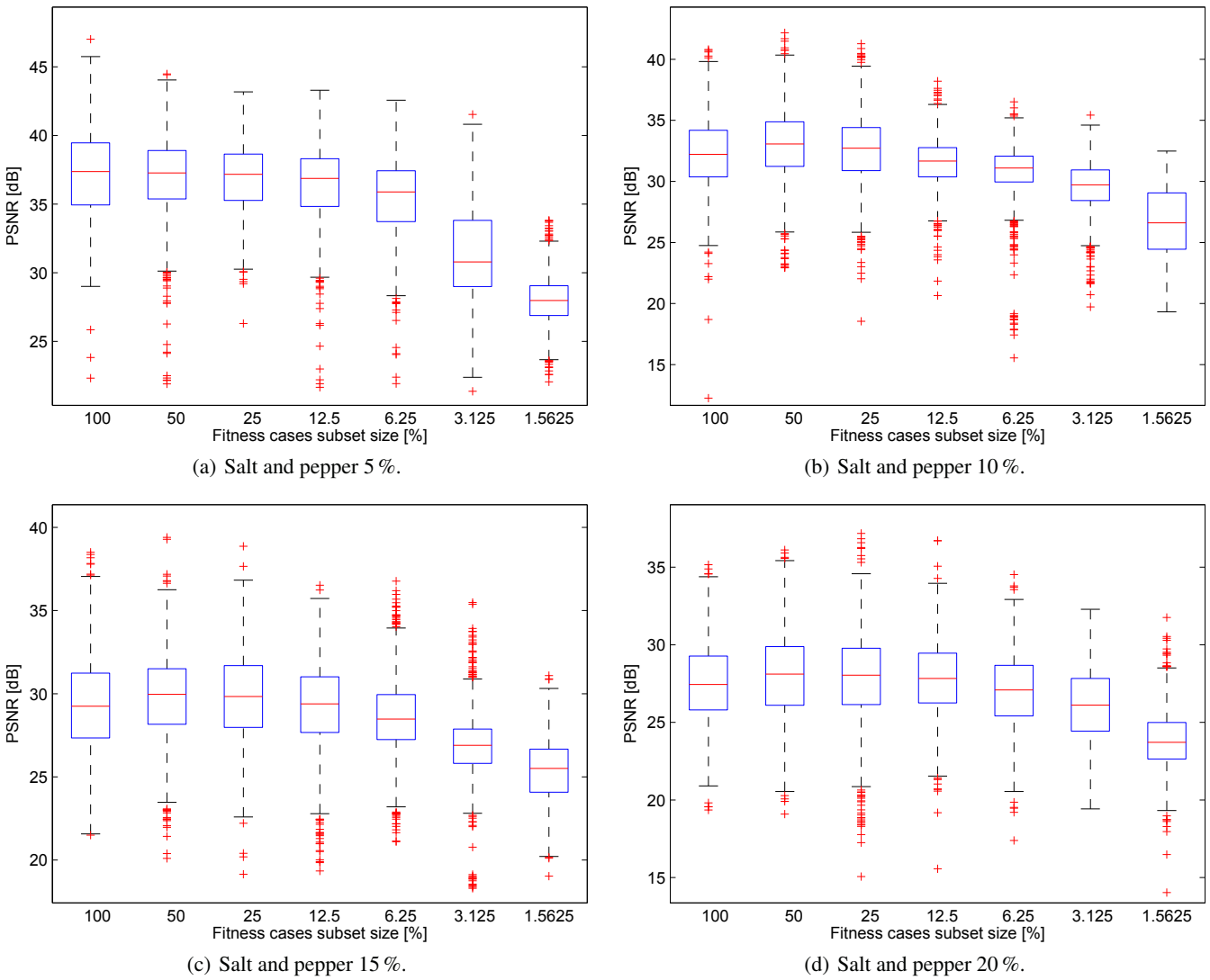
The proposed coevolutionary algorithm accelerated using FPGA was compared with the standard CGP algorithm in terms of filtering quality of evolved filters and with the higly optimized coevolution implementation running on an ordinary processor in terms of the execution time.

The quality of filtering was expressed using a measure typically used in the image processing community – as a peak signal-to-noise ratio (PSNR):

$$\mathrm{PSNR}(x,y) = 10\log_{10}\frac{255^2}{\frac{1}{MN}\sum_{i,j}(x(i,j)-y(i,j))^2}, \tag{5}$$

where $M\times N$ is the size of the image, x denotes the original image, y the filtered image and i,j are indexes of a pixel in the image. Figure 9 shows that using coevolutionary CGP running on an FPGA we are able to evolve image filters of comparable (or better) quality than standard CGP for all noise intensities. Furthermore, the higher the noise intensity, the smaller fitness cases subset can be used to get acceptable results.

Software and hardware performance comparison for standard CGP can be found in Table 3. The software imlementation is a command line tool written in C++ utilizing OpenMP library for running in multiple threads and SSE 4.1 instruc-



(a) Salt and pepper 5 %.

(b) Salt and pepper 10 %.

(c) Salt and pepper 15 %.

(d) Salt and pepper 20 %.

Figure 9: PSNR statistics calculated from 100 evolved filters (100 independent runs using the Lena training image for each noise intensity and each FCS size) for the 14 test images.

tions enabling to process 16 fitness cases in a single step. Before evaluation, each chromosome is analyzed to exclude the inactive nodes. The performance tests were performed on the Intel Core i7-860 processor (2.8 GHz), allowing 8 threads to be running simultaneously. The hardware platform configuration was as follows: 7 VRCs in the CGP Unit #0, 6 VRCs in the CGP Unit #1, in total $N_{\mathrm{VRC}} = 13$, the entire system was running on 100 MHz frequency. Despite a very efficient software implementation and powerful processor, the hardware implementation overcomes the SW version significantly. The bigger the population, the higher the acceleration, while the most advantageous choice of the population size is a multiple of the VRC count.

The coevolutionary design performance of the hardware platform was compared to a software implementation, again optimized using OpenMP and SSE 4.1 instructions. Because of two evolutionary processes running in parallel and very poor data locality, the performance of the software implementation was vastly degraded. Therefore the speed-up

Table 3: Hardware platform evolutionary process performance (10,000 generations, image size 256×256 pixels, 1-5 mutations per chromosome) obtained by running 100 independent tests for each population size.

population size	5	10	15	20	25
SW time (s)	30.83	74.65	122.39	183.40	233.71
HW time (s)	7.21	7.86	14.17	14.44	14.83
acceleration	4.28	9.50	8.63	12.70	15.77

Table 4: Coevolutionary design performance.

FCS size	50 %	25 %	12.5 %
SW time (s)	713.23	405.47	223.18
HW time (s)	12.51	6.91	4.23
acceleration	56.99	58.64	52.82
FCS size	6.25 %	3.125 %	1.5625 %
SW time (s)	133.91	88.26	71.92
HW time (s)	3.57	4.20	3.97
acceleration	37.49	21.04	18.11

is much more significant in the case of coevolutionary design. Table 4 shows performance tests results for software and hardware approaches. The experimental setup was as follows: 10,000 generations, population of 20 individuals, image size 256×256 pixels, 1-5 mutations per CGP chromosome, up to 2 % mutations per FCS. Note that for FCS sizes lower than 12.5 %, the evolution time is similar. Due to very low fitness cases count, the fitness computation time t_{f} is shorter than the mutations time t_{m} and hardware utilization goes down. Moreover, the FCS evolution runs faster due to lower overhead and the FCS is updated more often. That is why the computation time can surprisingly grow with FCS size decrease.

Conclusions

In this paper, a hardware platform for coevolutionary CGP speed-up based on FPGA technology has been proposed. Two-population coevolutionary algorithm running on dual MicroBlaze soft processor system has been accelerated using custom peripheral based on virtual reconfigurable circuit approach. The full pipelined VRC along with a special fitness cases memory enables very efficient fitness calculation. The performance of the hardware was experimentaly evaluated in the task of evolutionary image filter design. It was shown that using custom hardware, universal processor thoughput can be greatly overcome in the task of the evolutionary design and even more in the coveolutionary case. Various sizes of fitness cases subset have been applied to demostrate the coevolutionary approach benefits. Especially for higher noise intensities, reduction of the FCS size leads to better results.

With small modifications, the hardware platform can be used to effectively evolve other digital circuits using coevolutionary CGP. In our future work, we will focus on designing image filters for other noise types as well as other image transformations, combinational logic design and other tasks suitable for coevolutionary design.

Acknowledgements

This work was supported by the Czech science foundation project P103/10/1517 and the BUT project FIT-S-11-1.

References

Dobai, R. and Sekanina, L. (2013). Towards evolvable systems based on the Xilinx Zynq platform. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, pages 89–95. IEEE Computational Intelligence Society.

Gonzalez, R. C., Woods, R. E., and Eddins, S. L. (2009). "Standard" test images. *ImageProcessingPlace.com*. [online]. http://www.imageprocessingplace.com/.

Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42(1):228–234.

Imamura, K., Foster, J. A., and Krings, A. W. (2000). The Test Vector Problem and Limitations to Evolving Digital Circuits. In *Proc. of the Second NASA/DoD Workshop on Evolvable Hardware*, pages 75–79. IEEE Computer Society.

Miller, J. F., editor (2011). *Cartesian Genetic Programming*. Natural Computing Series. Springer Verlag.

Popovici, E., Bucci, A., Wiegand, R., and De Jong, E. (2012). Coevolutionary principles. In *Handbook of Natural Computing*, pages 987–1033. Springer Berlin Heidelberg.

Schmidt, M. D. and Lipson, H. (2008). Coevolution of Fitness Predictors. *IEEE Transactions on Evolutionary Computation*, 12(6):736–749.

Sekanina, L. (2003). *Evolvable Components - From Theory to Hardware Implementations*. Natural Computing Series. Springer Verlag.

Sekanina, L., Harding, S. L., Banzhaf, W., and Kowaliw, T. (2011). Image processing and CGP. In *Cartesian Genetic Programming*, pages 181–215. Springer Verlag.

Sikulova, M. and Sekanina, L. (2012a). Acceleration of evolutionary image filter design using coevolution in cartesian GP. In *Parallel Problem Solving from Nature - PPSN XII*, LNCS 7491, pages 163–172. Springer Verlag.

Sikulova, M. and Sekanina, L. (2012b). Coevolution in cartesian genetic programming. In *Genetic Programming*, LNCS 7244, pages 182–193. Springer Verlag.

Vanneschi, L. and Poli, R. (2012). Genetic programming – introduction, applications, theory and open issues. In *Handbook of Natural Computing*, pages 709–739. Springer Berlin Heidelberg.

Vasicek, Z. and Sekanina, L. (2010). Hardware accelerator of cartesian genetic programming with multiple fitness units. *Computing and Informatics*, 29(6):1359–1371.

Vasicek, Z. and Sekanina, L. (2011). Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327.