Computing and Informatics, Vol. 29, 2010, 1359–1371

HARDWARE ACCELERATOR OF CARTESIAN GENETIC PROGRAMMING WITH MULTIPLE FITNESS UNITS

Zdeněk Vašíček, Lukáš Sekanina

Faculty of Information Technology
Brno University of Technology
Božetěchova 2
612 66 Brno, Czech Republic
e-mail: {vasicek, sekanina}@fit.vutbr.cz

Revised manuscript received 11 May 2010

Abstract. A new accelerator of Cartesian genetic programming is presented in this paper. The accelerator is completely implemented in a single FPGA. The proposed architecture contains multiple instances of virtual reconfigurable circuit to evaluate several candidate solutions in parallel. An advanced memory organization was developed to achieve the maximum throughput of processing. The search algorithm is implemented using the on-chip PowerPC processor. In the benchmark problem (image filter evolution) the proposed platform provides a significant speedup (170) in comparison with a highly optimized software implementation. Moreover, the accelerator is 8 times faster than previous FPGA accelerators of image filter evolution.

Keywords: Cartesian genetic programming, hardware accelerator, evolutionary circuit design, FPGA

1 INTRODUCTION

Evolutionary algorithms (EAs) are capable of creating human-competitive inventions automatically [1]. However, the computational power which EA needs for obtaining innovative results is enormous for most applications. In order to reduce the computational time, various methods have been proposed. Domain-specific hardware accelerators represent a very promising solution due to the high performance,

low implementation cost and low power consumption. In addition to application-specific chips such as [2], field programmable gate arrays (FPGAs) have been utilized [3, 4, 5, 6, 7]. Modern FPGAs provide a cheap, flexible and powerful platform, often outperforming common workstations or even clusters of workstations in particular applications.

It has been demonstrated in our previous work that a single-chip FPGA-based accelerator running at 100 MHz can provide approx. 44 times higher performance in comparison with a common PC running at 2.4 GHz [8]. The speedup was obtained in the task of image filter evolution which can be considered as a symbolic regression problem. The accelerator utilizes a hardware implementation of Cartesian genetic programming (CGP) which is a special kind of genetic programming developed mainly for digital circuit evolution [9].

In this paper, a new FPGA-based accelerator of CGP is proposed. The goal of this work is to provide a high-performance as well as low-power evolutionary platform for acceleration of symbolic regression problems (over the integer representation). The main contribution can be seen in the utilization of multiple fitness evaluation units and new population memory architecture. The parallel computing approach combined with deep pipelining of fitness units enabled to accelerate the image filter evolution 170 times in comparison with a common PC running at 2.4 GHz.

The paper is organized as follows. Section 2 introduces the idea of Cartesian genetic programming. In Section 3 the architecture of the proposed accelerator is presented. Section 4 is devoted to experimental evaluation of the accelerator. Finally, conclusions are given in Section 5.

2 CARTESIAN GENETIC PROGRAMMING

Cartesian genetic programming was introduced by Miller and Thompson in 2000 [9]. CGP represents candidate programs as graphs consisting of an array of programmable nodes.

More precisely, a candidate program is modeled as an array of u (columns) $\times$ v (rows) of programmable elements (nodes). The number of inputs, n_i, and outputs, n_o, is fixed. Feedback is not allowed. In order to define the level of connectivity, so-called L-back parameter is used. Each node input can be connected either to the output of a node placed in the previous L columns or to some of primary inputs. Each node is programmed to perform one of the functions defined in the set Γ. Every individual is encoded using $u \times v \times 3 + n_o$ integers. The first $u \times v$ triplets encode the configuration of the CGP nodes (i.e. connection of their inputs and their functions), the last n_o-tuple encodes the connection of the primary outputs. While the size of chromosome is fixed, the size of phenotype is variable since some nodes need not to be used. This represents the main advantage of CGP (when compared to the GP) and allows to make an effective hardware accelerator.

CGP operates with the population of $1 + \lambda$ individuals (typically, $\lambda = 4$). The initial population is randomly generated. Every new population consists of the best individual and its λ offspring created using a point mutation operator. In case when two or more individuals have received the same fitness score in the previous population, the individual which did not serve as a parent in the previous population will be selected as a new parent.

For the symbolic regression problems, a training set is used in the fitness function. In case of image filter evolution a training image corrupted by a given type of noise is employed together with uncorrupted version of the same image [10]. The goal is to minimize the difference between the output of a candidate filter and the uncorrupted image. The evolution is stopped when the best fitness value stagnates or the maximum number of generations is achieved.

3 PROPOSED CGP ACCELERATOR

The basic idea of the CGP accelerator is that a given instance of CGP array (i.e. a reconfigurable array consisting of $u \times v$ programmable nodes) is implemented as a reconfigurable circuit in the FPGA. Its configuration is defined using a bitstream which is stored in a configuration register implemented also in the FPGA. This concept is called the virtual reconfigurable circuit (VRC) [10]. This work extends our original implementation [5, 8, 11].

3.1 Overview of the Architecture

The proposed CGP accelerator is completely implemented in a single FPGA and consists of Genetic Unit (GU), Fitness Unit (FU) and Control Unit (CU) (see Figure 1). Training data are stored in external SRAM memories. The GU as well as FU are connected to the internal FPGA bus which provides an effective communication interface between FPGA and PCI bus. The host PC is used to load training data, read the results, and define the parameters of CGP.

In order to maximize the overall performance, the CU plays the role of master, controls the entire system and provides an interface to the host PC. The PowerPC generates a new candidate individual when a request is issued. The instruction memory of the PowerPC is implemented using BRAMs; however, our search algorithm is completely executed from an instruction cache.

The population of candidate configurations is also stored in on-chip BRAM memories. The population memory is divided into N_b banks; each of them contains N_c configuration bitstreams. Each bitstream consists of the configuration data that are necessary to configure one VRC. All the bitstreams stored within a bank are evaluated in parallel. An additional bit (associated with every bank) determines the data validity; only valid configurations can be evaluated. In order to overlap the evaluation of a candidate configuration with generating a new candidate configuration, at least two memory banks have to be utilized. While the candidate solutions

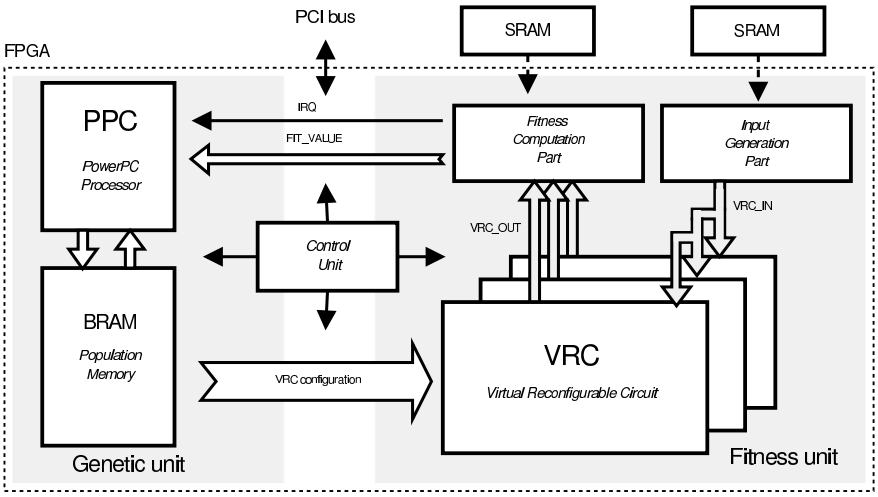

Fig. 1. Architecture of the proposed CGP accelerator

are evaluated, the N_c new candidate configurations are generated. The population memory provides two independent ports:

1. the 32-bit read/write port A connected to the PowerPC processor and
2. the m-bit read-only port B connected to the fitness unit used for the reconfiguration of VRCs.

Since corresponding columns of VRCs are reconfigured at the same time (i.e. in parallel), the part of bitstream which encodes one column of VRC can contain up-to m/N_c bits. Note that the width of the B port must be chosen with respect to

1. the implementation limits (m must be an integer divisible by 128),
2. the number of bits of a part of bitstream used to configure one column of VRC and
3. the number of VRC instances N_c (in our case $m = 256$ and $N_c = 4$).

The CU consists of two subcomponents working concurrently. The first subcomponent reconfigures the VRCs according to the configuration stored in the population memory. The second subcomponent is responsible for sending the fitness value to the PowerPC processor. As soon as the fitness value is valid, an interrupt request (IRQ) is generated to activate a service routine of the PowerPC. In this routine, PowerPC reads the fitness value together with some additional data and new candidate configurations are generated for the given bank. The PowerPC processor acknowledges the interrupt and sets up the validity bit. The PowerPC processor utilizes the Mersenne Twister algorithm to generate pseudorandom numbers. This algorithm was chosen on the basis of comparisons performed in [8, 12].

3.2 Fitness Unit

The fitness unit consists of N_c instances of VRC and two subcomponents:

1. the input generation part and
2. the fitness computation part.

The training data are stored in external SRAM memories. The fitness unit loads training data from the external SRAM1 memory and forwards them to the inputs of VRCs.

In case of the evolutionary design of image filters it is necessary to implement a local neighborhood function (also referred to as a sliding window function) producing wk^2 bits per one clock cycle that have to be forwarded to the inputs of VRCs, where k is the size of the filter window and w is the data width (in our case $k = 3$ and $w = 8$). The local neighborhood function can be efficiently implemented using k row buffers as shown in Figure 2.

In case of common one-dimensional symbolic regression problems, the training data can be forwarded directly from the SRAM1 to the VRC inputs. In case that the problem to be solved involves the utilization of a history of previous samples, the input generation part of the fitness unit will contain a buffer for previous samples. This buffer can be implemented using registers or BRAM memories.

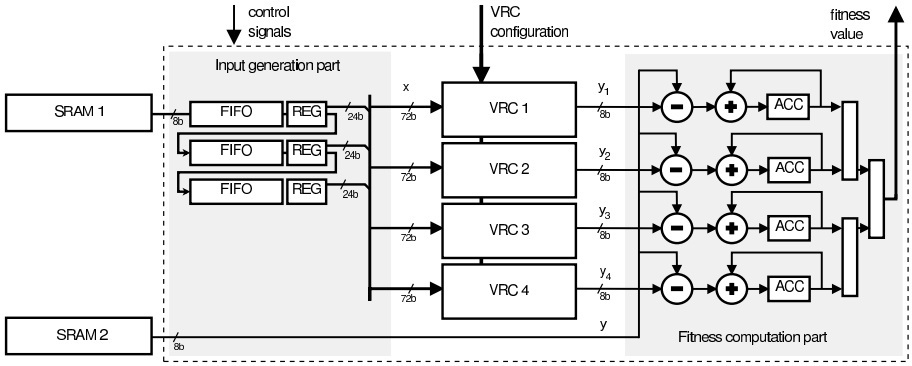

Fig. 2. Fitness Unit ($N_c = 4$)

The fitness computation part consists of N_c instances of a circuit that computes the fitness value; each VRC utilizes its own instance. In this paper, four VRCs with k^2 inputs and one output are used. For each VRC i, the absolute difference between the output value y_i and the required output value y (which is obtained from the external memory SRAM2) is calculated. Then, a temporary fitness value stored in accumulator (ACC_i) is updated by the difference $|y_i - y|$. As soon as FU evaluates the last training vector, the best fitness value together with the index of corresponding VRC is sent to the PowerPC. VRCs are then reconfigured using new bitstreams.

3.3 Virtual Reconfigurable Circuit

The VRC consists of Configurable Logic Blocks (CFBs) placed in a grid of u columns and v rows (see the example in Figure 3). Any CFB can be configured to implement one of 16 functions from Γ, where Γ includes addition, subtraction, shift, minimum, maximum, absolute difference and 10 logic functions (such as in [8]). All these functions operate with two 8-bit operands and produce a single 8-bit result. The operands are selected using two multiplexers. Each multiplexer connects the CFB either with a primary circuit input or the output of a CFB, which is placed in the preceding column. The reconfiguration is performed column by column, one column is configured in a single clock cycle. The computation is pipelined; a column of CFBs represents a stage of the pipeline. Registers are inserted between the columns in order to synchronize the input pixels with CFB outputs. Evolutionary algorithm directly operates with the configurations of the VRC; simply, a configuration is considered as a chromosome.

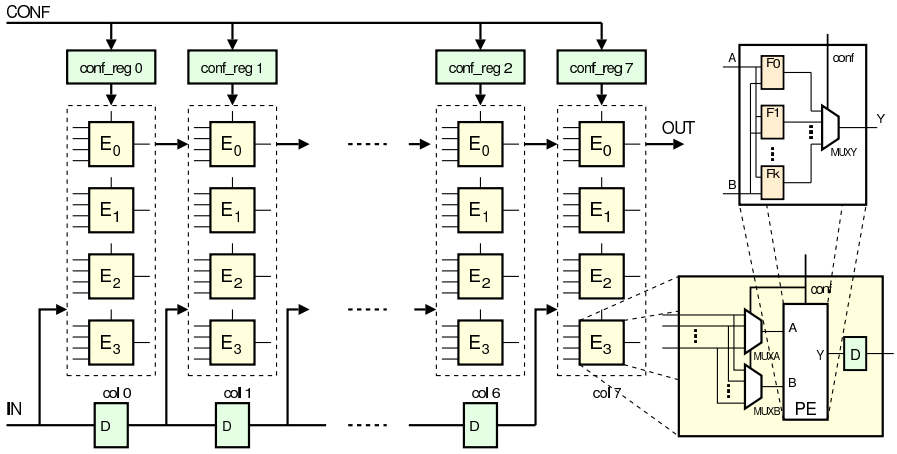

Fig. 3. VRC for symbolic regression problems

3.4 Genetic Unit

The introduction of multiple VRC instances requires to design a problem specific memory interface that allows avoiding the idle clock cycles. The memory banks are used in order to overlap the evaluation of the candidate solutions with the generation of new chromosomes. Moreover, each bank is divided into N_c equivalent sections, each of them is used to configure a single VRC. The population memory consists of several instances of BRAM memories arranged together to provide the required number of bits. This arrangement enables to reconfigure all VRC instances in parallel. In order to reduce the number of memory accesses issued by the PowerPC

processor, the population memory is equipped with a logic that enables to store only the differences between the configurations of neighboring sections.

In order to exploit the performance of proposed highly-parallel architecture, GU has to generate N_c new candidate configurations while another N_c candidate configurations are evaluated. Because the search algorithm utilizes a population of candidate solutions, a single genetic operator is used (i.e. mutation which inverts h bits of the configuration) and no crossover operator is applied, the number of memory accesses can be minimized by storing the differences between the configuration bitstream of the first offspring and of the remaining offspring.

The PowerPC keeps only the information about mutations (i.e. indices of inverted bits) and the best fitness value. FU contains a circuit generating a complete configuration bitstream for each VRC according to the partial information stored in the sections.

The mechanism controlling the bitstream generation works as follows. As soon as the evaluation is finished, the best fitness value f_{best} (out of the four evaluated individuals) together with the index of the corresponding VRC i is sent to the PowerPC. The three situations can occur

1. if $f_{best} < f_{parent}$ then the bitstream of the first mutant is reverted to the parent bitstream by applying the mutations leading to this configuration, however in reverse order,
2. if $i > 1$ then the differences between the first mutant and the i^{th} mutant stored in the i^{th} section have to be reflected to the first bitstream,
3. if $i = 1$ then nothing has to be done; the configuration bitstream corresponds with the new parent bitstream.

By applying the previous steps, the first section contains the parental bitstream and a new generation can be created. Note that the inverted bits stored in sections have to be cleared before a new generation is created. The same principle is applied for the remaining banks.

4 EVALUATION

In order to evaluate the performance of the proposed solution, the problem of evolutionary design of image filters will be investigated.

We will consider VRC that consists of 8 columns and 4 rows ($u = 8$ and $v = 4$). The configuration bitstream which is used to configure one VRC consists of 384 bits; i.e. 48 bits per a column are used. A single CFB is configured by 12 bits, 4 bits are used to select the connection of a single input, 4 bits are used to select one of the 16 functions. The population memory consists of 8 BRAM memories that provide 256 bit wide output. Hence a VRC with the configuration bitstream containing up to 64 bits per column can be used.

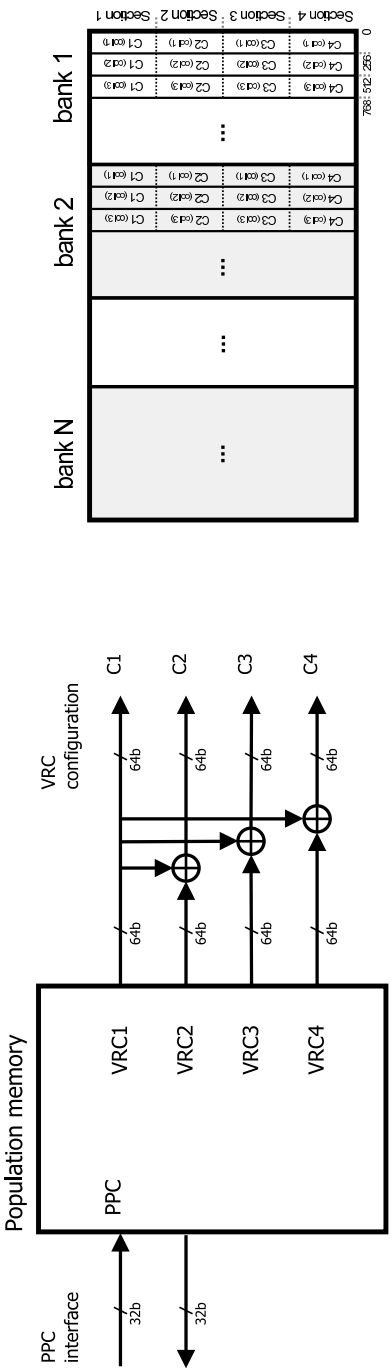

Fig. 4. Population memory and its internal organization

4.1 Theoretical Performance

Due to the pipelined reconfiguration as well as execution of VRC, the evaluation of N_c candidate programs requires $(M-2)(N-2)$ clock cycles, where $M \times N$ is the number of pixels of training image. The time t_{eval} needed to evaluate N_c candidate solutions can be expressed as

$$t_{eval} = (M-2)(N-2)\frac{1}{f} = (M-2)(N-2)\frac{1}{100}\mu s,$$

where f is the operation frequency ($f = 100\,\text{MHz}$). Since the generation of new candidate configurations is overlapped with the evaluation of other candidate solutions, the total time t_{total} can be expressed as

$$t_{total} = t_{init} + N_g \left\lceil \frac{p}{N_c} \right\rceil t_{eval},$$

where t_{init} corresponds with the time needed for the initialization (i.e. transferring the training data and programming the PowerPC processor), N_g is the number of generations, p is the population size and N_c is the number of VRC instances. The proposed platform provides the best performance if the number of VRC instances is equal to the population size or the population size is a multiple of the number of VRC instances ($p = kp$, where $k \in \mathbb{N}^+$). If the previous condition is met, all the VRC instances are utilized without stalls. Note that this condition does not represent any limitation since the population size is typically chosen between five and ten individuals and, moreover, the population size can be adjusted according to the number of utilized VRC instances.

4.2 Results of Synthesis

In order to implement the proposed system, a COMBO6X card equipped with Virtex II Pro XC2VP50 FPGA has been used. The evolvable system was described in VHDL, simulated using ModelSim and synthesized using Mentor Graphics Precision RTL 2009 and Xilinx ISE 10.1 tools. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic works at 100 MHz. Results of synthesis for the accelerator containing up to four VRC instances (4×8 CFBs each) are summarized in Table 1. According to the synthesis report, one instance of VRC occupies approx. 3 275 SLICEs and 1 084 DFFs. The whole design occupies approx. 60 % of the FPGA for $N_c = 4$, the four VRC instances represent approx. 90 % of the design size.

Table 2 summarises the results of synthesis for the XC5VFX100T FPGA which represents the modern Virtex-5 family. This FPGA will be available on the second generation of the COMBO cards (COMBO-LXT). The main difference between Virtex-5 and Virtex II Pro family is the internal structure of the basic building blocks (LUTs); while the Virtex II Pro chip contains 4-input LUTs the Virtex-5

resource	avail.	$N_c = 1$		$N_c = 2$		$N_c = 4$	
IO blocks	852	602	70 %	602	70 %	602	70 %
BRAM	232	16	7 %	16	7 %	16	7 %
SLICES	23 616	4 651	20 %	7 961	34 %	14 582	60 %
DFF	49 788	3 536	7 %	4 691	9 %	7 001	14 %

Table 1. Results of synthesis for various number of VRC instances (Virtex II Pro)

chip utilizes LUTs with 6 inputs. Moreover, the Virtex-5 family is equipped with more powerful PowerPC processor, faster logic and larger BRAM memories. Thus a well written design usually works on higher frequency and occupies smaller area.

resource	avail.	$N_c = 1$		$N_c = 2$		$N_c = 4$		$N_c = 8$	
IO blocks	640	640	94 %	640	94 %	602	94 %	602	94 %
BRAM	228	8	4 %	8	4 %	8	4 %	12	5 %
SLICES	16 000	1 828	11 %	3 157	20 %	5 819	36 %	11 158	70 %
DFF	65 280	3 633	6 %	4 788	7 %	7 098	11 %	11 718	18 %

Table 2. Results of synthesis for various number of VRC instances (Virtex-5, 100 MHz)

According to the synthesis report, one instance of VRC occupies approx. 1 290 SLICEs and 1 084 DFFs. The whole design occupies approx. 40 % of the FPGA for $N_c = 4$. The number of occupied resources indicates that this FPGA is able to contain approximately 2.5 times higher number of VRC instances and thus to provide 2.5 times higher computational power with nearly the same power consumption.

4.3 Experimental Evaluation

Experimental results show that approximately 25,000 candidate filters can be evaluated per second when the training set consists of 15 876 8-bit vectors (i.e. a training image containing 128×128 pixels is used) and four instances of VRC are employed. Table 3 contains the comparison of the proposed accelerator against the recently published works dealing with the evolutionary design of image filters in terms of the number of evaluated candidate solutions per second as well as the estimated power consumption. Note that the number of evaluations per second has been calculated for the image containing 128×128 pixels. The last column of Table 3 contains the relative speedup. It can be seen that the proposed solution works approximately 170 times faster than the highly optimized software version of the same algorithm written in C running at the Celeron 2.4 GHz processor. Note that even if only a single fitness unit operating at 100 MHz is utilized (third row of Table 3), the evolution is approx. 20 times faster than this software implementation running at a GHz processor.

Estimated results indicate that a cluster of 30 FPGAs will have the same power consumption as a common processor (65 W). Nevertheless, the cluster is capable of

providing the speedup of more than 5 000 supposing that one independent run of CGP is carried out using one FPGA.

Approach	**Platform**	**clock freq.**	**power cons.**	**evals per sec**	**speedup**
Proposed HW accelerator (4 VRCs)	FPGA XC2VP50	100 MHz	2 W	25 195	1
Complete HW accelerator [11]	FPGA XC2V3000	50 MHz	1 W	3 150	8
HW accelerator with PowerPC [8]	FPGA XC2VP50	50 MHz	2 W	3 150	8
Complete HW accelerator [7]	FPGA XCV2000	33 MHz	1 W	1 935	13
Muli-VRC accelerator [13]	FPGA XCV2000	30 MHz	1 W	1 935	13
Highly optimized SW [8]	CPU Celeron	2.4 GHz	65 W	145	170
SW [7]	CPU Pentium IV	2.0 GHz	60 W	16	1 495

Table 3. Comparison of the proposed accelerator with published approaches

Apart from the FPGA-based accelerators, several papers have been published in recent years dealing with the acceleration of CGP using common GPUs [14, 15, 16]. Harding and Banzhaf achieved the speedup between 0.02 and 100 for the problem of symbolic regression using the GPU NVidia GeForce 7300 GO [14]. Direct comparison between the results is difficult, as they used extremely large CGP array (10 000 nodes) and relatively small number of training vectors (2 000) in order to reduce the huge overhead arising during the data transfer to the GPU or accessing the content of the GPU memory. Another common approach to increase the speedup on GPUs is to introduce higher level of parallelism by increasing the number of individuals in the population [15, 16]. Although this approach enables to overlap the expensive data transfers with the evaluation of other individuals, the method seems to be unpractical. According to the published works population-parallel approaches appear to be more effective for smaller data sets but unable to compete with the FPGA-based accelerators on very large data sets.

5 CONCLUSION

In this paper, a new parallel and pipelined hardware architecture was presented for the acceleration of symbolic regression problems. The proposed architecture contains multiple instances of virtual reconfigurable circuit to evaluate several candidate solutions in parallel. An advanced memory organization was developed to achieve the maximum throughput of processing. The accelerator was implemented in the FPGA and its performance was compared with a software implementation and various GPU-based solutions. In the benchmark problem (image filter evolution) the proposed platform provides a significant speedup (170) in comparison with a highly optimized software implementation.

Acknowledgment

This work was partially supported by the Grant Agency of the Czech Republic under contract No. GA102/07/0850 Design and hardware implementation of a patent-invention machine, No. GD102/09/H042 Mathematical and Engineering Approaches

to Developing Reliable and Secure Concurrent and Distributed Computer Systems, the BUT grant FIT-10-S-1 and the Research Plan No. MSM 0021630528 Security-Oriented Research in Information Technology.

REFERENCES

[1] Koza, J. R.—Keane, M. A.—Streeter, M. J.—Mydlowec, W.—Yu, J.—Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers 2003.

[2] Sakanashi, H.—Iwata, M.—Higuchi, T.: EHW Applied to Image Data Compression. In Higuchi, T., Liu, Y., Yao, X. (Eds.): Evolvable Hardware, Springer 2006, pp. 19–40.

[3] Shackleford, B.: A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine. Genetic Programming and Evolvable Machines, Vol. 2, 2001, No. 1, pp. 33–60.

[4] Tufte, G.—Haddow, P.: Prototyping a GA Pipeline for Complete Hardware Evolution. In Stoica, A., Keymeulen, D., Lohn, J. (Eds.): Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, USA, IEEE Computer Society 1999, pp. 143–150.

[5] Vašíček, Z.—Sekanina, L.: An Evolvable Hardware System in Xilinx Virtex II Pro FPGA. International Journal of Innovative Computing and Applications, Vol. 1, 2007, No. 1, pp. 63–73.

[6] Glette, K.—Torresen, J.—Yasunaga, M.—Yamaguchi, Y.: On-Chip Evolution Using a Soft Processor Core Applied to Image Recognition. In: The 1st NASA/ESA Conference on Adaptive Hardware and Systems, Los Alamitos, CA, USA, IEEE Computer Society 2006, pp. 373–380.

[7] Wang, J.—Chen, Q. S.—L. C.: Design and Implementation of a Virtual Reconfigurable Architecture for Different Applications of Intrinsic Evolvable Hardware. IET computers and digital techniques, Vol. 2, 2008, No. 5, pp. 386–400.

[8] Vašíček, Z.—Sekanina, L.: Evaluation of a New Platform for Image Filter Evolution. In: Proc. of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems, IEEE Computer Society 2007, pp. 577–584.

[9] Miller, J.—Thomson, P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming EuroGP2000, Volume 1802 of LNCS, Springer 2000, pp. 121–132.

[10] Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations. Natural Computing, Springer-Verlag Berlin 2004.

[11] Martínek, T.—Sekanina, L.: An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA. In: Evolvable Systems: From Biology to Hardware. Volume 3637 of LNCS, Springer Verlag 2005, pp. 76–85.

[12] Drutarovský, M.—Šimka, M.—Fischer, V.—Celle, F.: A Simple PLL-Based True Random Number Generator for Embedded Digital Systems. Computing and Informatics, Vol. 23, 2004, No. 5-6, pp. 501–516.

[13] Wang, J.—Piao, C.—Lee, C.: Implementing Multi-VRC Cores to Evolve Combinational Logic Circuits in Parallel. In: Evolvable Systems: From Biology to Hardware, Volume 4684 of LNCS, 2007, pp. 23–34.

[14] Harding, S.—Banzhaf, W.: Fast Genetic Programming on GPUs. In: Proceedings of the 10th European Conference on Genetic Programming, Volume 4445 of Lecture Notes in Computer Science, Valencia (Spain), Springer 2007, pp. 90–101.

[15] Chitty, D. M.: A Data Parallel Approach to Genetic Programming Using Programmable Graphics Hardware. In: GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation, Volume 2, pp. 1566–1573, ACM Press London 2007.

[16] Robilliard, D.—Marion-Poty, V.—Fonlupt, C.: Population Parallel GP on the G80 GPU. In: Proc. of European Conference on Genetic Programming, Volume 4971 of LNCS, Springer-Verlag 2008, pp. 98–109.

Zdeněk Vašíček received M. Sc. degree in electrical engineering and computer science from the Faculty of Information Technology, Brno University of Technology, Czech Republic in 2006. Currently, he is a Ph. D. student at the Faculty of Information Technology. His research interests are focused on evolvable hardware. He was awarded the J. Hlavka Award in 2006. He is (co)author of more than 20 conference/journal papers focused on evolvable hardware and hardware design.

Lukáš Sekanina received all his degrees from Brno University of Technology, Czech Republic (M. Sc. in 1999 and Ph. D. in 2002). He was awarded the Fulbright scholarship to work with NASA Jet Propulsion Laboratory in Pasadena in 2004. He was a visiting lecturer with Pennsylvania State University and visiting researcher with University of Oslo in 2001. He has served as a program committee member of 10 international conferences and as editorial board member of International Journal of Innovative Computing and Applications. He co-authored more than 80 papers mainly on evolvable hardware, with over 400 citations. Currently, he is Associate Professor with the Faculty of Information Technology, Brno University of Technology. His research interests include evolutionary design and evolvable hardware.