Hardware Accelerators for Cartesian Genetic Programming

Zdenek Vasicek and Lukas Sekanina

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
vasicek@fit.vutbr.cz, sekanina@fit.vutbr.cz

Abstract. A new class of FPGA-based accelerators is presented for Cartesian Genetic Programming (CGP). The accelerators contain a genetic engine which is reused in all applications. Candidate programs (circuits) are evaluated using application-specific virtual reconfigurable circuit (VRC) and fitness unit. Two types of VRCs are proposed. The first one is devoted for symbolic regression problems over the fixed point representation. The second one is designed for evolution of logic circuits. In both cases a significant speedup of evolution (30–40 times) was obtained in comparison with a highly optimized software implementation of CGP. This speedup can be increased by creating multiple fitness units.

1 Introduction

According to John Koza, genetic programming (GP) can routinely deliver high-return human-competitive machine intelligence [1]. Its competitiveness and performance has been demonstrated in many tasks and design areas. Simultaneously, the computational power which GP needs for obtaining innovative results is enormous for most applications. GP usually spends most of time by running domain-specific simulators which evaluate candidate individuals using large training sets. In order to reduce the computational time, various methods have been employed. In general, they can be divided into four classes: (1) algorithmic – the use of smart search strategies, genetic operators and fitness evaluation strategies, (2) source code optimization for a given platform, (3) parallel GP implementations on clusters of workstations and (4) hardware accelerators. However, even with a parallel GP, the evolution is very time consuming. For example, Koza's team has utilized two clusters of workstations, 1000 x Pentium II/350 MHz processor and 70 x DEC Alpha/533 MHz processor. For 36 tasks solved using GP on the clusters, the average population size is 3,350,000 individuals, 128.7 generations are produced in average and the average time to reaching a solution is 81.9 hours [1].

This paper is focused on the acceleration of GP using a suitable digital hardware. For genetic algorithms, FPGA (Field Programmable Gate Arrays) based implementations have been created for a long time [2, 3]. As the fitness evaluation of a candidate program is the most time consuming part of GP, hardware

M. O'Neill et al. (Eds.): EuroGP 2008, LNCS 4971, pp. 230–241, 2008.

acceleration should primarily be devoted to the fitness calculation. A straighforward implementation involves multiple fitness calculation units which work concurrently. Another key issue in hardware is whether the particular problem requires the floating-point (FP) operations or fixed-point (FX) operations. The fixed-point arithmetic circuits or even logic circuits can be accelerated in a much easier way than floating-point operations on a commonly accessible hardware such as FPGA. Martin implemented a complete linear genetic programming system in an FPGA. It operates with FX expressions encoded as linear programs. Depending on the number of hardware fitness evaluation units, he reported the speedup 18 (for 2 fitness units) - 419 (64 fitness units) for the even 6-parity problem and 13 (2 fitness units) - 107 (32 fitness units) for the artificial ant problem in comparison with the PowePC processor running at 200 MHz [4].

Recently, Graphics Processing Units (GPUs) that are available in common desktop computers have been used to parallelize the fitness evaluation (also for the FP domain) [5, 6, 7]. The CPU converts arrays of test cases to textures on the GPU and loads a shader program into the shader processors. According to a GP expression, a shader program is created. The program is then executed, and the resulting texture is converted back in to an array. The fitness is determined from this output array [6]. Chitty [7] reports the speedup 0,4 – 30 depending on target problem (two symbolic regressions, Iris classification and 8-input multiplexer tested) for NVidia GeForce 6400 GO graphic card in comparison with a 1.7 GHz Pentium 4 processor. Harding and Banzhaf have shown how the speedup of candidate individual evaluation depends on the expression length for various problems. With the growing expression length and growing number of test cases, GPU becomes more effective than CPU. The maximum speedup is approx. 1000 for Boolean expressions and 14 for a protein classification problem. Note that these results only show the number of times faster evaluating evolved GP expressions is on the GPU (NVidia GeForce 7300 GO) compared to CPU implementation (Intel Centrino T2400 running at 1.83 GHz). I.e., the speedup of evolution was not reported. Unfortunately, for training sets of a common size, the overhead of transferring data to the GPU and for constructing shaders leads to a worse performance than CPU.

In the recent years, human-competitive results were obtained using Cartesian Genetic Programming (CGP) [8]. CGP is a sort of genetic programming which represents candidate programs as graphs consisting of an array of programmable nodes. This representation is natural for hardware implementation. In this paper, we propose an approach to building CGP accelerators in an FPGA. The accelerator consists of genetic unit, fitness unit and the so-called virtual reconfigurable circuit (VRC) which is utilized to evaluate candidate programs. We will show that even if only a single fitness unit operating at 100 MHz is utilized, the evolution is 30–40 times faster than a highly optimized software implementation running at a GHz processor. This approach is well suited especially for integer-level symbolic regression problems and evolution of logic expressions. The implementation utilizes a commercial off-the-shelf FPGA Virtex II Pro which contains sufficient logic resources and on-chip PowerPC processors. As genetic operations

are implemented in the PowerPC processor, the designer can define a new target problem and change various parameters of CGP very quickly.

The proposed solution was originally intended for evolution of image filters. In this particular problem, human competitive results were obtained because "the result (i.e. image filters presented in [9]) is publishable in its own right as a new scientific result – independent of the fact that the result was mechanically created" (criterion D from [10]). The goal of this paper is to demonstrate that the method can be extended to be considered as a general CGP accelerator for those problems which utilize FX operators or logic operators. The VRCs for typical target domains will be presented together with an analysis of the impact of their parameters on the performance. In particular, we will investigate the effect of setting the level of interconnectivity (the L-back parameter of CGP). A new hardware approach will be presented which allows optimizing not only for function but also for the size of a candidate program (not reported so far in literature). In addition to the use of multiple fitness units and pipelining, we will also introduce a new parallel approach to the evaluation of candidate programs. The accelerators will be evaluated using benchmark problems commonly used in this area.

2 Cartesian Genetic Programming

In CGP, a candidate program is modeled as an array of u (columns) $\times$ v (rows) of programmable elements (gates). The number of inputs, n_i, and outputs, n_o, is fixed. Feedback is not allowed. Each node input can be connected either to the output of a node placed in the previous L columns or to some of program inputs. The L-back parameter, in fact, 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. Each node is programmed to perform one of functions defined in the set Γ (n_f denotes $|\Gamma|$). As Figure 1 shows, while the size of chromosome is fixed, the size of phenotype is variable (i.e. some nodes are not used). Every individual is encoded using $u \times v \times 3 + n_o$ integers.

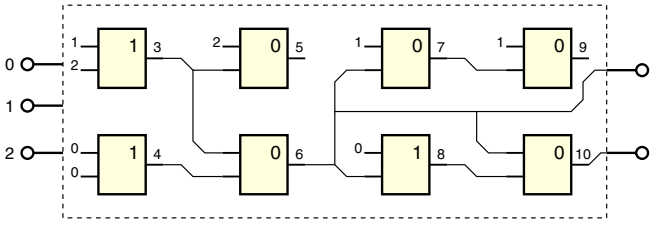

Fig. 1. An example of a candidate program. CGP parameters are as follows: $L = 3$, $u = 4$, $v = 2$, $\Gamma = \{$AND (0), OR (1)$\}$. Nodes 5 and 9 are not utilized. Chromosome: 1,2,1, 0,0,1, 2,3,0, 3,4,0 1,6,0, 0,6,1, 1,7,0, 6,8,0, 6, 10. The last two integers indicate the outputs of the program.

CGP operates with the population of λ individuals (typically, $\lambda = 5 - 20$). The initial population is randomly generated. Every new population consists of the best individual and its mutants. 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. This strategy is used to ensure the diversity of population.

The fitness function usually takes one of two forms. For the symbolic regression problems, a training set is used. The goal is to minimize the difference between the output of a candidate program and required output. For evolution of logic circuits, all possible input combinations are applied at the candidate circuit inputs, the outputs are collected and the goal to minimize the difference between obtained truth table and required truth table. In case when the evolution has found a solution which produces correct outputs for all possible input combinations, other parameters, such the number of components or delay are getting to minimize. The evolution is stopped when the best fitness value stagnates or the maximum number of generations is exhausted.

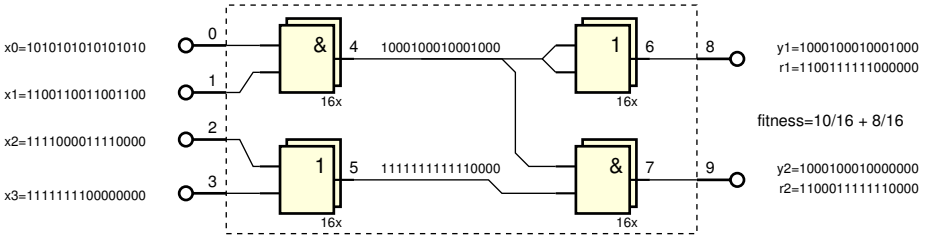

Fig. 2. Parallel simulation of a combinational circuit. Values $y1$ and $y2$ are the results of simulation, $r1$ and $r2$ are the required outputs.

Software implementations of CGP, which are intended for evolution of logic circuits, strongly benefit from the so-called *parallel simulation*. In a circuit simulator working at the gate level, a single gate is usually modeled using a logic function. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more than one evaluation of a gate in a single step. Therefore, when a combinational circuit under simulation has four inputs and it is possible to concurrently perform bitwise operations over $2^4 = 16$ bits in the simulator then this circuit can completely be simulated by applying a single 16-bit test vector at each input (see encoding in Fig. 2). In contrast, when it is impossible then sixteen four-bit test vectors must be applied sequentially. Practically, current processors allow us to operate with 64 bit operands, i.e. it is possible to evaluate the truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the obtained speedup is 64 against the sequential simulation. In case that a circuit has more than 6 inputs then the speedup is constant, i.e. 64. This technique can be also utilized in hardware. However, it is mainly useful for gate-level evolution.

In case of function-level evolution, for example, over b-bit operators (such as addition, subtraction, maximum etc.) the speedup is only c/b, where c is the number of bits of the operators implemented in hardware.

3 Accelerators for CGP

The basic idea of proposed accelerator is that a given instance of CGP (i.e. a reconfigurable graph consisting of $u \times v$ programmable nodes) is implemented as a reconfigurable circuit on 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 [11]. In order to evaluate a candidate chromosome, a controller has to store the chromosome into the configuration register of VRC and activate the fitness unit (FU). FU generates the input vectors for VRC, reads the output vectors from VRC and compares them with required output vectors. The fitness value is sent to the on-chip PowerPC processor where new candidate chromosomes are created. This architecture was introduced in [12].

3.1 Architecture Overview

The proposed CGP accelerator is completely implemented in a single FPGA and consists of Genetic unit (GU), Processor and Memory Interface (PMI), Fitness Unit (FU), VRC and a Control Unit (CU) – a communication interface to a common PC (see Fig. 3). The PC is used just to define parameters of CGP and target data (the truth table or training set). External SRAM memories are used to store large training sets (e.g. training images for designing of image filters), while on-chip BlockRAM (BRAM) memories are used to store small training sets.

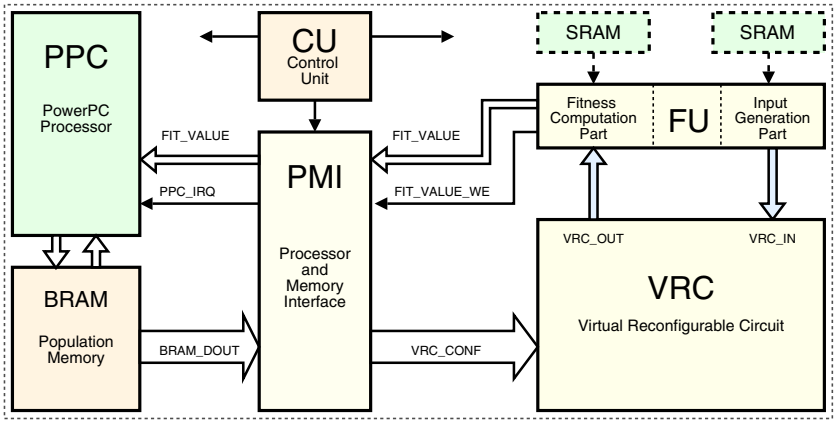

Fig. 3. Generic architecture of CGP accelerators in the FPGA Virtex 2 Pro

All components (except the VRC) are connected to the internal bus called LocalBus which provides an effective communication interface between FPGA and PCI bus. 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 requirement is specified. The instruction memory of the PowerPC is implemented using BRAMs. However, our search algorithm can completely be stored in an instruction cache.

The population of candidate configurations is also stored in on-chip BRAM memories. The population memory is divided into banks; each of them contains a single configuration bitstream of VRC. An additional bit (associated with every bank) determines 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 a circuit is evaluated, a new candidate configuration is generated. The new configuration is used immediately after completing the evaluation of the previous one.

The PMI component consists of two subcomponents working concurrently. The first subcomponent, controlled by the CU, reconfigures the VRC using configurations 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, it is sent (together with some additional data, such as the size of phenotype) to the PowerPC. An interrupt (IRQ) is generated to activate a service routine of the PowerPC. In this routine, a new candidate configuration is generated for the given bank. The PowerPC processor acknowledges the interrupt (IRQACK) and sets up the validity bit. This process is controlled by the FU. The PMI component also provides an interface to the population memory via LocalBus.

The proposed system allows the use of various search algorithms [12]. These algorithms utilize a population of candidate solutions and a single genetic operator — mutation, which inverts k bits of the chromosome (i.e. of the configuration). No crossover operator is used. An analysis of various mutation operators and pseudorandom number generators was presented in [12, 13].

3.2 VRC for Symbolic Regression Problems

Proposed CGP accelerators mainly differ in the VRC organization and fitness unit. Fig. 4 shows the VRC implemented for the image filter design problem, which is a kind of a symbolic regression problem over the FX representation [12]. Every candidate program (image filter) is considered as a digital circuit of nine 8-bit inputs and a single 8-bit output.

The VRC consists of 2-input Configurable Logic Blocks (CFBs), denoted as E_i, placed in a grid of 8 columns and 4 rows. Any input of each CFB may be connected either to a primary circuit input or to the output of a CFB, which is placed anywhere in the preceding column. Any CFB can be programmed to implement one of 16 function from Γ, where Γ includes addition, subtraction, shift, minimum, maximum and logic functions. All these functions operate with 8-bit

operands and produce 8-bit results. The reconfiguration is performed column by column. The computation is pipelined; a column of CFBs represents a stage of the pipeline. Registers (denoted D) are inserted between the columns in order to synchronize the input pixels with CFB outputs. The configuration bitstream of VRC, which is stored in a register array *conf_reg*, consists of 384 bits. 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. Evolutionary algorithm directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

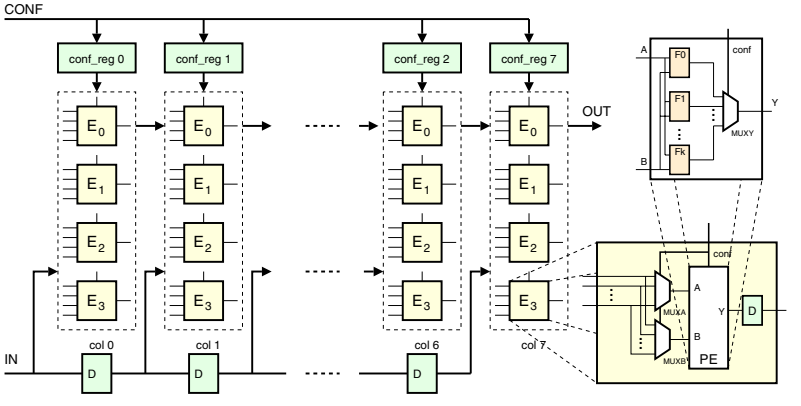

Fig. 4. VRC for symbolic regression problems

In tasks of symbolic regression, training data are stored in external SRAM memories. Fitness unit loads training data from external SRAM1 memory and forwards them to the inputs of VRC. The outputs of VRC, y_i, are compared with required outputs, r_i, (which are loaded from another external memory, SRAM2) and simultaneously stored into the third external memory, SRAM3. The FU can be considered as an extension of the VRC pipeline because in each clock cycle, a temporary fitness value is updated by a new difference, $|y_i - r_i|$. Due to pipelined reconfiguration as well as execution of VRC, the evaluation of a candidate program (circuit) requires k clock cycles, where k is the number of training vectors.

3.3 VRC for Logic Expressions

The architecture of VRC is similar to the VRC for symbolic regression. There are four main differences: PEs contain only logic functions, L-back=2 is supported, the size of phenotype can be calculated and a data parallel operation of PEs (the same as used in the software parallel simulation) is introduced. The size of data is denoted as "data width", dw, in the rest of paper. If PEs operate at dw bits then the speedup against the bit-level execution is dw-times. In order to support L-back=2, additional registers (D) have been used to store the results of stage

$i-2$ for stage i of the pipeline (see Fig. 5). The number of configuration bits for a single column is $2*log_2(n_i+2u)+log_2(n_f)$. In contrast to symbolic regression, the training data (truth table) is stored in BRAMs. For example, if $n_i = 16$ then 64 BRAMs are utilized. All possible input combinations are generated in the process of fitness calculation. When the size of circuit is not optimized, the maximum fitness value is $2^{n_i}n_o$.

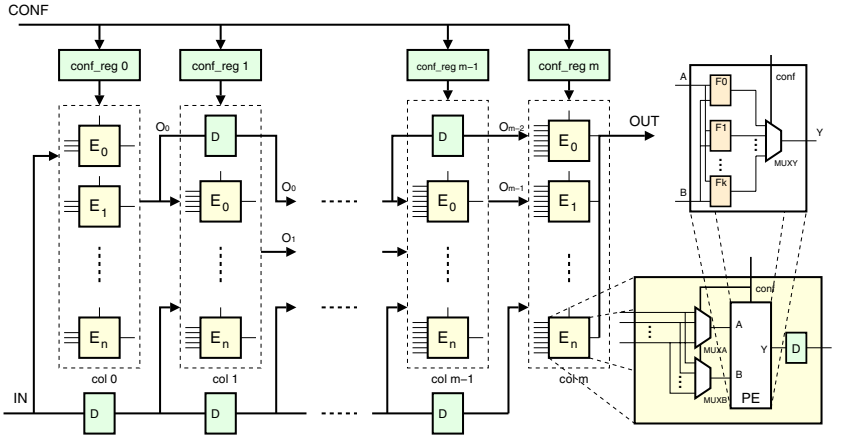

Fig. 5. VRC for evolution of digital circuits

Figure 6 explains the calculation of the size of a candidate circuit. The method assumes that a PE can implement a single wire. Once a functionally-perfect solution is found, the size is optimized. The objective is to maximize the number of PEs which operate as wires. The configuration of a single column of VRC is analyzed using comparators. The comparator returns 1 in case that a particular PE operates as a wire. These 1s are added using a tree of adders. This calculation is performed when the column of PEs is configured. It costs no extra time. The size of phenotype is stored to 8 the least significant bits of the fitness value.

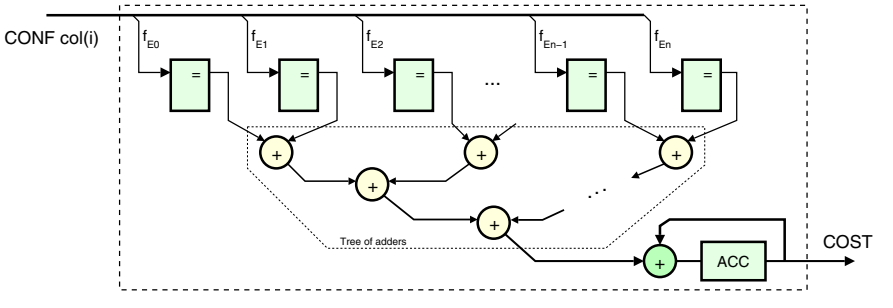

Fig. 6. Calculation of the size of a phenotype

4 Experimental Results

4.1 Evolution of Digital Circuits

Table 1 provides results of synthesis for various parameters of VRC. While the size of VRC and the number of inputs and outputs are fixed, the number of test vectors evaluated in parallel (i.e. dw) increases from 1 to 12. When no data parallel execution is used, the whole design occupies approx. 10% resources; when $dw = 12$ (i.e. 12 test vectors are evaluated in parallel by a PE) the design occupies approx. 90% resources. Using this setup we can achieve 27 times faster evaluation in comparison with a highly optimized SW implementation running at a CPU Intel Xeon 3 GHz processor (and utilizing a parallel simulation at 32 bits), even if the VRC works at 100 MHz.

Table 1. Results of synthesis for VRC with 10x10 PEs, 9 inputs, 9 outputs and 4 logic functions per PE (XC2VP50-ff1517 Xilinx FPGA). DFF is the number of flip-flops and FG is the number of function generators.

		# of vectors evaluated in parallel (dw)				
resource	**available**	**1**	**2**	**4**	**8**	**12**
BRAMs	232	14	16	20	28	36
	used	6.0%	6.9%	8.6%	12.1%	15.5%
DFFs	49788	2743	2993	3533	4709	5843
	used	5.5%	6.0%	7.1%	9.5%	11.7%
FGs	47232	4836	7813	14164	26734	41281
	used	10.2%	16.5%	30.0%	56.6%	87.4%

Table 2 contains the results of synthesis for various VRC sizes. The number of inputs, outputs, logic functions and data width are fixed. The last row shows the number of configuration bits of VRC.

Table 2. Results of synthesis for various VRCs of 9 inputs, 9 outputs, 4 logic functions and $dw = 2$ (FPGA XC2VP50-ff1517)

		VRC size			
resource	**available**	10×10	12×12	14×14	16×16
DFFs	49788	1644	2336	3634	4664
	used	3.3%	4.7%	7.3%	9.4%
FGs	47232	6242	9012	26700	32352
	used	13.2%	19.1%	56.5%	68.5%
# of conf. bits		1200	2016	2744	3584

In order to investigate the impact of the L-back parameter, we created two VRCs with $L = 1$ and $L = 2$. Proposed implementations were evaluated in the task of multiplier evolution, a traditional hard benchmark problem for evolutionary circuit design. A parallel version of Hill Climbing algorithm with neighbourhood of two and population size of 8 individuals was used (see [13]).

Table 3 summarizes results of 10 independent experiments for each problem. We can see that the increasing value of L-back parameter has the positive effect on the average number of generations and the success rate. Obtained results are comparable to the best-known results [14] (where the authors allowed the maximum value of L-back parameter).

Table 3. Results for evolution of multipliers ($\Gamma = \{$wire, and, xor, $\bar{a}$ and $b\}$)

Parameters of evolution										
multiplier	**2 × 2**		**2 × 3**		**3 × 3**		**3 × 4**		**4 × 4**	
l-back	1	2	1	2	1	2	1	2	1	2
VRC	8x8	8x8	10x10	10x10	10x10	10x10	10x10	10x10	16x16	16x16
inputs	4	4	5	5	6	6	7	7	8	8
gener. (max)	10k	10k	100k	100k	1M	1M	10M	10M	20M	20M
Results										
success rate	91%	96%	92%	100%	72%	96%	18%	84%	0%	4%
gates (min)	7	7	13	13	29	24	60	45	-	125
gates (max)	19	13	20	21	45	47	67	68	-	156
gates (avg)	9	8	15	15	34	33	61	57	-	138
gener. (avg)	1.8k	1.5k	20k	13k	22k	284k	4.84M	3.84M	-	14.2M

Table 4 compares the number of evaluated candidate circuits per one second in a highly optimized SW implementation and proposed HW accelerator. In case of the SW implementation, the time of circuit evaluation depends on the size of the phenotype and the number of training vectors. On the other hand, in hardware, this time depends only on the number of training vectors. Hence, the accelerator becomes more useful for larger VRCs and larger sets of training data.

Table 4. The number of evaluations per second. VRC operates at 100 MHz ($dw = 4$), SW is executed on the Intel(R) Xeon(TM) CPU 3.06 GHz ($dw = 32$).

#	**VRC size (SW)**			**VRC size (HW)**			**evaluation**
inputs	**10 × 10**	**12 × 12**	**16 × 16**	**10 × 10**	**12 × 12**	**16 × 16**	**speedup**
6	400	296	222	6250	6250	6250	15–28
7	250	173	89	3125	3125	3125	12–35
8	154	95	51	1563	1563	1563	10–30
9	85	50	25	781	781	781	9–31

4.2 Symbolic Regression Problems

Similarly to the accelerator for logic circuit synthesis, the CGP accelerator for symbolic regression problems was implemented on the COMBO6X card equipped with Virtex II Pro 2VP50ff1517 FPGA. Results of synthesis are summarized in Table 5. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic (including VRC and FU)

works at 100 MHz. Experimental results show that approximately 6,000 candidate programs can be evaluated per second when the training set consists of 15876 vectors which is 44 times faster than the same algorithm running at the Celeron 2.4 GHz [12]. This accelerator was utilized to discover novel implementations of image filters [12, 9, 13].

Table 5. Results of synthesis for the symbolic regression problems

VRC	IO blocks	BRAM	Slices	DFF
Available	852	232	23 616	49 788
4×8 CFBs	602	12	4 591	3 638
used	70%	5%	20%	7%

5 Discussion

The obtained speedup (30–40 against a common PC) is significant although only a single fitness unit was utilized. Note that the results reported in [6, 7, 4] employed multiple fitness units. In order to exploit also this level of parallelism, we can create up to 7 VRCs (depending on the number of PEs) on our FPGA. It means that the FPGA which we are currently using is able to speed up the evolution 100–200 times in comparison with a PC.

6 Conclusions

A new class of FPGA-based accelerators was presented for CGP. The accelerators contain a genetic engine which is reused in all applications. Candidate programs (circuits) are evaluated in an application-specific virtual reconfigurable circuit and fitness unit. Two types of VRCs are proposed. The first one is devoted for symbolic regression problems over the FX representation. The second one is designed for evolution of logic circuits. In both cases a significant speedup of evolution was obtained in comparison with a highly optimized software implementation of CGP. This speedup can be increased by creating multiple fitness units. Moreover, as the system is implemented on a single chip, it will be useful for online in-situ adaptive computing.

Acknowledgements

This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* 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, Dordrecht (2003)

[2] Shackleford, B.: A high-performance, pipelined, FPGA-based genetic algorithm machine. Genetic Programming and Evolvable Machines 2(1), 33–60 (2001)

[3] 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, pp. 143–150. IEEE Computer Society, Los Alamitos (1999)

[4] Martin, P.: Genetic Programming in Hardware. PhD thesis, University of Essex (2003)

[5] Fok, K.L., Wong, T.T., Wong, M.L.: Evolutionary computing on consumer graphics hardware. IEEE Intelligent Systems 22(2), 69–78 (2007)

[6] Harding, S., Banzhaf, W.: Fast genetic programming on GPUs. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 90–101. Springer, Heidelberg (2007)

[7] Chitty, D.M.: A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, vol. 2, pp. 1566–1573. ACM Press, New York (2007)

[8] Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 121–132. Springer, Heidelberg (2000)

[9] Vasicek, Z., Sekanina, L.: An area-efficient alternative to adaptive median filtering in fpgas. In: Proc. of 2007 Conf. on Field Programmable Logic and Applications, pp. 216–221. IEEE Computer Society, Los Alamitos (2007)

[10] Koza, J.R., Bennett III F.H., Andre, D., Keane, M.A.: Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann, San Francisco (1999)

[11] Sekanina, L.: Evolvable components: From Theory to Hardware Implementations. In: Natural Computing, Springer, Berlin (2004)

[12] Vasicek, Z., Sekanina, L.: An evolvable hardware system in xilinx virtex ii pro fpga. International Journal of Innovative Computing and Applications 1(1), 63–73 (2007)

[13] Vasicek, 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, pp. 577–584. IEEE Computer Society, Los Alamitos (2007)

[14] Vassilev, V., Job, D., Miller, J.F.: Towards the automatic design of more efficient digital circuits. In: Proc. of the 2nd NASA/DoD Workshop of Evolvable Hardware, pp. 151–160. IEEE Computer Society, Los Alamitos, CA, US (2000)