Efficient Hardware Accelerator for Symbolic Regression Problems

Zdenek Vasicek and Lukas Sekanina

Brno University of Technology, Božetěchova 2
61266 Brno, Czech Republic
{vasicek,sekanina}@fit.vutbr.cz

Abstract. In this paper, a new hardware architecture for the acceleration of symbolic regression problems using Cartesian Genetic Programming (CGP) is presented. In order to minimize the number of expensive memory accesses, a new algorithm is proposed. The search algorithm is implemented using PowerPC processor which is available in Xilinx FPGAs of Virtex family. A significant speedup of evolution is obtained in comparison with a highly optimized software implementation of CGP.

1 Introduction

Although the evolutionary algorithms were invented many years ago, they have become popular in the recent two decades when the performance of personal computers increased sufficiently. The most used evolutionary algorithm in the area of evolutionary design, Genetic Programming (GP), and its variants have been successfully applied to solve many difficult problems. However, 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 proposed. In general, they can be divided into four classes: (1) algorithm optimization for a given problem, (2) source code optimization for a given platform, (3) parallelization on clusters of workstations or GPUs and (4) hardware accelerators. Domain-specific hardware accelerators represent the most promising method due to the high performance, low implementation cost and low power consumption. The hardware accelerators can be divided into two groups: (1) application-specific (ASIC) chips developed for a given problem [1] and (2) accelerators based on reconfigurable Field Programmable Gate Arrays (FPGA) [2–6]. In fact, the ASIC-based hardware accelerators are used only in rare cases due to the high implementation cost and low flexibility. In contrast with ASICs, modern FPGAs provide a cheap, flexible and powerful platform able to compete even with common workstations. For example, it has been demonstrated 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 [7]. This performance has been achieved by introducing a kind of systolic array which significantly accelerates the evaluation of a candidate solution.

In this paper, an extension of the FPGA-based accelerator published in [7] is proposed. The goal of this work is to provide a high-performance as well as low-power evolutionary platform that accelerates the solving of the symbolic regression problems. In order to provide a high-throughput solution, the proposed accelerator utilizes multiple instances of a pipelined version of virtual reconfigurable circuit (VRC) for parallel evaluation of candidate solutions. Note that a single VRC is used in [7]. Since modern Xilinx FPGAs contain integrated PowerPC cores, the search engine can be implemented by using them. The main benefit of this architecture is that it allows the user to easily tune the search algorithm for a given problem while keeping the process of evolution on a single chip.

This paper is organized as follows. Section 2 describes Cartesian Genetic Programming; mainly the encoding is discussed here. In Section 3 the architecture of the proposed accelerator is introduced. Section 4 includes the experimental evaluation. Finally, conclusions are given in the last section.

2 Cartesian Genetic Programming

CGP is a kind of genetic programming which represents candidate programs as graphs consisting of an array of programmable nodes. The array-based representation is suitable for hardware implementation. 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. 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. 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 the CGP (compared to the GP) and allows to make an effective hardware accelerator.

CGP operates with the population of λ individuals (typically, $\lambda = 5$). 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. (i) 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. (ii) For evolution of logic circuits, all possible input combinations are applied at the candidate circuit inputs, the outputs are collected and the goal is to minimize the difference between obtained truth table and required truth table. The evolution is stopped when the best fitness value stagnates or the maximum number of generations is exhausted.

Software implementations of CGP, which are intended for evolution of logic circuits, strongly benefit from the so-called parallel simulation. The idea of parallel simulation is to utilize bitwise operators operating on multiple bits to perform more than one evaluation of a gate in a single step. Practically, current processors allow us to operate with 64 bit operands, thus it is possible to evaluate the truth table of a six-input circuit by applying a single 64-bit test vector at each input. However, this approach can not be applied for the symbolic regression problems since the function-level evolution uses usually complex operations (such as addition, minimum, maximum, difference etc.). Thus the parallelization of a candidate circuit evaluation using a cluster of workstations represents the only way how to significantly accelerate this task in software.

3 Proposed CGP Accelerator

The basic idea of the CGP accelerator is that a given instance (i.e. a reconfigurable array 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 [8].

3.1 Proposed 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 Fig. 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 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.

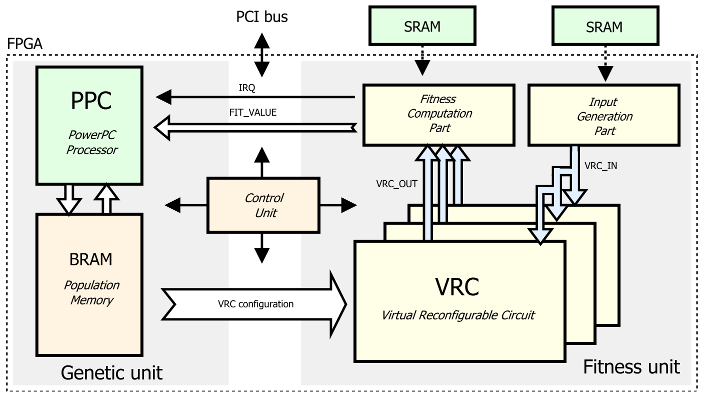

Fig. 1. Architecture of the proposed CGP accelerator

While the candidate solutions are evaluated, N_c new candidate configurations are generated. The population memory provides two independent ports (a) 32-bit read/write port A connected to the PowerPC processor and (b) m-bit read-only port B connected to the fitness unit used for the reconfiguration of VRCs. Since corresponding columns of each VRC 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 B port must be chosen with respect to (a) the implementation limits (m must be an integer divisible by 128), (b) the number of bits of a part of bitstream used to configure one column of VRC and (c) 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.

3.2 Fitness Unit

The fitness unit consists of N_c instances of VRC and two subcomponents: (a) the input generation part and (b) the fitness computation part. The training data are stored in external SRAM memories. The fitness unit loads training data from 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. While the architecture places the lowest demand on the external memory bandwidth, the highest demand is placed on the internal memory bandwidth. Since each row buffer can be implemented using embedded BRAM memories, this approach does not cause problems.

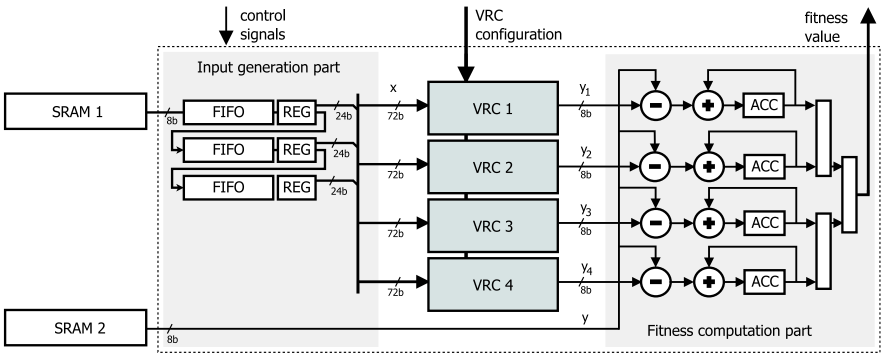

Fig. 2. Fitness Unit

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 considered. For each VRC i, the absolute difference between output value y_i and required output value y loaded from external memory SRAM2 is calculated. Then, a temporary fitness value stored in accumulator (ACC) 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 transmitted to the PowerPC and VRCs are reconfigured using new bitstreams.

The VRC consists of Configurable Logic Blocks (CFBs) placed in a grid of 8 columns and 4 rows (see Fig. 3). Any CFB can be programmed to implement one of 16 function from Γ, where Γ includes addition, subtraction, shift, minimum, maximum etc. 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 (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; 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. Evolutionary algorithm directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

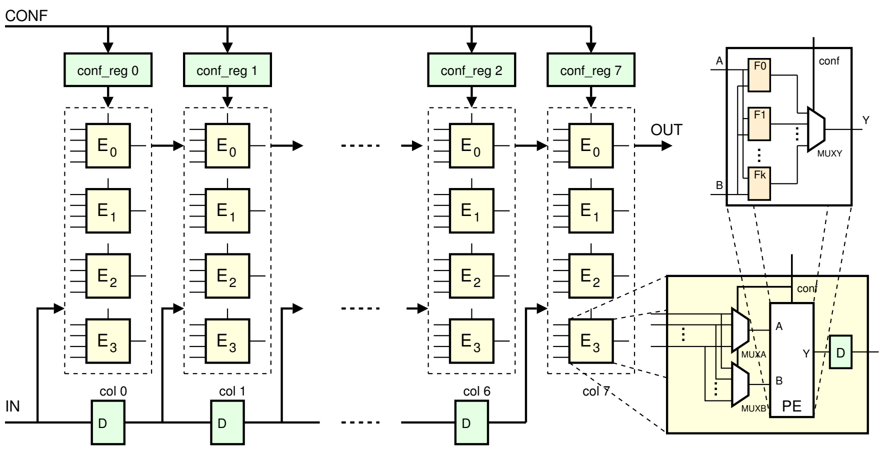

Fig. 3. VRC for symbolic regression problems

3.3 Genetic unit

In order to exploit the performance of the proposed highly-parallel architecture, GU has to generate N_c new candidate configurations while another N_c candidate configurations are evaluated. As the throughput of the population memory connected to the PowerPC can introduce a bottleneck, the number of memory accesses has to be minimized.

If the following assumptions are satisfied, the number of memory accesses can be minimized by storing the differences between the configuration bitstream of a parent and its mutated versions: (a) The search algorithm utilizes a population of candidate solutions, (b) a single genetic operator (mutation) which inverts k bits of the configuration is used and (c) no crossover operator is applied

As mentioned before, each bank of the population memory is divided into N_c sections that are evaluated in parallel. The first section contains the configuration bitstream which belongs to the first mutant while the other $N_c - 1$ sections contain only the differences between the configuration bitstream of the first mutant and the configuration bitstream of the i-th mutant. The PowerPC keeps only the information about mutations (i.e. indices of inverted bits) and best fitness value. FU contains a circuit generating complete configuration bitstream for each VRC according to the partial information stored in the sections. As soon as the evaluation is finished, the best fitness value f_{best} together with the index of the corresponding VRC i is sent to the PowerPC. The three situations can occur (i) 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, (ii) if $i > 1$ then differences between the first mutant and i-th mutant stored in i-th section have to be reflected to the first bitstream, (iii) 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 remaining banks.

4 Evaluation

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 images. 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. Since the generation of new candidate configurations is overlapped with the evaluation of candidate solutions, the total time t_{total} can be expressed as

$$t_{total} = t_{init} + N_g \frac{p}{N_c} t_{eval},$$

where N_g is the number of generations, p is the population size and N_c is the number of VRC instances (This is valid for $N_c \leq p$).

In order to implement the proposed system, a COMBO6X card equipped with Virtex II Pro 2VP50ff1517 FPGA has been used. The evolvable system was described in VHDL, simulated using ModelSim and synthesized using Mentor Graphics Precision RTL 2006a and Xilinx ISE 10.1 tools. Results of synthesis for the accelerator containing up to four VRC instances (4×8 CFBs each) are summarized in Table 1. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic works at 100 MHz.

Table 1. Results of synthesis for the various number of VRC instances

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	4 614	20%	7 925	34%	14 549	61%
DFF	49 788	3 638	7%	4 793	10%	7 103	14%

Experimental results show that approximately 24,000 candidate filters can be evaluated per second when the training set consists of 15876 8-bit vectors (i.e. a training image containing 128×128 pixels is used) and four instances of VRC are employed. 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 (see [7]). In comparison with a similar architecture published in [6], our architecture runs approximately 13 times faster for the same experimental setup.

5 Conclusion

In this paper, a new parallel as well as pipelined hardware architecture was presented for the acceleration of symbolic regression problems. The proposed system consists of two main units: (a) genetic unit and (b) fitness unit. The fitness unit contains multiple instances of virtual reconfigurable circuit to evaluate several candidate solutions in parallel. The genetic unit consists of embedded PowerPC processor. The search engine is implemented using the embedded processor. Proposed platform provides a significant speedup of digital circuit evolution in comparison with a highly optimized software implementation.

Acknowledgments

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 and the Research Plan No. MSM 0021630528 Security-Oriented Research in Information Technology.

References

1. Sakanashi, H., Iwata, M., Higuchi, T.: EHW Applied to Image Data Compression. In Higuchi, T., Liu, Y., Yao, X., eds.: Evolvable Hardware, Springer (2006) 19–40
2. Shackleford, B.: A high-performance, pipelined, FPGA-based genetic algorithm machine. Genetic Programming and Evolvable Machines **2**(1) (2001) 33–60
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, IEEE Computer Society (1999) 143–150
4. Vasicek, Z., Sekanina, L.: An evolvable hardware system in Xilinx Virtex II Pro FPGA. International Journal of Innovative Computing and Applications **1**(1) (2007) 63–73
5. 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) 373–380
6. 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 **2**(5) (2008) 386–400
7. 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, IEEE Computer Society (2007) 577–584
8. Sekanina, L.: Evolvable components: From Theory to Hardware Implementations. Natural Computing. Springer-Verlag Berlin (2004)