A Core Generator for Multi-ALU Processors Utilized in Genetic Parallel Programming

Zbyšek Gajda
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
gajda@fit.vutbr.cz

***Abstract*— Genetic Parallel Programming (GPP) evolves parallel programs for MIMD architectures with multiple arithmetic/logic processors (MAPs). This paper describes a tool intended for rapid development of GPP applications. A new software tool is proposed which is able to generate a simulator (in C language) of the MAP and a VHDL implementation of the MAP whose structure and parameters are specified in an input xml file. The proposed tool is intended to serve as first version of the core generator for MAPs utilized in GPP. Typical MAPs are synthetized and their performance is compared against the simulation running on a common PC for a typical task – a symbolic regression.**

I. Introduction

Linear Genetic programming is a widely used evolutionary computation method in which candidate programs are represented as a list of machine code instructions [1]. In Genetic Parallel Programming (GPP) a program consists of a sequence of parallel instructions [3]. GPP is based on multi-ALU processors (MAP) that execute, in parallel, several sub-instructions in one clock cycle. In contrast to the experience of a human designer, it was shown that evolutionary approach is more successful in the design of parallel programs than sequential programs [4]. Hence the GPP approach has been applied to various domains, including symbolic regression, data classification and circuit design [3], [4], [2]. In order to speed up the evolutionary design of parallel programs, an FPGA implementation of MAP utilizing 16 ALUs was proposed [2]. However, in this implementation, ALUs could perform only logic functions.

The objective of this paper is to provide a tool for rapid development of GPP applications. A new software tool is proposed which is able to generate a simulator (in C language) of MAP and a VHDL implementation of MAP whose structure and parameters are specified in an input xml file. Typical MAPs are synthetized and their performance is compared against the SW simulator on a typical task – a symbolic regression.

II. GPP and Multi-ALU Processors

A. Genetic Parallel Programming

GPP is a variant of linear GP that evolves parallel codes for MIMD architecture with multiple arithmetic/logic units (ALUs). Figure 1 shows overall architecture of GPP system which consists of a special multi-ALU processor and an evolution engine (EE).

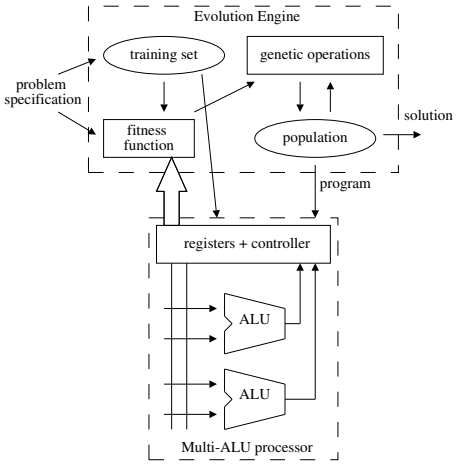

Fig. 1. Architecture of GPP system

Various test problems have indicated that parallel programs are more evolvable than sequential programs, i.e. that it is more efficient to evolve parallel programs (using multiple ALUs) than sequential programs (using one ALU). This phenomenon is called the *GPP accelerating phenomenon*.

B. Multi-ALU Processors

MAP is a kind of VLIW processors. As Figure 2 shows, MAP consists of ALUs, registers, ports, crossbar switching network and a controller. Multiple ALUs access a shared register file through a crossbar switching network. ALUs can be either identical or specific and may maintain status flags. Registers are used to store results of ALUs or constants. ALUs receive register values through a programmable crossbar network and ports. Each port can be connected to the output of any one register. A port's value can be shared by more than one ALU input. In order to prevent multiple ALUs from writing to the same register concurrently, each ALU can write only to some registers assigned beforehand.

Every MAP instruction constists of a branch-ctrl sub-instruction, ALU-ctrl sub-instructions and reg-sel sub-instructions. The branch-ctrl sub-instruction controls a program flow. The ALU-ctrl sub-instruction selects an arithmetic/logic operation, input ports and an output register. The reg-sel sub-instruction selects a register which is connected to a port. As all possible bit combinations in every instruction

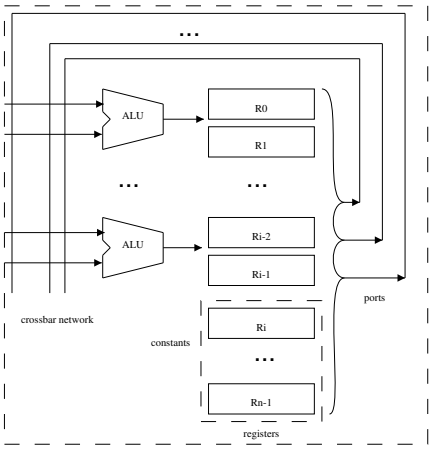

Fig. 2. Structure of MAP

are legal, programs can be generated by EE without any repairs. The genotype of an individual contains control codes of parallel instructions.

III. The Proposed Core Generator

The core generator modifies generically written C code and VHDL description of MAP and automatically generates the required structures.

A. Specification of MAPs

The XML language was used for specification of MAPs generated by the core generator. An XML file describes MAP parameters such as the number of ALUs, registers, output registers per ALU, ports and size of a datapath, program memory. The file also defines constant registers, branch operations and ALU operations. The number of bits for each field of the instruction is given in a report which is generated by the core generator according to the XML specification.

B. Simulator in C

It is important to have a simulator of MAP running on PC because many experiments have to be performed in order to "tune" effectiveness of the evolutionary design process. Furthermore, programs evolved either in SW or in FPGA have to be analyzed. The simulator can do this job in an effective and easy way (a dis-assembler was also created). The proposed simulator only simulates the execution of concurrent sub-instructions; it does not simulate a real HW. The performance of the simulator will be discussed in Section IV-D.

C. VHDL implementation

In order customize MAP according to user specification, MAP is composed of parameterizable modules controlled by FSM. Figure 3 shows the generic architecture of MAP.

The architecture includes: an ALU unit, a decoders unit, a registers unit, multiplexers units and a control unit. The ALU unit (alus_block) consists of the predefined number of ALUs. The decoders unit (alus_regs_decoders_block) is utilized to select registers designated to store results of ALUs. The registers unit (regs_block) consists of the store registers and the constant registers. The multplexers unit #1 (regs_port_muxes_block) consists of multiplexers which connect the register outputs to ports. The multiplexers unit #2 (ports_alus_muxes_block) implements the crossbar-network.

The control unit (ctrl_block) controls a data flow of MAP which includes: a program memory, an instruction register (IR), a branch unit and a Finitive State Machine (FSM). The program memory stores a parallel program. The IR stores a single parallel instrucion. The branch unit controls a program flow. The FSM controls an instruction fetching, decoding, executing and writing-back.

IV. Results

A. Specification

In order to evaluate the SW simulator and the implementation in FPGA, three different MAPs (M_A, M_B, M_C) were specified (see Table I). The core generator was used to generate corresponding simulators and VHDL implementations. It was specified for all the MAPs to utilize branch operations *next, end, jump-if-equal-zero* and *jump-if-greater-than-zero* and ALU operations *r=a+b*, *r=a-b*, *r=a* and *no-operation*.

TABLE I
Parameters of MAPs

MAP	M_A	M_B	M_C
$ALUs$	4	4	8
$data\ path$	4	16	16
$ports$	4	4	8
$output\ registers\ per\ ALU$	4	4	2
$registers$	32	32	32
$program\ size$	32	32	32

B. Results of synthesis

All the MAPs were synthesized using Xilinx ISE 6 to Xilinx Virtex 2 FPGA (device=xc2v250). Results of the synthesis are summarized in Table II.

TABLE II
Results of synthesis

MAP	M_A	M_B	M_C
$Slices$	265	853	1534
$Slice\ Registers$	163	403	536
$4\ input\ LUTs$	258	976	2429
$f_{max}[MHz]$	211.33	197.37	186.1

C. Symbolic Regression Using GPP

In order to verify that the simulator of MAP works correctly, MAP and EE were utilized to solve a simple problem. The objective was to find the formula $x^6-2x^4+x^2$ using symbolic regression in case that the training set consists of 20 samples (see [4]). The MAP was specified as M_B (see Tab.I) withtwo branch-ctrl instructions: *next* and *end*. The constant registers were set to the values 1 and -1. Each ALU was programmed

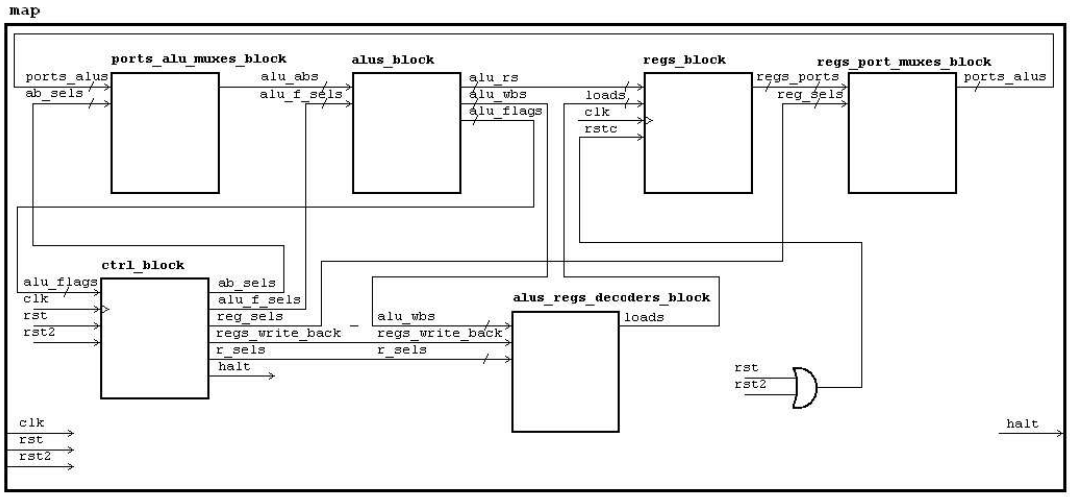

Fig. 3. Multi-ALU processor architecture

to perform one of the following operations: *r=a*, *r=b*, *r=neg a*, *r=neg b*, *r=a+b*, *r=a-b*, *r=a*b* and *no-operation*. Parameters of EE: the population size = 2000, the crossover probability = 1.0, the instruction bit mutation probability = 0.02, the selection method = a tournament (size=10) and the maximum number of generations = 20000. Seven out of ten independent runs were successful. A valid solution was found in the generation 3220 on avarage.

D. Performance Analysis

It is important to note that the evolution has not been performed in the FPGA yet. Hence it is impossible to compare the results presented here and in [2].

The objective herein was to compare the performance of the simulator of MAP (running at Athlon64@3.2GHz) and the HW implementation of MAP (in the Virtex2 FPGA) for a given parallel program. In order to do that we have used randomly generated benchmark programs and measured their performance (in MIPS). Table III shows that the FPGA implementation is k times faster (in average) than the simulator. In our case the value of k roughly corresponds to the number of ALUs divided by 2. The obtained speedup is not impressive; however, one has to consider that the simulator does not simulate real MAP (i.e. all multiplexers, decoders etc.) but it only simulates the execution of MAP's instructions.

TABLE III
COMPARSION OF PERFORMANCE

MAP	M_A	M_B	M_C
$MIPS_{fpga}$	35.22	33,06	31.02
$MIPS_{sim}$	14	14	7.9
$speedup$	2.52	2.36	3.93
$ALUs$	4	4	8
$f_{max}[MHz]$	211.33	197.37	186.1

In comparison to [2], our implementation of MAP in FPGA supports arithmetic as well as logic functions. The experiments have shown that the implementation of MAP is at least as powerful as the implementation presented in [2] (although our design is generated automatically from a specification).

V. CONCLUSIONS

In this paper, a new CAD tool was introduced to support development of applications of GPP that utilize MAPs. According to the specification, this tool is able to generate software simulator of MAP and its VHDL implementation. Results of synthesis of a typical MAP and simulations performed indicate that the FPGA implementation would be more powerful than a software simulation even if the simulator were running on a very powerful PC. In order to perform a complete GPP with the MAP implemented in hardware, an interface has to be created between EE and MAP. This interface could be a performance bottleneck of the whole system. Future research will deal with this problem.

ACKNOWLEDGMENT

The research was performed with the Grant Agency of the Czech Republic under contract No. 102/06/0599 *Methods of polymorphic digital circuit design*.

REFERENCES

[1] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming – An Introduction*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[2] W. S. Lau, G. Li, K.-H. Lee, K.-S. Leung, and S. M. Cheang. Multi-logic-unit processor: A combinational logic circuit evaluation engine for genetic parallel programming. In *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *LNCS*, pages 167–177, Lausanne, Switzerland, 2005. Springer.

[3] K. S. Leung, K. H. Lee, and S. M. Cheang. Evolving parallel machine programs for a Multi-ALU processor. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1703–1708. IEEE Press, 2002.

[4] K. S. Leung, K. H. Lee, and S. M. Cheang. Parallel programs are more evolvable than sequential programs. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 107–118, Essex, 2003. Springer-Verlag.