Evolving Machine Microprograms

P.A. Castillo
Department of Architecture and Computer Technology
University of Granada (Spain)
pedro@atc.ugr.es

G. Fernández
Department of Architecture and Computer Technology
University of Granada (Spain)
gerfer@correo.ugr.es

A. Mora
Department of Architecture and Computer Technology
University of Granada (Spain)
amorag@geneura.ugr.es

J.J. Merelo
Department of Architecture and Computer Technology
University of Granada (Spain)
jmerelo@geneura.ugr.es

J.L. Bernier
Department of Architecture and Computer Technology
University of Granada (Spain)
jbernier@atc.ugr.es

A. Prieto
Department of Architecture and Computer Technology
University of Granada (Spain)
aprieto@ugr.es

ABSTRACT

The realization of a control unit can be done using a complex circuitry or microprogramming. The latter may be considered as an alternative method of implementation of machine instructions that can reduce the complexity and increase the flexibility of the control unit. The microcode efficiency and speed are of vital importance for the computer to execute machine instructions fast. This is a difficult task and it requires expert knowledge. It would be interesting and very helpful to have automated tools that, given a machine instruction description, could generate an efficient and correct microprogram. A good option is to use evolutionary computation techniques, which have proved been effective in the evolution of computer programs. In this paper, we intend to show how evolutionary computing techniques could be used to face this problem of generating efficient microprograms. We have developed a microarchitecture simulator of a real machine in order to evaluate an individual and to assign it the fitness value (to determine whether this candidate solution correctly implements the instruction machine). The proposed method is successful in generating correct solutions, not only for the machine code instruction set, but for new machine instructions not included in such set. We have shown that our approach can generate microprogramms to execute (to schedule microinstructions) the machine level instructions for a real machine. Moreover, this evolutive method could be applied to any microarchitecture just by changing the microinstruction set and pre-conditions of each machine instruction to guide evolution.

Categories and Subject Descriptors

B.1.4 [**Microprogram Design Aids**]: Firmware engineering; D.2.2 [**Design Tools and Techniques**]: Evolutionary prototyping

General Terms

Design

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

Keywords

computer architecture, microprogramming, microarchitecture, evolutionary computation techniques, optimization, automatic design

1. INTRODUCTION

Building the control unit using microcode, the hardware design becomes simplified. However, microprogramming is a difficult task because writing good microcode requires an intimate knowledge of the hardware. Compilation from higher level languages is generally not an option, since the generated code is usually not enough quality. It would be interesting and very helpful to have an automated tool that, given a machine instruction description, could generate an efficient and correct microprogram.

In this paper we propose an evolutive method that could be applied to any microarchitecture just by changing the requirements (pre-conditions) of each machine instruction to guide the evolutionary method. We have analyzed in detail the microarchitecture of a real basic computer (Elemental Didactic Computer, CODE2). As an evolutionary problem, we have carried out the evolution of the microcode for every machine-level instruction, taking each one at a time. To test the proposed method, new machine code instructions have been defined.

2. ARCHITECTURE OF CODE2

Processor microarchitecture the is level below the macroarchitecture. The macroarchitecture corresponds to the assembly-language level. Code and data are transferred between main memory and the CPU.

At the microarchitecture-level, the control unit manages the fetch-decode-execute instruction cycle. It includes a memory of control where the microcode for all machine instructions is stored. Each control word stored in this memory includes some fields that along with the flag register control the sequencing logic to determine the next microinstruction to be executed. Bits in the control word are sent to the control bus as control signals.

CODE2 implements a register-register architecture. At the assembly-language level this machine has 16 general purpose registers (r0 to rF) and 16 machine instructions. Some registers have a special role: rE register is usually used as

stack pointer, and rD register as address register. The data path also includes the instruction register (IR) and the program counter (PC). The ALU is capable of the following operations: adition, substraction, logic NAND, logic right and left shift, and aritmetic right shift. The ALU also includes zero, sign, carry and overflow flags. Main memory size is 65536 16 bits words. Finally, CODE2 can handle 256 input ports and 256 output ports.

3. PROPOSED METHOD

The proposed method we have implemented is based on an evolutionary algorithm. In this approach, an individual codifies the sequence of microinstructions that implements every machine instruction. We have used a numeric representation to encode the list of microinstructions. An individual encodes a list of numbers, each of one represents a microinstruction. As different machine instructions have variable lengths (number of microinstructions), individual sizes are randomly initialized.

As variation operators, our method uses mutation, crossover, gene insertion and gene deletion. Mutation randomly changes a gene (microinstruction) taking into account the mutation rate. Crossover carries out the multipoint crossover between two individuals, exchanging their microinstructions. Insertion operator inserts a new microinstruction into the individual, while the last operator (insertion), randomly removes a microinstruction.

The fitness evaluation is based on simulating microinstructions. For every machine instruction, we have defined the set of resources that should be used or avoided (operations allowed and forbidden to guide the evolutionary search). The fitness function takes into account four values: 1) whether the implementation is correct, 2) how many forbidden operations are included, 3) how many allowed operations are included, and 4) individual length (the shorter, the better, if it is correct). In the case of non-correct solutions, the one that includes less forbidden operations is taken as the best.

As an example, suppose the following individual:

RA=ra	*id of the 1st register used*
RT=RF[RA]	*using a temporary register*
WA=rx	*id of the register used as destination*
RA=rs	*id of the 2nd register used*
RF[WA]=RF[RA]+RT	*the ALU carries out the operation*

It is coded as (24,16,33,9,14). In this case, the fitness value obtained after simulation was (1, 0, 3, 5). That means that this individual, the implementation using microinstructions, is correct (1) and the number of microinstructions is 5.

The source code of the proposed method is available for download at `http://atc.ugr.es/pedro/ev-micropr/`

4. EXPERIMENTS AND RESULTS

The target of the experiment was to optimize the CODE2 machine instruction set. Each machine instruction is taken separately, the pre-conditions are set and the evolutionary method is run to design that machine instruction using a sequence of microinstructions (search for a correct implementation). The second experiment is based on defining and optimizing new instructions for CODE2 (the architecture was not thought to include these new machine instructions).

Obtained results show that in nearly all runs (8 out of 10), the evolutionary method is able to find a correct solution that is, a set of correct microinstructions:

Instruction	Obtained implementation
ADDS	RA=ra RT=RF[RA] WA=rx RA=rs RF[RA]=RF[RA]+RT
LLI	WA=rx RF[WA]=0x00FF##IR
IN	AR=0x00FF##IR WA=rx DR=IP[AR] RF[WA]=DR
SHRA	WA=rx RA=rx RF[WA]=shra(RF[RA])

The second experiment is based on defining new machine instructions (not included in the original instruction set of CODE2). In the case of the NOT instruction, correct (and optimal) solutions were obtained. In the case of the logical AND instruction, although the solution obtained is not correct, just by including (by hand) an extra microinstruction we could obtain the optimal implementation.

5. CONCLUSIONS

In this work, an evolutionary approach has been applied to the problem of designing the microarchitecture of a basic real computer. We have evolved microprograms that implement not only CODE2's machine code instruction set, but new machine instructions.

Obtained results show that EAs provide for machine microcode optimizations; moreover it is usually possible to generate correct and efficient solutions.

Our approach needs some guidance introduced related to which resources should be used or avoided in the fittest solutions. This evolutionary guidance is derived automatically from the specifications of macroinstruction behavior, and does not require any "intelligent" input from human operators. Thus, the method could easily be applied to other microarchitectures by changing the pre-conditions and microinstruction set.

As a future line of work, it would be interesting adding new machine instructions and designing their microprograms using the evolutionary method. We also intend to apply our method to more complex real architectures.

6. ACKNOWLEDGMENTS

This work has been supported by the Spanish MICYT project TIN2007-68083-C02-01, the Junta de Andalucia CICE project P06-TIC-02025 and the Granada University PIUGR 9/11/06 project.