Fast Cycle-Accurate Interpreted Simulation

Zdeněk Přikryl, Karel Masařík, Tomáš Hruška and Adam Husář

Brno University of Technology,
Faculty of Information Technology,
Czech Republic
Outline

- Introduction
- State of the Art
- Concept of the Simulator
- Experimental results
- Conclusion
Introduction

- Hardware/software co-design
  - Partitioning of the tasks into HW/SW according to criteria (e.g. performance)
- Application Specific Instruction-set Processor (ASIP)
  - Specific functions of HW are available via application specific instructions
- ASIP Design Methodology
  - Architecture Description Languages (ADLs)
Architecture Description

Languages

- Describe ASIP in several ways
  - Instruction set, microarchitecture, etc.
- Automatic generation of a tool-chain
  - Assembler, simulator, profiler, etc.
- Allow fluent development of the application for the designed system
ADL ISAC

- ISAC stands for Instruction Set Architecture C
- Developed within the Lissom project
  - Inspired by LISA language
  - New features
- Belongs into the so-called mixed ADLs
- Two parts
  - Description of resources (e.g. registers)
  - Description of four basic models of ASIP
ADL ISAC Example

REGISTER bit[32] gprs [64];
BUS bit[32] sysbus { /* parameters */ }

OPERATION decode {
    CODINGROOT { {gr1(sysbus[pc])} };
}

GROUP gr1 = iadd, iload, ...;

OPERATION iadd {
    INSTANCE reg ALIAS {rs, rd};
    ASSEMBLER { “add” rd “,” rs };  
    CODING { 0b00001111 rd rs};
    BEHAVIOR { gpr[rd] = gpr[rd] + gpr[rs]; };
    ACTIVATION { %1 actop1; };
    EXPRESSION { OPC-ADD; };
}
State of the Art

- Cycle x instruction accurate simulator
- Compiled x interpreted simulator
- Static x dynamic scheduling of events in a microarchitecture

- Our simulator is cycle-accurate, interpreted and based on static scheduling of events
Concept of the Simulator

- Uses extended finite automaton as an instruction analyzer
  - First, an extended lazy finite automaton is built from ISAC model
  - Second, an instruction analyzer is created from extended lazy finite automaton
- Uses extended finite automaton for scheduling of events in a microarchitecture
  - First, an activation tree is built form ISAC model
  - Second, an automaton is created from the activation tree
Extended Lazy Finite Automaton (1/2)

- Instruction analyzer is a 7-tuple $M = (Q, \Sigma, P, s, F, S, z)$
  - $Q, s, F$ analogous to finite automaton
  - $P$ is a set of transitions in form: $pa \rightarrow q$, where $p, q \in Q$ and $a \in \Sigma^*$
  - $\Sigma$ is an input alphabet $\Sigma = \{0, 1\}$
  - $S$ is a set of semantic actions
  - $z$ is a relation $z \subseteq P \times S$
- Has to go through the process of determinization
Extended Lazy Finite Automaton (2/2)

MTV 6.-8.12.2009
Instruction Analyzer (1/2)

- Instruction analyzer is a 7-tuple $M = (Q, \Sigma, P, s, F, S, z)$
  - $Q, s, P, F$ analogical to finite automaton
  - $\Sigma$ is an input alphabet $\Sigma = \{0, 1, \varepsilon\}$
  - $S$ is a set of semantic actions
  - $z$ is a relation $z \subseteq P \times S$
Instruction Analyzer (2/2)

1

(“0”, ε) -> 2

(“1”, ε) -> 12

(“0”, ε) -> 5

(“1”, ε) -> 6

(“0”, ε) -> 7

(“1”, ε) -> 16

(“0”, ε) -> 13

(“1”, ε) -> 14

(ε, rd=1; gpr[rd]--;
17

18

(ε, rd=2; gpr[rd]--;
15

(ε, rd=2; gpr[rd]++;
11

(ε, rd=1; gpr[rd]++;
10

(ε, rd=1; gpr[rd]++;
9

(ε, rd=2; gpr[rd]++;
8

(ε, rd=1; gpr[rd]--;)

(ε, rd=1; gpr[rd]--;
(ε, rd=2; gpr[rd]++;)
(ε, rd=2; gpr[rd]++;)
Activation tree (1/2)

- Activation tree is quadruple $G = (E, H, s, z)$
  - $E$ is a partially ordered set of nodes (events)
  - $H$ is a set of edges $H \subseteq E \times E$
  - $s$ is a node rating $s : E \rightarrow 2^H^*$
  - $z$ is an edge rating $z : H \rightarrow C \times D$, where $C$ is a set of valid ANSI C conditions including empty condition $\varepsilon$, and $D \subseteq \mathbb{N}_0$ is a set of delays
Activation tree (2/2)

```
main (ε, 0) -> decode
  
actop1 (gr1_actop1==true, 0) -> gr1

gr1 (gr1_actop2==true, 1) -> actop2

halt (ε, 0) -> actop2
```
Event Automaton (1/2)

- Event automaton is a 7-tuple $M = (Q, \Sigma, P, s, F, S, z)$
  - $Q, s, F$ analogous to finite automaton
  - $\Sigma$ is an input alphabet containing events
  - $P$ is a finite set of transitions in form $c : pa \to q$, where $c$ is valid ANSI C condition including empty condition $\varepsilon$, states $p, q \in Q$ and $a \in \Sigma$
  - $S$ is a set of semantic actions
  - $z$ is a relation $z \subseteq P \times S$
  - Algorithms creating deterministic event automaton
Event Automaton (2/2)

\[(main, \varepsilon, main;)\]

\[(decode, \varepsilon, decode;)\]

\[(main, \varepsilon, main;)\]

\[(decode, \varepsilon, decode;)\]

\[(main, \varepsilon, main;)\]

\[(main, \varepsilon, main;)\]

\[(main, \varepsilon, main;)\]

\[(main, \varepsilon, main;)\]

\[(main, \varepsilon, main;)\]

\[(actop1, main;halt;)\]

\[(actop2, main;halt;)\]

\[(actop1, main;halt;)\]

\[(actop2, main;halt;)\]

\[(actop1, main;halt;)\]

\[(actop2, main;halt;)\]

\[(actop1, main;halt;)\]
Experimental results

- Table shows speeds of our simulators
- Models are without pipelines
- Several algorithms from MiBench testing suite were chosen

<table>
<thead>
<tr>
<th>Processor</th>
<th>Simulator [MHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIPS</td>
<td>30.25</td>
</tr>
<tr>
<td>Chili3</td>
<td>2.77</td>
</tr>
<tr>
<td>ARMv5</td>
<td>13.17</td>
</tr>
</tbody>
</table>
Conclusion

- Algorithm for simulator creation is based on the introduced formal models
- Event automaton can be transformed either to C (simulator, profiler) or to VHDL (hardware realization)
  - Huge additional verification of hardware realization is not needed
- Created simulator is cycle-accurate, interpreted and based on static scheduling of events
Thank You for Your Attention