Trace-Based Context-Sensitive Timing Simulation Considering Execution Path Variations

Sebastian Ottlik, Jan Micha Borrmann, Sadik Asbach, Alexander Viehl, Wolfgang Rosenstiel, Oliver Bringmann
Motivation

- Rising complexity of embedded functionality
  - Realized by complex, safety-critical SW
- Need SW performance analysis

- Advantages of performance simulation:
  - Good observability, repeatability, …
- Need accuracy/performance trade-off

Recent research demonstrates potential of context-sensitive simulation
Context-Sensitive Timing Simulation

Source code

Target binary code

Host binary code

Basic block exec.

Context

Low level analysis

Target binary control-flow

Block timings in different contexts

Block execution took x cycles

© FZI Forschungszentrum Informatik
What is a Context?

- Time to execute instruction block
  - Depends on uArch state
  - uArch state depends on preceding instructions
  - Use preceding control flow to approximate uArch State

- Necessary to make analysis feasible, but also useful to prevent excessive number of contexts
Related Work

<table>
<thead>
<tr>
<th>Publication</th>
<th>Contexts</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chakravarti et al., CODES 2013</td>
<td>1-block</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Plyaskin et al., VLSI-SoC 2011</td>
<td>n-block</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Stattelmann et al., DAC 2011</td>
<td>VIVU(n, k)</td>
<td>Static</td>
</tr>
<tr>
<td>Ottlik et al., CASES 2014</td>
<td>VIVU(n, k)</td>
<td>Static</td>
</tr>
</tbody>
</table>

**Drawbacks of existing approaches:**

- Static analysis: Hard for superscalar processors
- Plyaskin et al.: No support for control flow variation between observation and simulation
- Chakravarti et al.: Limited use of contexts
- Need to know uArch details
Basic Idea of Trace-Based Approach

- Observing hardware executions
  - No need for uArch model, can consider workload
- Timings are stored in Timing Database (TDB)
  - Must consider differences in execution order between observation and simulation during TDB generation
Way-Point-Oriented HW Traces

- Output by a dedicated HW unit
- Supported by many vendors
  - ARM CoreSight, IEEE-ISTO Nexus 5001, …
Control-Flow Reconstruction

1. Process trace step-by-step
   - CFG created on-the-fly
   - May have to split blocks
   - Identify edge control flow type

2. Identify Loops

3. Extract Loops

Result: Interprocedural Control Flow Graph
   - Loops are represented as recursive routines
Context Extraction

- Walk through CFG, maintain current context
  - Need CFG first, second pass over trace required
- Write trace timings to TDB in current context
  - Single timing for multiple blocks
    - Interpolate individual timings using instruction count
  - Multiple timings for a single block/context
    - Take average value, store number of observations in TDB

Result: Context-sensitive timings of the observed execution
Timing Database Contents (so far)

- Trace does not cover every block in every context
  - Block not executed at all, no contexts in other blocks
  - Block only executed in some context during tracing

Goal: Degrade simulation accuracy gracefully
VIVU(n,k) Contexts

- VIVU contexts abstract the call sequence leading to routine execution (incl. loops)
- Call string: sequence of call edges in CFG
  - For example: $e_2 \circ e_3 \circ e_4$
- VIVU: call strings $\rightarrow$ contexts

<table>
<thead>
<tr>
<th>VIVU</th>
<th>Call String</th>
</tr>
</thead>
<tbody>
<tr>
<td>n k</td>
<td>$e_2 \circ e_3 \circ e_4$</td>
</tr>
<tr>
<td>&gt;1 &gt;2</td>
<td>$(e_2, 0) \circ (e_3, 1)$</td>
</tr>
<tr>
<td>&gt;1 1</td>
<td>$(e_3, 1)$</td>
</tr>
<tr>
<td>1 1</td>
<td>$(e_3, 1)$</td>
</tr>
</tbody>
</table>

See: F. Martin et al., „Analysis of Loops“, Compiler Construction, Springer, 1998
Context Generalization

- Simulation uses tree-based lookup
  - Selects most specific context available
  - See: Ottlik et al., CASES 2014
- Idea: Synthesize new contexts, fill missing data
- Avg. timing from similar contexts
- For example: Fill node E
  - Prefer timing from F
  - If unavailable: Average of H and I
- Implemented in two walks over extended tree
Simulation Basics

- For each simulated block
  - Advance time by cycle count in current context
  - Update current context if necessary

- Handling unexpected control flow
  - Previous work: Context reset, not good for tracing
  - Solution: Apply heuristic to maintain context info
Experimental Setup

- Xilinx Zynq XC7Z020/Zedbo
- Software on Cortex-A9
  - OOO Pipeline w/ speculation
  - Caches: Split L1, shared L2
  - DDR3 RAM
- Custom tracing logic in FPGA
  - Trace streaming via GigE

- Focus on simulation accuracy in presence of input variations between tracing and simulation
Case Study: Image Compression

- **libjpeg-turbo based image compression**
  - Utilizes SIMD instructions of Cortex-A9
- **Varied image size & contents, compression quality**
  - Quality 100 is a special case, requires extra TDB (otherwise sim. error up to ~30%, with extra TDB ~17%)
  - Same size: additional improvements (max. error ~12%)
Case Study: Traffic Sign Recognition

- Video based recognition of circular traffic signs
  - Segmentation detects circles
  - Each circle input to SVM

- Varied inputs:
  - Image contents
  - Segmentation parameters
    - Min. gradient
    - Min. votes
Case Study: Traffic Sign Recognition

- High accuracy requires traffic sign during tracing
  - Max. and Avg. error less than 8% and 2%
  - Same image contents: Max. error nearly 2%
  - Application timing slightly idealized in simulation

Hardware Measurements

Simulation Results
Summary

- Tracing as data source for timing simulation
  - Different inputs in simulation

- Simulation highly accurate
  - Without cache model

- Supports complex embedded processors
- Supports complex embedded software
- No need for knowledge of processor details
  - Only requires limited knowledge of instruction set
Questions?

ottlik@fzi.de