Overview

- Technical introduction to: Simics, Ruby, Opal
- Simics: Full-System Simulator
- Ruby: Memory Timing Simulation
- Opal: Out-of-order Micro-architecture Simulator

Outline

- I. Overview
- II. Simics introduction
- III. Ruby
  - A. Simics interfaces
  - B. Software architecture
- IV. Opal
  - A. Software architecture

Simics

- Full system multi-processor simulator
- Simulated target: SPARC V9 (E 15k-like)
- Nice Features:
  - documentation
  - checkpoints
  - disk images
  - Scripting in python

Simics Devices (sun4u)

Simics Timing Model

- One instruction = one cycle
- Modulo interrupts, traps
- Cycle time is determined by clock frequency
Outline

I. Overview
II. Simics introduction
III. Ruby
   - A. Simics interfaces
   - B. Software architecture
   - C. CMP overview
IV. Opal
   - A. Software architecture
V. BibTeX

Ruby Introduction

- Models timing for caches, memory, interconnect and directories
- Implements multiple cache coherence protocols
- Uses event-driven simulation

Ruby Timing Model

- Queues act as delay centers
  - 1 cycle between queues

How to Drive Ruby

- Three ways to run ruby:
  - Random Tester
  - Simics (only)
  - Simics + Opal

Ruby Random Tester

- Stand-alone executable
- Action-Check pairs
  - Massive false sharing
  - Action: write a set of values in a block
  - Check: validate the values are correct
  - Invutable when developing protocols
- Other testers available
  - Lock contention
  - Deterministic behavior
  - Etc...

Ruby-Simics Interfaces

- Timing-Model interface

1. Simics encounters a memory instruction
2. Simics creates memory_tras structure
3. timing_model_callback is called
4. Ruby returns stall time
5. Ruby changes stall time
6. Simics commits instruction
7. "memory_tras" collects used memory
Simics Memory Interfaces
- Timing-Model interface
  - Provides: memory reference structure
  - Address, Ld/St, Size, I/O
  - Ruby returns stall time
- Polling interface
  - Ruby is called every N steps

SLICC Introduction
- SLICC: Specification Language for Implementing Cache Coherence Protocols
- Models multiple coherence protocols

Where’s Waldo?
- Describes the FSM in cache controller
- Data Structures
  - L1 CacheEntry.h
  - L2 CacheEntry.h
  - Directory_Entry.h
  - Node.h
- Control
  - L1_Transitions.h
  - L2_Transitions.h
  - Directory_Transitions.h

Day in the life of a Request
- simics
- simics/core/transitions/ruby/ruby.c
- ruby/simics/simicsDriver.C
- makeRequest
- ruby/system/STD/Sequences.C
- makeRequest
  - doRequest()
  - node->L1Cache->tryCacheAccess()
  - tryCacheAccess()
  - issueRequest()
  - ...
Ruby Configuration
- `ruby/config/config.include`
  - All parameters defined here
- `ruby/config/rubyconfig.defaults`
  - Defines parameters for the ruby module
  - All parameters can be adjusted at runtime
- `ruby/config/tester.defaults`
  - Defines parameters for the tester

CMP Overview
- Node contains
  - Exactly one Processor and L1 H/D Cache
    - 1:16 in the system
    - Partitioned across 1:16 chips
  - 0 to N L2 Cache Banks
    - At least one per chip
  - 0 to N Directories
    - At least one per system
- Network
  - One network connects all components in the system
  - Composed of switches and point-to-point links

Outline
- I. Overview
- II. Simics introduction
- III. Ruby SW architecture
- IV. Opal SW architecture

Processor Simulator: Opal
- Models a R10000 like out-of-order processor
- SPARC V9 instruction set
- Timing-First Organization

Timing-First Simulation
- Timing Simulator
  - Does functional execution of user and privileged operations
  - Does speculative out-of-order multi-processor timing simulation
  - Does NOT implement functionality of full instruction set or any devices
- Functional Simulator
  - Does full-system multi-processor simulation
  - Does NOT model detailed micro-architectural timing

Timing-First Operation
- As instruction retires, step CPU in functional simulator
- Verify instruction's execution
- Reload state if timing deviates from functional
  - Instructions with unimplemented side-effects
  - Not loads/store to I/O devices
Benefits of Timing-First

- Supports speculative multi-processor timing models
- Leverages existing simulators
- Software development advantages
  - Increases flexibility and reduces code complexity
  - Immediate, precise check on timing simulator

Conclusions

- Simics
  - Functional simulator
  - Attach timing modules to control execution time
- Ruby
  - Uses generated and non-generated code to simulate the memory system
  - Extended to simulate CMPs
- Opal
  - Timing first out-of-order processor model
  - Drives execution

Top Level Interfaces

Backup Slides

More Opal Details

Pipeline Overview

System
Sequencer
- Instruction Window
- Register Files
- Caches / LSQ / MSHR (or ruby cache intf)
- Branch Predictors
- Simics / Checking Routines
- Micro-architectural checkpointing
- Instruction / Memory / Branch Tracing

Static Instruction
- One static instructions per physical address
- Can be cached in instruction pages
- Fields of interest:
  - opcode, type, source / dest registers

Dynamic Instructions
- One dynamic instruction per in-flight instruction
  - data: renamed registers, events
  - functional execution
  - predict & actual program counter

Instruction Window
- All in-flight instructions are tracked
- Markers delimit pipeline progress
- Implemented using rotating buffer

Abstract Register File (arf)
- Instructions treat registers uniformly

Statistics
- pseq statistics
- observer functions
  - observe instruction
  - observe static instruction
  - observe thread switch
  - observe transaction complete
**Branch Predictor Overview**

- Branch Trace
  - Start
  - Branch Trace Take
  - Branch Trace Finish

**BP Classes**

**Configuration Files**

- Files define all micro-architectural parameters
  - imported as global ALL CAPS variables
  - "name=value" pairs
  - found in opal/config
- Must load file before running opal!
  - load-module opal
  - opal0.conf filename

**Adding global variables**

- config.include
- config.defaults

**Template for stand-alone opal**

```plaintext
makefile
```

**Makefile Defines**

- PIPELINE_VER: pipeline visualization output
- McOUNT_Verbose: startup debugging
- VERIFY_SIMD; once per new version of simd
- RECODE_PARCH: disables static instruction caching
- USE_MINUTES: increases performance
- Most defines should be variables not compile time options.