Out-of-Order Microprocessor Simulation

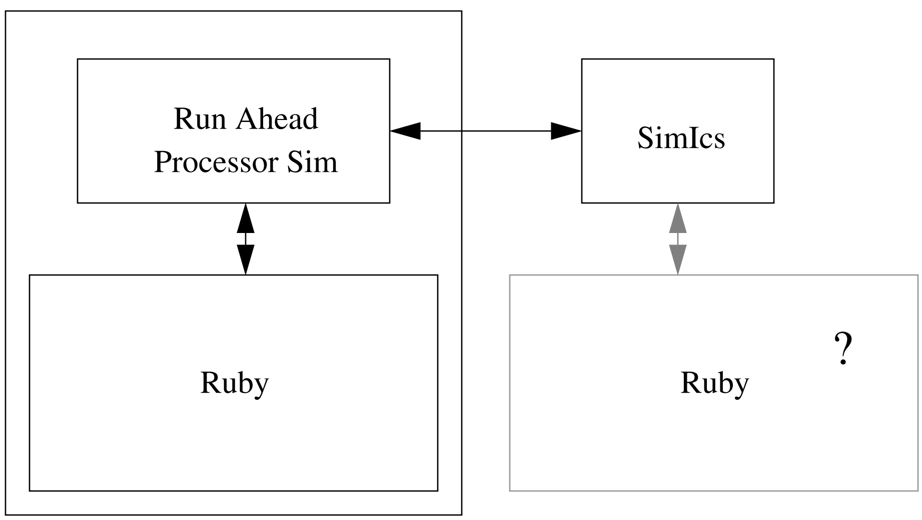
Goal: Timing accurate model of a speculative, out-of-order microprocessor in a multiprocessor system

- Enable micro-architectural design comparisons
- Enable (faster) approximation of speculative microprocessors
- RAPS: Run-Ahead Processor Simulator

Timing Overview

- SimIcs must allow SIM_continue() on a per processor basis
- Number of instructions retired per processor (SimIcs Time) can be *arbitrarily* skewed
- Need more accurate time scale: count time by "clock cycles"
- Global time maintained by: Ruby event queue

Architecture Overview



- RAPS steps SimIcs on retire.
- Should SimIcs reads memory from a timing accurate "duplicate" Ruby?
  No. Read results could differ due to contention.

- Does SimIcs as retirement engine need to stall?
  No. Times are skewed - hard to say when an instruction could retire

- Propose: Oracle interface into Ruby for "in-flight" data.

Assumptions

High Confidence	Medium Confidence	Lower Confidence	Less Confident
Ruby			
SimIcs	Timing Models		Only Native Implementation
Speculative Sparc	Statistics	SMT	MT Simulation

Design Issues

SimIcs

- Run-ahead speculative simulator results do not match on retire: Why?
  - A. Mis-speculation
  - B. Implementation Error
  - C. Exception

ALL are treated the same: SQUASH!

    Flush pipelines
    outstanding memory requests continue (phantom consumer?)

- MUST be able to step processors independently
- SHOULD be able to speculate past exceptions (TBE entries)
  - Register exception call-backs
- SHOULD be able to get easier ASID / PID access

```
extern struct unix_process *get_processor_unix_process(
  processor_t *cpu_ptr );
void set_processor_unix_process( processor_t *cpu_ptr,
  struct unix_process *proc );
```

- SHOULD be able to step processors faster (30x slowdown)
- SHOULD be able to read condition code / floating-point state registers
- SMT: could constrain verification checks - privileged instructions?

Ruby

Raps interfaces with Ruby.
Ruby continues to owns data, timing, data, cache statistics.
Ruby scales event queue to be clock cycle based.
Ruby stores non-speculative state... Raps store speculative state (?)

    Raps can't squash anything issued to Ruby

- MUST support finer grain timing
- MUST support for SimIcs "run behind" model
- SHOULD integrate with Raps: SMT architectures want feedback where misses take place
- SHOULD add "flush" interface for membar, stbar instructions
- COULD extend MessageQueue to support squashing
- COULD try to squash memory requests driven by implementation errors

Speculative Sparc Architecture “V9++”

- Register Windows

Interface?

    (CWP, Register Number) Pair

Model?

    A. Large flat register file
    B. Sum Addressed? (CWP, Register Number)
    C. Limit speculation to k extra windows above & below CWP ($k = 2$)

- Delay Slots

Software Implementation?

    global state

Hardware Implementation?

    global state

- Squash

Native Implementation

Initially assuming running RAPS on a Sparc V9 (condition codes)

Timing Models

Statistics

What research questions are you immediately interested in?

Design for SMT (Multiscalar?)

Multi-Threaded Simulation

SimIcs interfaces which need to be MT safe:

-- Most SimIcs interfaces are “read-only” so should be easy to make MT safe

- SIM_continue
- SIM_get_program_counter
- SIM_logical_to_physical
- SIM_read_phys_memory

Ruby interfaces which need to be MT safe ?

Design / Scheduling

Coding Standards
Object oriented design and testing
Early integration with ruby
Early implementation of statistics gathering
SimpleScalar-like interfaces: ease transition from earlier Wisconsin success