Computation Structures Group

MIT’s RISCY Expedition

Andy Wright, Sizhuo Zhang, Thomas Bourgeat, Amol Bhave, Jamey Hicks, Arvind

Computer Science & Artificial Intelligence Lab. Massachusetts Institute of Technology

RISC V Workshop, Redwood Shores, CA
January 6, 2015
Introduction

We made an 64-bit RISC-V processor
- IMAFD extensions
- Machine, Supervisor, and User modes
- Boots RISC-V Linux with Sv39 Paged Virtual Memory
- Tandem verified with Spike
- Written in Bluespec System Verilog

We are using this as a base for our “Riscy Expedition”
Riscy Expedition

- Formal specification of the ISA
  - using an operational model
- Formally verified processor implementations
- Memory Consistency Models
  - Should RISC V have a weak memory model?
  - Which memory model?
- Accelerators
- Microarchitecture exploration
- VLSI implementations using standard ASIC flow – “push button” synthesis
  - Processors tuned for ASIC designs
Verified Design

- Our Philosophy: Get a working processor first, figure out why it is slow, and make it faster without breaking it.
- There is no separation of Design and Verification – There’s just verified design.
- Currently we are using *Tandem Verification*.
- In our expedition we are moving to *Formal Verification*.

* Technology developed by Bluespec Inc
Tandem verification

S contains:
- The pc at step n.
- The instruction executed at step n.
- The operands and the concerned registers.
- The packets exchanged with HTIF, if exists.
- The memory operations wanted to be performed, if exists.
- ...

Very useful but for highly-parameterized designs it provides only a weak sense of correctness.

\[ S_{\text{spike}}(n) = S_{\text{riscy}}(n) \]
Formal Verification and Specification

- Formal Verification is constructing a proof that a processor matches a formal specification
  - But we need a formal specification for RISC-V
- In the ISA there are lots of (odd) cases that need to be specified for a formal verification
  - Are referenced bits in page table entries set for speculatively accessed pages?
  - A single instruction can result in up to 13 effective memory accesses. How exactly do they interact with each other?
    - How does they influence the memory model?
Memory Consistency Model

Strong consistency models (e.g. SC, TSO)
- Easy to understand, but restricts optimizations

Weak consistency models
- Permit HW optimizations, but difficult to understand
- Release Consistency (RC)
  - Defined with “operation performed with respect to certain processor”
- Power/ARM
  - Definition in official ISA is not clear
  - General description: no ordering should be assumed
  - Special cases where ordering is enforced
    - Dependent loads cannot be reordered
We want *Simple* operational definitions

- Legal behaviors must be observable on a simple abstract machine consisting of cores and a shared monolithic memory
- Cores execute one instruction at a time in program order
  - May have local buffers, like the store buffer used to define TSO; useful in modeling implicit reordering of memory instructions
  - The effect of Fence instructions is to flush local buffers
- Multi-ported monolithic memory which processes one read or write immediately
WMM: A new weak consistency model

- WMM is defined using a store buffer and an invalidation buffer per core
  - Permits all load-store reorderings except stores overtaking loads
  - Allows almost all microarchitectural optimizations
    - out-of-order execution
    - memory dependency speculation
    - load-value speculation
  - Provides an easy to understand operational model between cores
“Flavored” Memory Accesses

This looks like 4 separate cores accessing the same memory
Flavored Memory Synchronization

SFENCE.VM

- **D.ST < SFENCE.VM < VAT.LD**
  - VAT.LD must see result of D.ST
- With this specification, some SFENCE.VMs are missing from the Linux kernel

No instruction for the other direction (yet)

- **VAT.ST < ? < D.LD**
- So does D.LD always need to see the results of all previous VAT.STs immediately?

We are looking at these flavored accesses closely and incorporating them into the memory model
Further Work

- **Accelerators**
  - Convolutional Neural Network accelerator in BlueDBM

- **Microarchitecture**
  - Out-of-order execution multicore processor

- **ASIC Implementation**
  - Processor building blocks tuned for ASIC performance
Questions?

Contact information:
- Andy Wright – acwright@mit.edu
- csg.csail.mit.edu/riscy-e