Formal Methods and Systems

David Cock, ETH Zürich

18/01/16
Overview

- Correctness challenges in Barrelfish.
- System configuration using SAT.
- Tracing and online invariant checking.
- Better languages for Systems.
The State of the Fish

- 7 architectures: OMAP44xx, ARMv7/GEM5, X-Gene 1, ARMv8/GEM5, Xeon Phi, x86-64, x86-32
- 42 applications + 51 test apps
- 9 languages
- 32 committers
- 9 years old
- > 1.1M lines of code

This is no longer a small research project! We're starting to see the engineering challenges of a large system.
Getting It Right

A lesson from history: It's easier to prove code correct, if it actually is correct!

- We embarked on a new port last year: ARMv8.
- This forced us to face some codebase “challenges”.
- We now support fewer platforms, more thoroughly.
- We now make a core vs. non-core distribution.
- Proper debugging is coming (more later).
SAT Solving and the SKB
Handling OS complexity

- System Knowledge Base
  - Hardware info
  - Runtime state

- Rich semantic model
  - Represent the hardware
  - Reason about it
  - Embed policy choices
What goes in?

- Hardware resource discovery
  - E.g. PCI enumeration, ACPI, CPUID…
- Online hardware profiling
  - Inter-core all-pairs latency, cache measurements…
- Operating system state
  - Locks, process placement, etc.
- “Things we just know”
  - SoC specs, assertions from data sheets, etc.
Current SKB applications

- General name server / service registry
- Coordination service / lock manager
- Device management
  - Driver startup / hotplug
- PCIe bridge configuration
  - A surprisingly hard CSAT problem!
- Intra-machine routing
  - Efficient multicast tree construction
- Cache-aware thread placement
  - Used by e.g. databases for query planning
Prolog + SAT

- There are limits to what Prolog will efficiently solve.
- Address allocation under alignment constraints e.g. PCI, is better expressed in terms of bits.
- SAT solvers have gotten really good lately.
- Can we express PCI bridge config as SAT (yes!).
- Can we put a SAT solver in the SKB (research!).
Tracing for Invariants
HW Tracing for Correctness

Are HW operations right?

- $\exists v.a \rightarrow pa$
- $\forall v.a \rightarrow pa$
- unmap(pa);
- cleanDCache();
- flushTLB();

- Real time pipeline trace on ARM.
- Can halt and inspect caches.
- HW has “errata” (bugs).
- Check that it actually works!
- Catch transient and race bugs.

Filter at line rate

Log & process offline

Check temporal assertions

18 January 2016

Industry Retreat 2016
HW Tracing for Performance

• Should see N coherency messages.
• Do we?
  - The HW knows!

Is URPC optimal?

Core 0

URPC[0] = x;
URPC[1] = 1;

Core 1

while(!URPC[1]);
x = URPC[0];

Log & process offline

18 January 2016

Industry Retreat 2016

Log & process offline

18 January 2016

5Gb/s

Filter at line rate

Log & process offline

18 January 2016

Industry Retreat 2016
Online Example: LTL to Büchi

- LTL(-ish) formula: A store on core 1 is eventually visible on core 2.
- Think regular expressions for infinite streams.
- As for REs, we compile a checking automaton.
- Run the automaton in real time and look for violations.

\[
\text{store } 0xa000.1 \quad \Rightarrow \quad \Diamond \Box \text{read } 0xa000 = 1
\]
Could We Trace a Rack?

- Barrelfish is aiming for rack-scale single-image systems.
- We'll rely on a lot of coordination and consensus algorithms.
- It would be really useful to debug these noninvasively.
- 64 SoCs x 5Gb/s = 320Gb/s trace output.
- That'll need some data reduction, but it's very feasible.
- Online checkers (e.g. automata) will be essential at this scale and up.
Languages
Languages and Formal Methods

- Practical kernels are C/C++/ASM
- Some things we might like:
  - First-class messaging (Go)
  - Specifying layout (Rust)

The hard part about reasoning about “C”, is that we keep stepping outside the language.
What *Should* We Write Kernels In?

- Some languages have some of what we want:
  - No runtime, high performance (C)
  - Predictable resource usage (C, Rust)
  - Clear and clean semantics (Haskell, Rust?)

- No languages have everything (yet):
  - Enough expressive power: Can you enable the MMU, or thread switch without breaking the language rules?

- We should experiment with this: start with Clang/LLVM, drop the ugly parts?
Poster on HW tracing this evening.