Litmus Testing at Rack Scale

David Cock, ETH Zürich
david.cock@inf.ethz.ch

ABSTRACT

In tackling some of the challenges of rack-scale computing—large & persistent memories, non-coherent caches, and the expansion of the system interconnect to include an Infiniband or converged Ethernet-type network—research operating systems are exercising hardware in novel ways. For example, the fast user-level message passing of Barrelfish [Baumann et al., 2009] exploits a deep understanding of the x86 cache coherence protocol for extremely low-cost inter-core messaging, while SpaceJMP [Hajj et al., 2016] relies on rapidly multiplexing a virtual address space, to handle large physical memories.

One challenge posed by both of these approaches is that they rely on the behaviour of underspecified portions of the architecture: coherency protocols, MMU updates, caches. These are hairy enough on their own, but combined, we have a combinatorial explosion of potential interactions. Today, the correctness of mechanisms that cleverly exploit such hardware features, in ways unintended by their manufacturers, is seriously in doubt. This is below the level at with the seL4 project [Klein et al., 2009] axiomatised the hardware; There are no formal verification results here, and it's not clear that there will be.

The programming languages community also address the issue of underspecified modern hardware, particularly the semantics of relaxed (non-sequentially-consistent) memory models under concurrency. A leading approach there, which we propose to adopt, is the *litmus test* [Alglave et al., 2011]. Litmus testing, rather than trying to fully specify the hardware's behaviour, builds a set of candidate executions, often validated by exhaustive testing, that together cover all cases encountered in practice e.g. a store followed by a barrier is visible to a later load.

Such litmus tests exist for some operations we consider: ARM specifies litmus tests for modifying virtual memory mappings such that later loads cannot see invalid results, for example. Our goal is to extend these to cover both uses not anticipated by the vendor, in particular concerning the *interaction* of mechanisms, and to validate them in real time at a large scale. As a motivating example, consider a rack-scale Barrelfish/ARMv8 system, where UMP-style messaging is extended across a non-cache-coherent RDMA-based network. Defining the semantics of such a protocol involves (at least): the weak memory model of ARM, the coherence protocol within a SoC, the non-coherent RDMA operations, and the dynamic establishment of VM mappings for buffers. Moreover, all of these must be considered *simultaneously*: Exactly which actions (barriers, cache invalidations, . . .) are needed to ensure the needed semantics?

Litmus tests for such a protocol require the use of real hardware, in a realistic deployment. As we're worried about concurrent behaviour (races), we need to validate our tests in real time (or at least by non-intrusive logging), on an actual distributed platform. We propose to make use of *program trace*, as implemented on many modern architectures, in particular ARM. By collecting cycle-accurate trace data nonintrusively across a rack of machines, we hope to use rack-scale litmus testing to validate the hardware interface of a rack-scale OS. This talk will lay out some of the challenges involved, particularly the need for data reduction of 100Gb/s+ trace streams, and expected further applications of the hardware and software developed.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

References

Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. Litmus: Running tests against hardware. In *17th TACAS*, pages 41–44, 2011.

Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The Multikernel: a new OS architecture for scalable multicore systems. In *22nd SOSP*, pages 29–44, 2009.

Izzat El Hajj, Alexander Merritt, Gerd Zellweger, Dejan Milojicic, Reto Achermann, Paolo Faraboschi, Wen mei Hwu, Timothy Roscoe, and Karsten Schwan. SpaceJMP: Programming with multiple virtual address spaces. In *21st ASPLOS*, 2016. (to appear).

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *22nd SOSP*, pages 207–220, Oct 2009.