SIGOPS to SIGARCH: "Now it's *our* turn to push *you* around"

Jeffrey C. Mogul
HP Labs, Palo Alto, CA, 94304
Jeff.Mogul@hp.com

1 Introduction

For too many years, operating systems researchers and developers have pretty much taken whatever the computer architects have dished out. With the exception of virtualization support, and maybe transactional memory, architects and their cronies in the CPU industry have not sought or encouraged innovations that would improve the execution environment for operating systems. Even worse, many architecture researchers do not even bother to simulate and report on operating system behavior when evaluating their novel proposals.

Times have changed: architects are running out of new ideas that lead to significant application-level performance improvements; we must now rely on improved parallelism. But parallelism stresses the very issues that operating systems research has focussed on: distribution, resource management, I/O, etc. Also, many modern applications spend significant execution time in OS functions; it really does matter whether a CPU works well on OS code. So it is time for us to stop accepting what architects throw at us, and time to start telling them how to design our computers.

2 What has gone wrong

I have always been amused that scientific computing papers often refer to CPU time spent in the operating system as "noise." (e.g., [7]). This implies a rather negative view of the value that an OS provides. Perhaps for HPC users, the operating system really is just an annoyance, but for most computers, from sensor-net nodes through handhelds and laptops to servers, the OS does useful work, and often a lot of it.

There is some evidence to show that, for many real-world applications, plenty of execution time is spent in the OS [4]. (I will assume a loose definition of "the operating system" – it's more than just kernel code, since in many cases people have simply moved OS functionality into user-mode libraries.) Perhaps this is not yet "ample" evidence, although I suspect that this is mostly for lack of a systematic study.

But computer architects, at least from the evidence available in the scientific literature, seem to assume that the operating system does not exist — except perhaps when they assume it will magically manage application-thread resources that the hardware cannot manage itself. Architecture papers tend to use application-only benchmarks, and seldom account even for the interference between application and OS execution (there are, of course, counter-examples; e.g., Nellans *et al.* [4]). In short, while architecture researchers sometimes pay lip service to the OS, they almost never discuss the impact of architecture on OS behavior.

Meanwhile, the typical operating systems paper almost always uses the phrase "on commodity hardware". We assume that we are stuck with whatever flaws this hardware has.

3 Pushing back, constructively

Here are a few suggestions for constructive steps that operating systems researchers could take to help architecture researchers see the error of their ways.

3.1 Put the OS back in ASPLOS

SIGOPS co-sponsors a conference called ASPLOS. ASPLOS ostensibly stands for "Architectural Support for Programming Languages and Operating Systems." However, based on my somewhat rushed examination of the ASPLOS 2010 proceedings, it would be more accurately called "Architectural Support and OS Support for Making Computer-Intensive Benchmarks Run Faster." Only one or two papers appear to describe new or improved architectural support for an OS function (Sanchez *et al.* [5] describe hardware support for fine-grained scheduling on multicores; Johnson *et al.* [3] describe how to remove responsibility for contention management from the OS scheduler).

Perhaps there really are no further hardware innovations that would improve support for operating systems, but this seems unlikely. SIGOPS should encourage the ASPLOS leadership to actively solicit some good papers of this sort.

3.2 Fund a decent simulator

Given the complexity of modern silicon, it is almost impossible to do architectural research without simulation. (Emulation via FPGAs is only practical for very simple processors – for example, Chuck Thacker's Beehive system.) However, most widely-available, well-supported simulators do not do "full system" simulation

of low-level architectural behavior – that is, you can't run a real operating system on them. There are a few exceptions, such as M5 [2], but M5 does not currently support the x86 ISA – therefore, it is hard to validate against modern hardware or to make a truly convincing case.

Our community (both SIGOPS and SIGARCH) would be a lot better off if we had a well-supported, Open Source, cycle-level, x86 simulator. Getting this to happen appears to require at least a modest amount of funding. This money will probably have to come from a government agency, and the community needs to articulate why this is worth doing.

3.3 Create an OS-relevant benchmark suite

Architecture researchers believe in quantitative measurements, which is good; they believe in shared benchmarks, which is good; but the benchmarks they use (SpecCPU, SPLASH, PARSEC) almost never involve the operating system. This is bad.

There are some benchmarks that stress operating system functions (e.g., SPECWeb, Rubis, TPC-W). Architecture researchers almost never use these, for a few reasons. First, they are hard to get running, and often have complex parameter settings. Second, the systems we want to simulate often involve networks of computers, and this greatly complicates the problem of getting something running. Third, even if one has a full-system simulator, getting results in any reasonable amount of wall-clock time requires running the benchmark for just a few seconds (or less) of simulated time, which means that the benchmark has to be seriously perverted – most of these benchmarks are not designed to give useful results so quickly.

If we want architecture researchers to think about support for operating systems, we will probably need to help them with a suite of benchmarks that stress the things that we care about, and are pre-packaged to run easily (without a lot of thought about parameters) on cycle-level simulators.

There are a lot of possibilities for OS-relevant benchmarks; in addition to those listed above, one might include Hadoop, or a virtualization management workload [6]. (Micro-benchmarks, such as LMbench, also have their uses, but can also be quite misleading.) The trick will be getting them to run, with useful results, on a simulator: e.g., a benchmark that expresses the "essence of Hadoop" in just 1 second.

4 A few examples

Here are a few last-minute examples of the kind of architectural explorations OS people could provoke:

- As Baumann *et al*. have pointed out, your computer is a distributed system [1]. They argue that therefore the operating system should be structured as a distributed system, communicating internally by messages rather than shared memory. But, as they write, "On current commodity hardware" (there's that phrase again!) "the cache coherence protocol is ultimately our message transport." Perhaps, if their multi-kernel approach is the right one, we should be demanding a true core-to-core message-passing mechanism, exposed in the official architecture.
- One of the traditional functions of the OS is to manage resources for contending processes and threads. As hardware becomes more complex, with more resources to be contended over, the resource-management job of the OS becomes harder. Visibility matters: if you can't see it, it's hard to manage it. Of course, inferential techniques can sometimes be made to work, but explicit monitoring of important resources seems like a better way.
  Modern CPUs do have lots of performance counters; this would seem like an ideal interface between hardware and the OS. However, most of these counters are poorly documented, and inconsistent even between CPUs from the same vendor. We should demand a consistent, well-documented subset of these performance counters that the OS can rely on – and when the architects invent a new resource that could run out, they should be honest enough to make its performance counters part of the official definition.

References

[1] A. Baumann, S. Peter, A. Schpbach, A. Singhania, T. Roscoe, P. Barham, and R. Isaacs. Your computer is already a distributed system. Why isn't your OS? In *Proc. HotOS*, 2009.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, 2006.

[3] F. R. Johnson, R. Stoica, A. Ailamaki, and T. C. Mowry. Decoupling contention management from scheduling. In *Proc. ASPLOS*, pages 117–128, 2010.

[4] D. Nellans, R. Balasubramonian, and E. Brunvand. OS Execution on Multi-Cores: Is Out-Sourcing Worthwhile? *SIGOPS Oper. Syst. Rev.*, 43(2):104–105, 2009.

[5] D. Sanchez, R. M. Yoo, and C. Kozyrakis. Flexible architectural support for fine-grain scheduling. In *Proc. ASPLOS*, pages 311–322, 2010.

[6] V. Soundararajan and J. M. Anderson. The impact of management operations on the virtualized datacenter. In *Proc. ISCA*, pages 326–337, 2010.

[7] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, OS clock ticks, and fine-grained parallel applications. In *Proc. ICS intl. conf. on Supercomputing*, pages 303–312, 2005.