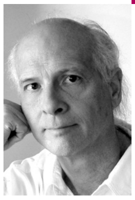
RIK FARROW

musings



Rik is the Editor of *;login:*.

rik@usenix.org

IN MY FEBRUARY 2010 COLUMN [1], I took a humorous look at the future of computing. I let my imagination go wild, as my candidate for the future of system administration, Chuck, dealt with the various issues that arose during a workday. Of course, the picture really isn't as rosy as I made it out to be. For example, the organic, in-wall system might take at least 20 years before it becomes practical. And if a system with equivalent power were built using today's technology, Chuck would have been quickly roasted—that is, if the wall itself didn't melt down first.

You have likely heard of the walls of power and memory that current system and CPU designers are facing. Clock speeds are not getting much faster, as faster clock speeds mean more power dissipated as heat. At 120 watts per centimeter squared, Chuck's office wall would need to radiate megawatts of power if each centimeter produced the equivalent processing power found in today's server-class CPUs. But, on a more practical level, just dealing with cooling one 120 watt chip is difficult enough without making big changes in how servers are designed.

The wall of memory continues because SDRAM has not improved in performance nearly as fast as processors have. And with multicore chips, there are now many processors sharing the memory bandwidth from the same SDRAM. System designers can optimize potential bandwidth by having multiple memory controllers and pathways to memory, and this does help some of the time. But the problem still exists: in programs that are memory intensive and that have poor spatial locality, CPU cores will be stalled waiting for memory.

There are actually two other walls that haven't been mentioned but are equally important when considering the future of system designs: cost and history. These walls are related, as we shall see.

Multicore

The current designs for overcoming the walls of memory and power require having more than a single core in each CPU. Each core can run at a lower clock rate, and this reduces the power required. And by slowing down the processors, they won't be stalled for as many cycles—but they will still have to wait for memory. Clever cache design, having multiple threads backed by their own register sets, helps to hide memory latency by keeping the cores busy, as was done by Sun in their Niagara chips [2] and Intel with HyperThreading. But the problems of power and memory still exist.

Just designing chips with more cores is not likely to help. Intel's Nehalem Beckton (Xeon 7500) with eight cores, and AMD's Magny Cours [3] with 12 cores (but on two dies) use similar strategies to deal with the memory wall. Both have multiple paths to memory and lots of L3 cache (the Xeon has twice as much). I did some back-of-the-envelope calculations for the maximum memory bandwidth requirements for these chips, and, no surprise, memory cannot keep up. Of course, my exaggerated calculations don't match real-world applications. In the real world, memory access patterns—for example, does an application access only a small portion of memory that fits in cache or make almost random accesses across a large amount of memory?—are all over the board [4]. Faster clocks and more cores with more memory bandwidth do make a difference for high-end server applications.

But there are more problems with multicore designs today. On the power front, a helpful computer scientist pointed out to me that almost one-half of the power budget for current multicore chips goes to support, such as I/O, cache, and memory controllers. In a 120-watt chip, that leaves only 60 watts of power left for the cores themselves. In a 12-core design, that's 5 watts per core. If it were a 64-core design, there would only be about 0.9 watts per core—enough for relatively slow, in-order-execution designs like the Intel Atom, but not enough processing power for databases or High Performance Computing (HPC).

The same scientist mentioned that there are few applications today that can take advantage of more than four cores. Writing parallel applications is hard, as well as expensive, so only those applications, such as databases and specific HPC apps, are written to run in parallel. Unless and until there are good uses for more cores, having many cores will only be useful for the handful of applications that already support a lot of parallel execution.

Mixed Cores

Instead of today's multicore systems, I expect we will see more mixed cores. Sun's Niagara, Intel's Xeon, and AMD Magny Cours all have homogeneous cores—each core has the same architecture. Each core is a general-purpose processor, complete with floating point, integer, single-instruction, multiple-data (SIMD), and other capabilities. The design of core architectures is based on research into which general-purpose instructions get executed often and are worthwhile spending time and chip real estate to optimize.

Now imagine instead that code you will be running often has no floating point requirement in it at all. You could have processor cores without floating point support and focus on integer-only performance.

You might be thinking, "How silly is that?" but I did more than just think about it. I ran greps on the source code to the Barrelfish, MINIX 3, and Linux kernels, and lo and behold, kernels have very little floating point. None in Barrelfish and MINIX 3, and just support for emulating the floating point processor that was lacking in x386 chips and architecture support in the Linux kernel. It seems that kernel code, especially multikernels and microkernels, could run very well on integer-only cores.

Many userland utilities are integer-only as well. I took a look at the most recent version of Apache (2.2.15), and it turns out that several modules, including mod_ssl, use doubles, as does the Apache Portable Runtime utility library. Perhaps with some work (and for sites that don't require SSL), you could run Apache without floating point support in hardware.

Going to the other extreme, some applications require a mix of floating point and integer operations. Scientific applications commonly require both [4]. But the GPUs used in today's graphics cards, and starting to appear as co-processors, are specialized, floating point-only pipelines. The Xeon 7500 includes a GPU not on the same die as the cores but part of the same package. Perhaps processing units from GPUs might also wind up being cores within a future multicore chip.

The Other Walls

I mentioned that besides the walls of power and memory, there are the walls of cost and history. I don't have any references for these walls—but I didn't invent them, just named them.

Drastically changing CPU architectures, for example, to support heterogeneous cores, would have an enormous cost. This cost would involve having to create new operating systems, compilers, and support libraries that would support the ability to properly use specialized cores. While an operating system may already run properly on an integer-only core, many other applications would not. With compiler and library support, applications could have integer-only threads that could run in parallel with the rest of the application, taking advantage of specialized cores. But doing this involves developing not just the new CPUs, but also the operating systems, compilers, libraries, and perhaps new languages (or language extensions) required.

Then there is history. Any change in the way CPU architects design CPUs means that the way programmers work, system administrators manage, and distributions provide packages would all have to change as well. If some organization designed a heterogeneous, manycore CPU that blew the socks off today's hottest (and I mean hot) multicore CPUs, the new CPU would be useless without a lot of software to support it. Just writing that software would require programmers who thought differently about writing applications—for example, having integer-only threads whenever possible.

Fortunately, we do have several groups working on operating systems and libraries that can support heterogeneous cores. Barrelfish [5, 6] is designed to do this, although that has not yet been tested. Both MINIX 3 and seL4 intend to support a multikernel design that might be able to work with heterogeneous cores. Groups at UC Berkeley (Par Labs), CMU, Google, and (likely) Microsoft have researched running on heterogeneous multicore CPUs. So people are thinking about this.

Going back to cost, going against history has a cost as well. In the obvious case, the cost is replacing decades of code development with something radically new and different. And then there is the cost involved because people who have spent years working with the existing paradigms would have to change as well. When you have spent your career promoting a particular way of thinking about something, accepting a big change may have a totally unacceptable cost for many people. The cost of change has held back many huge discoveries and technical improvements, whether the topic is the earth revolving around the sun, germ theory, or CPU design.

Whether having integer-only processing cores makes sense is really not the point. We have used CPUs designed for general processing tasks for some very good reasons. The first is cost, in that making a family of closely related designs makes development and support (the compilers and libraries) cheaper. In the world of embedded systems, you can order systems-on-chip (SoC) with just the processing and I/O support features your application requires. But there will be extra cost for anything customized. Volume manufacturing drives price down quickly.

For now, I expect to see multicore CPUs with homogeneous cores. I don't expect to see cache-coherent systems with more than 16 cores (although you can already have quad-CPU systems with more than 16 cores). I say this because of the limitation on the amount of power available in single-die designs and the pressure on memory busses required by cache-coherency.

Designs like the Intel Single Chip Cloud use much simpler CPUs and dispense with cache-coherency in exchange for message passing. If you read the MINIX 3 article in this issue or read about Barrelfish in the April 2010 issue, you will see that message passing appears to be the way forward for multicore systems.

Lineup

Andrew Tanenbaum, with help from many co-authors, updates us on what is happening with MINIX 3. MINIX 3 is a modern operating system designed both as a platform for learning about OS design and as a real OS with a BSD-style license, with reliability, flexibility, and security highest on the feature list. I like what I read about the future directions of the MINIX 3 project. And where Linux (or Solaris or BSD) runs on every core in multicore CPUs, MINIX 3 not only has a much smaller code footprint but might someday require only some library code to run on each core, as Barrelfish does today (the dispatcher and CPU driver).

Jim Sangwine writes another article that looks to the near future. Sangwine's current research focus is on the use of 3D in Web applications. In his article, Sangwine looks at the leading contenders, Flash with the PaperScript library, WebGL, and O3D, using history as well as sample applications written using each library to test for performance and ease of programming.

Alva Couch takes us on a sociological safari into programming via ritual. Couch, like many of the older generation, tended to want to know everything about what they were doing—I certainly did. But today's programmers work much faster, taking advantage of Internet searches to find what works without needing to know *why* it works. Couch examines design patterns, examples, and rituals to distinguish these aids to programming, and he draws some conclusions about this new phenomenon.

Andy Seely takes us on a different journey, one into the realm of DoD sysadmin. Seely works for SAIC, and their customer needed a secure, easy-to-maintain (dead simple to maintain, really) and reliable replacement for a failed DNS server setup. Seely came up with a hardened Solaris VM that can be cloned, configured, and tested remotely, while being instantiated wherever needed just by using a VMware console.

Matt Simmons journeys back in time to LISA '09. Simmons shares his experience of attending his first LISA, and may perhaps encourage those who have yet to attend to head to San Jose for LISA '10.

Rudi van Drunen is back with a look at using small embedded systems. As an example, he focuses on Arduino, a small, inexpensive, open source platform that comes with an IDE for programming, lots of example code, and even plug-in modules (shields) for adding extensions to this simple processor.

David Blank-Edelman has written about libraries useful for creating dummy data. The article is not dumb, or just full of stuff, but demonstrates how you can create various types of test data quickly.

Peter Galvin shares his excitement about a new Sun/Oracle product, the Exadata V2. The Exadata V2 is both a database engine and a storage subsystem, but with a difference. Even the storage subsystem can carry out database operations, and the entire system can act as a transactional processing center and a data warehouse simultaneously—no mean feat. Delving into the hardware is part of what got me going about CPU futures (as did Tanenbaum's article).

Dave Josephsen writes about the use of packet capture tools for monitoring. Josephsen describes useful tools for capturing packets, including combining many sources into single archives using standard formats (flows and pcap), as well as one of my own favorite analysis tools, Argus.

Robert Ferrell covers his top five misconceptions about information security. In case you think he is just making this stuff up, he actually has special, super-secret techniques for collecting the information that becomes the ideas behind his columns. And no, he is not using Data::Generator.

Our main book reviewer, Elizabeth Zwicky, is taking a vacation this time. However, we have two excellent book reviews, one from Brandon Ching about a book on how organizations can use Second Life, and the other from Sam Stover giving us an in-depth look at a "short" (in Stover's terms) book about cloud security.

In conference reports, this issue covers FAST '10 and the SustainIT and TaPP workshops.

As regular readers of "Musings" may have noted, I fully expect the future of computing to be distributed. In the larger sense, this already exists in the forms of clouds, Web apps, Hadoop, and new frameworks such as Microsoft's Midori and .NET projects. But I also believe that the days of the homogeneous core processors are numbered, and I can only muse upon the possible futures of computing, from small embedded systems such as smart phones all the way up to high-end servers.

REFERENCES

[1] February 2010 "Musings": http://www.usenix.org/publications/login/2010-02/openpdfs/musings10-2.pdf.

[2] Richard McDougall and James Laudon, "Multi-Core Microprocessors Are Here," *;login:,* vol. 31, no. 5, Oct. 2006: http://www.usenix.org/publications/login/2006-10/pdfs/mcdougall.pdf.

[3] Nebojsa Novakovic, "Head to Head: Intel's Nehalem EX and AMD's Magny Cours," *The Inquirer*, April 6, 2010: http://www.theinquirer.net/inquirer/feature/1599367/head-head-intel-nehalem-ex-amd-magny-cours.

[4] R.C. Murphy and P.M. Kogge, "On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications": http://www.sandia.gov/~rcmurph/doc/ToC_56_7.pdf.

[5] Barrelfish: http://www.barrelfish.org.

[6] Rik Farrow, "The Barrelfish Multikernel: An Interview with Timothy Roscoe," *;login:*, vol. 35, no. 2, April 2010: http://www.usenix.org/publications/login/2010-04/pdfs/roscoe.pdf.