RIK FARROW

musings

rik@usenix.org

ONE OF THE ADVANTAGES OF BEING a "greybeard" is that I get to go on about just how hard things used to be. You know, like how I had to walk to school barefoot, in the snow, and uphill in both directions. That is a bit of an exaggeration, of course. I did have shoes. But I did also build my own first computer, and I modified the motherboard in my second as well.

I worked for a while at NorthStar Computers, a Berkeley startup whose first product was a disk controller for floppy drives. By the time I arrived in 1979, the company had been selling Horizon-2s both in kit and assembled forms. I got one of the last four kits and went to work.

The Horizon-2 [1] used a 6-MHz Z80, eight-bit processor on the motherboard, along with two serial ports, one parallel port, and twelve connectors on an S-100 bus. That "100" in the bus name means 100 pins per slot, so just soldering in the connectors meant 1200 careful solder joints. Disk storage was in the form of proprietary 5.25-inch floppy disks that held 90 kilobytes each, and memory maxed out at 64 kilobytes—although you couldn't use the last 8 kilobytes, as the ROM, containing the boot code and floppy disk drivers, overlapped 1 kilobyte of this memory space.

Later advances included double-sided drives with a whopping 360 kilobytes per disk, then a five-megabyte hard drive (which all by itself cost more than the entire system).

Instead of using NorthStar DOS, I would use CP/M, created by Digital Research, a company whose name might have become as famous as Microsoft except for some executive decisions made while dealing with IBM. I eventually added a surplus 20-MB hard drive, patched a driver into CP/M, and sold the souped-up Horizon-2 to a collective [2] which ran a grocery delivery business on this system for several years.

Fast Forward

Besides the observation that this primitive system, and ones like it, were used to run businesses, some other principles found in the Horizon-2 survive to this day. For example, every memory cell in DRAM [3] must be refreshed every 64 ms or less by reading in a row of memory, then rewriting it. Memory refresh occurs in parallel with the real work done by memory, supplying data to the

processor. In the Horizon and other S-100 bus designs, CPUs accessed memory synchronously, controlling all bus signals at the awesome speeds of up to 10 MHz.

I continued to use systems with CPUs running at less than 10 MHz through most of the 1980s. Compiling was slow but not impossible, because hard disks sped up disk-bound operations tremendously over floppy disks. But even at these slow bus speeds, access to memory was not instantaneous.

Before a row of memory can be read, the address that represents that row must be decoded. The decoding takes time, generally several bus-clock cycles (a delay called RAS; see [4] and [5]). After address decoding, the contents of memory are copied from sense registers to where they can be copied to the bus. The rate at which the actual copying of memory takes place depends on the type of memory, the width of the bus, and the frequency of the bus.

Over the years, CPU and DRAM chip developers have come up with schemes for improving the transfer rates between the CPU and DRAM. Bus frequencies have gotten faster, buses have gotten wider, and DRAMs have also gotten faster—but nowhere near as fast as CPUs have become. For example, a 3.2-GHz Xeon processor is running four times faster than its memory bus when various tricks, such as multiple reads per bus cycle, are taken into account.

Caches were designed to hide this discrepancy as much as possible. Level 1 cache resides on the processor die itself and runs at the same clock rate as the processor. You might find yourself wondering why CPU designers don't use more Level 1 cache than we typically see, but the problem is that chip designers must make several tradeoffs. Cache uses Static RAM (SRAM), which is much faster than DRAM, does not require refreshing, but also requires many more transistors to implement a single bit. More transistors means more die real estate, as well as more energy to support. Also, the process of address decoding is an issue for cache just as it is for DRAM, and, to speed up address decoding, various schemes that can quickly determine if a cache line holds a particular address are used.

If there is a Level 1 cache miss, perhaps the appropriate line of memory will be found in Level 2 cache, which exists off the CPU die but takes longer to access. Since this is also SRAM, it too is faster than DRAM, and Level 2 cache generally occupies a separate chip (or chips), so there can be more of it. But these memory accesses are slower than those for Level 1, forcing the CPU to wait for memory to become available. If the desired addresses in memory are not found in any cache, then DRAM must be accessed. And DRAM introduces both the RAS setup time and data transfer at the fastest rate that the memory bus supports. During this time, the processor may become idle as pipelines empty, and will continue idle until the wait state induced by slower memory ends.

Even 6MHz Z80 processors suffered wait states, and this problem has only gotten worse over time. Chip designers have worked around this problem as best they can, and we can see this in system benchmarks. That is, faster processors generally mean better benchmarks, but these benchmarks do not scale linearly. In other words, increasing the processor speeds by 20% do not result in real-world benchmarks increasing by 20%. In fact, the relation between processor speeds and benchmark performance varies all over the place, something you will notice if you read motherboard reviews. And memory is a major source of this variation. Other sources include the other buses and I/O devices.

Steve Johnson, during his Guru session at Annual Tech '06 in Boston, demonstrated that how you store and access elements in an array can make enormous differences in CPU-bound performance. Processing data elements stored sequentially in two different arrays was many times faster than accessing data elements that appeared separated by just a small number of bytes. Johnson's talk stirred up a lot of discussion, because it makes the points about caches, bus speeds, and memory concrete, as well as uncovering some interesting performance surprises.

The Fix

Fixing these performance bottlenecks poses a big problem for the leading CPU vendors, Intel and AMD. AMD has taken the lead currently through the use of faster bus designs. But is this the only way to go? Sun Microsystems engineers have bet that there is another way that will work well for some, although not all, applications.

I had heard about a new line of Sparc CPUs that include multiple cores, each able to quickly (one cycle) switch among four threads. The multi-threading capability of Sparc chips is old news, but having up to eight cores per chip was something new. I had learned about the new Sun designs at BSDCan (see summary in this issue) and read what I could find about the design before I came to Boston. Then, the first night there, Richard McDougall of Sun walked up with a T2000, a 2U rackmount system containing the new CPU. McDougall and Jim Maury used the T2000 during their Solaris performance and debugging class and set it up for use during Peter Galvin's Solaris classes, so people had plenty of time to "experience" the new Sun design.

What the Sun engineers said was that when running a highly threaded application such as Apache or Oracle, the eight cores, each running at 1.2 GHz, can operate without wait states as much as 80% of the time. This figure really had me wondering about the percentage of wait states for dual Xeon processors running at three times that clock rate. Whereas the rackmounted T2000 had the usual noisy-as-a-vacuum-cleaner fans, the processor itself has a short heat sink, with no fan on it, and it remained cool to the touch—a feature that Sun hopes will make this new twist on old Sparc technology popular where heat is an issue.

Of course, if your application is single-threaded and includes lots of floating-point calculations, then this design will not work well for you. (There is a single floating-point processor for all cores in the current multi-core Sparc, something that will change in the future.) However, since I have been writing and dreaming about how multi-core systems could be a much better design for future operating systems and for security, I was pretty excited about my face-to-face encounter with the T2000.

I have often written in my column that games, not business software, have been the strongest driving force behind the adoption of Windows and the design of faster PC hardware. If you're not playing first-person shooters, you don't need multi-gigahertz processors and fan-cooled video cards. I used word-processing software (WordStar) that worked quite well on a 10 Mhz processor for many years, writing code, documentation, and even books without using a system that could double as a space heater or high-end graphics workstation. There certainly are applications that do make excellent use of fast hardware today, such as Web browsers, Google Earth, and the many applications of cluster computing we hear about.

The point I'd like to leave with you is that CPU speed is not the current barrier to performance that it might seem to be. System performance relies on the entire system and will be limited by the slowest component or the slowest pathway to that component.

The Lineup

In the June issue, Kurt Chan wrote about disk types and architectures from the perspective of a NAS vendor. In this issue, Mark Uris writes about his experiences building a high-performance, multi-terabyte storage system at NCAR. The design of DataMonster uses nearline enterprise SATA drives, a design decision that at first glance appears to conflict with what Chan wrote. But I believe that the applications found at NCAR mainly result in large file transfers, not OLTP, which Chan cited as the main reason for building with enterprise drives.

Mark Burgess has written the first in what I hope will be a series of articles about configuration management. Darrell Fuhriman has written an article explaining identity management from the perspective of a sysadmin, and Andy Seely writes about administering systems used to support DoD operations and how that differs from normal sysadmin.

In the Security section, Eric Sorenson discusses creating your own network black hole as a technique for monitoring your network. Pete Herzog has written about Dru Lavigne's research into TCP/IP services, uncovering the history behind many of the obscure services you may have noticed.

Several authors of papers that appeared in the NSDI symposium (see summaries in this issue) have written articles about their projects. KyoungSoo Park and Vivek Pai write about CoBlitz, a Content Distribution Network (CDN) that works with unmodified Web browsers and servers; it performed faster than BitTorrent in real-world tests. Better yet, you can start using CoBlitz today. Ryan Peterson, Venugopalan Ramasubramanian, and Emin Gün Sirer write about Corona, a practical publish-subscribe system for the Web that solves the problems posed by RSS and current publish-subscribe systems.

In the Columns section, David Blank-Edelman begins a two-part series about the use of tie() to associate variables with databases, a practice some people (but not David) consider controversial. Robert Haskins discusses traffic-shaping tools that can be used both by ISPs and by any organization with a lot of Internet-bound traffic. Heison Chak explains the differences between soft and hard phones that are using VoIP. Finally, there is an excellent assortment of book reviews.

REFERENCES

[1] The NorthStar Horizon: http://www.old-computers.com/museum/computer.asp?c=50.

[2] The Purple Rose Collective: http://fic.ic.org/video/purpleroseinfo.php.

[3] Explanation of DRAM: http://en.wikipedia.org/wiki/DRAM.

[4] A clear explanation of different frontside bus types and what overclocking means: http://www.directron.com/fsbguide.html.

[5] Guide to understanding DRAM terms: http://www.dewassoc.com/performance/memory/memory_speeds.htm.