Musings

RIK FARROW

Rik is the editor of *;login:*.
rik@usenix.org

While preparing this issue of *;login:*, I found myself falling down a rabbit hole, like Alice in Wonderland. And when I hit bottom, all I could do was look around and puzzle about what I discovered there. My adventures started with a casual comment, made by an ex-Cray Research employee, about the design of current supercomputers. He told me that today's supercomputers cannot perform some of the tasks that they are designed for, and used weather forecasting as his example.

I was stunned. Could this be true? Or was I just being dragged down some fictional rabbit hole? I decided to learn more about supercomputer history.

Supercomputers

It is humbling to learn about the early history of computer design. Things we take for granted, such as pipelining instructions and vector processing, were important inventions in the 1970s. The first supercomputers were built from discrete components—that is, transistors soldered to circuit boards—and had clock speeds in the tens of nanoseconds. To put that in real terms, the Control Data Corporation's (CDC) 7600 had a clock cycle of 27.5 ns, or in today's terms, 36.4 MHz. This was CDC's second supercomputer (the 6600 was first), but included instruction pipelining, an invention of Seymour Cray. The CDC 7600 peaked at 36 MFLOPS, but generally got 10 MFLOPS with carefully tuned code.

The other cool thing about the CDC 7600 was that it broke down at least once a day. This was actually a pretty common feature for '70s computers, and something I am old enough to remember.

Seymour Cray wanted to start over with a new design, rather than build on the CDC 7600, and when he was unable to do that at CDC, he started his own company, Cray Research (CR). CR's first design used some integrated circuits, but only very simple ones, like NOR gates and static RAM chips (1024 *bits* at 50 ns). Besides instruction pipelining, the Cray 1 was superscalar; that is, it could complete multiple instructions per clock cycle. Like the CDC 7600, it used a single address space, although the Cray 1 had 16 banks of interleaved memory, allowing four 64-bit words to be read per clock cycle—12.5 ns or 80MHZ. The Cray 1 peaked at 136 MFLOPS, and NCAR continued to use this Cray 1 for weather forecasting until 1986.

To put these levels of performance into perspective, I took a look at the European Centre for Medium-Range Weather Forecasts (ECMWF) supercomputer history Web page [1]. In 1976, it took the CDC 6600 12 days to produce a 10-day forecast, which doesn't sound terribly useful to me. But the ECMWF considered this

hopeful enough to acquire a Cray-1A in 1978. With the Cray-1A, they managed to produce that same 10-day forecast in just five hours.

The ECMWF continued buying and using Crays until 1996. Each new Cray came with more CPUs, more memory, faster clock speeds, and more processing power. But they all shared a critical design element: shared memory. For example, all of the 16 CPUs in the Cray Y-MP shared a total of 16 GB. The Cray T3D was a big departure from this design, as its 128 Alpha processors each had their own 128 MBs of memory. Memory was now distributed among the processors, and the processors were connected together in a torus for fast inter-processor communication.

It was this change, from a shared memory architecture to a distributed memory model, that I believe the Cray guy was concerned with. The ECMWF had to rewrite their software so that it would run well on a distributed memory architecture [1], while all previous forecasting software relied on a shared memory model. The difference is that with shared memory, any data in memory is equally "far" away from any CPU (has similar latency), while in distributed memory, some data requires much longer to fetch, because it "belongs" to another CPU and is also physically separated. The "belonging to" is an important issue, as writes to memory must be coordinated, so that one CPU does not change data while another CPU is using that data—an important issue even with today's multicore desktop CPUs.

The ECMWF and UCAR [2] today use supercomputers based on IBM's POWER5 or POWER6-based clusters. These clusters include CPU nodes (pairs of CPUs) that share memory, but are also connected to other nodes in the same rack over a high speed interconnect, and then are connected to other racks using InfiniBand. So these supercomputers have both shared memory and distributed memory. Having both memory models in the same supercomputer makes programming more difficult, as programmers need to be aware of the differences in latency when fetching "near" data (shared) and "far" (distributed, across a network).

By this time, I was pretty convinced that my acquaintance had misled me into thinking that the supercomputers of old were superior to the supercomputers of today. Instead, what I found is that the programming models had to change, to combine both the techniques designed for the older, shared memory designs and the newer technique for working with distributed memory correctly and efficiently.

Clusters

Today's supercomputers are clusters that combine the shared memory of '70s supercomputers with distributed memory models. Just check out the Top 500 supercomputer list [3] and you will see that the world's fastest computers are clusters that combined shared memory models with fast interconnects to distributed memory. The number two (in November 2010) was a Cray XT5-HE at Oak Ridge National labs, composed of 37,336 six-core 2.6 GHz AMD Opterons (using shared memory) connected together using a proprietary interconnect [4].

You may be wondering why I was even bothering with this wild goose chase. I found the discussions of early supercomputers (just search for "Cray") actually quite interesting and helpful in understanding current supercomputer designs. And as you read this issue of *;login:*, you will notice that clusters, and even supercomputers, have become more common.

MapReduce requires a loosely organized cluster of commodity computers to operate, as does Microsoft's Dryad. When you read the summaries for LISA, you will learn about the trials of the sysadmin in charge of getting a supercomputing cluster interconnect working properly, when the interconnect hardware was limited to a single supplier—one whose key product didn't work very reliably.

There's certainly more that can be written about this topic, and I do have more research to complete before I am willing to write more. Stayed tuned...

The Lineup

Derek Murray and Steven Hand provide us with a crystal clear view of the different ways to program for large clusters. Each method has limitations, with MPI, the most popular library used in scientific programming, requiring a lot of knowledge of the cluster and node architecture. The other big branch in programming for clusters uses distributed execution engines. These range from completely unordered systems, such as SETI@home, to MapReduce, with some ordering, and Dryad, with explicit dependencies. Murray and Hand introduce Skywriting and Ciel, a scripting language and an execution engine that support dynamic dependencies, something none of the other distributed executions can do. Skywriting supports iteration, as well as the simple and explicit ordering of MapReduce and Dryad, providing more freedom to the programmer, without requiring that the programmer understand the architecture of the cluster.

As this issue of *;login:* includes summaries of LISA '10, we have several articles related to LISA. First up, Cory Lueninghoener presents strategies for getting started with configuration management. Learning any software tool takes time and effort, and configuration management incorporates additional obstacles to implementation: homebrew systems and scripts, the coworkers who have built (and probably love) these systems, control issues as configuration management takes over loosely managed systems, as well as the threat of a massive failure during the early implementation phases. Lueninghoener presents advice for dealing with each of these issues, as well as simple strategies for getting up and running smoothly using any of the popular configuration management tools.

Jim Donn and Tim Hartmann share their journey into using Splunk for logging. I found I barely understood Splunk at all before reading their article (and listening to their IT [5]). Donn and Hartmann, like Lueninghoener, provide a roadmap for moving from a barely functioning logging system to one that supports many different user communities. This case study can help you understand Splunk, and may also serve as a warning to you: it seems that Splunk has addictive qualities, which actually speaks very highly of its usability.

Doug Hughes, a Program Committee Co-Chair of LISA '11, tells us why old styles of backups just don't work anymore. In this interview, Hughes explains the systems they use at D. E. Shaw Research, where the focus is on supporting custom supercomputers used for molecular dynamics simulation. This work produces continuous streams of results, all of which must be reliably stored and archived. Hughes describes both a past and the present solution they are using to back up millions of files and tens of terabytes of new data every month.

Ole Tange has another GNU tool to share this time. In the February 2011 issue Tange described Parallel, an improved version of xargs. In this article,Tange explains DBURL, a part of Parallel that unifies access to databases.

David Blank-Edelman decided that his last column, about transferring data, just didn't take us far enough. After all, ftp and rsync are soooo last century. In this column David shows us how to use Perl modules for data transfer, all with his gentle humor and easy style.

Pete Galvin suggests that those who were once interested in Solaris take a look at Solaris Express 11. Solaris Express represents the next release of Solaris, which Oracle has promised for sometime later this year. Solaris Express incorporates Open Solaris features, and Pete covers features he hasn't discussed in previous columns. He also provides analysis on just why you might be interested in a Solaris coming from Oracle.

Dave Josephson waxes eloquent about the reality of clusters and his utter horror at the reality of the hardware. Well, not quite, but close enough. Dave then cuts to the chase with some important advice about monitoring widely distributed clusters, something he must do almost every day (he does take time off, or at least pretends to).

Robert Ferrell takes us for a ride, ranging from a rant about the future (quaternary computing) to civility in the US Congress, then ramping down to poke holes in the belief systems of a certain vendor.

Elizabeth Zwicky has two book reviews, written with her usual flair, followed by three by Sam Stover (one a security book, the other two related to Arduino and Zigbee). Evan Teran took the plunge into a book on kernel exploitation and survived to tell us about it. I review just one book, as the one I really wanted to review still has me wondering just what to say about it. I guess you will just have to wait until the next issue...

We have the conference reports from LISA '10. Do keep in mind that you can watch videos of most of the presentations of LISA's three tracks, and some presentations, such as David Blank-Edelman's closing session, are much better experienced then read about.

References

[1] European Centre for Medium-Range Weather Forecasts (ECMWF), supercomputer history: http://www.ecmwf.int/services/computing/overview/supercomputer_history.html.

[2] UCAR Bluefire supercomputer: http://www2.cisl.ucar.edu/docs/bluefire/system-information-overview.

[3] The Top 500 supercomputers: http://www.top500.org/.

[4] Jaguar supercomputer: https://secure.wikimedia.org/wikipedia/en/wiki/Jaguar_%28computer%29.

[5] LISA 2010 Tech: http://www.usenix.org/events/lisa10/tech/#thurs.