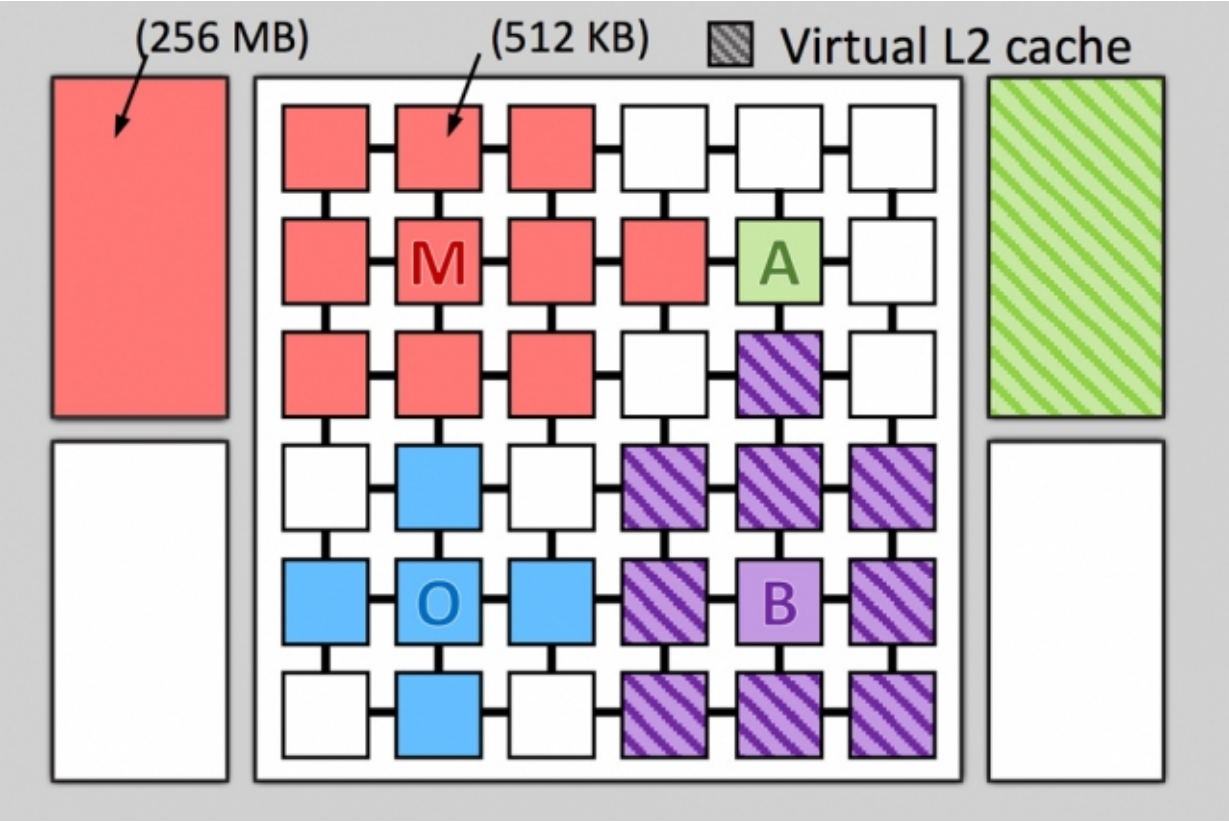
Generating ad hoc 'cache hierarchies' increases chip processing speed while reducing energy consumption

July 10 2017, by Larry Hardesty



This figure shows a 36-tile Jenga system that's running four applications. Jenga gives each application a custom virtual cache hierarchy. Credit: Massachusetts Institute of Technology

For decades, computer chips have increased efficiency by using "caches," small, local memory banks that store frequently used data and cut down on time- and energy-consuming communication with off-chip memory.

Today's chips generally have three or even four different levels of cache, each of which is more capacious but slower than the last. The sizes of the caches represent a compromise between the needs of different kinds of programs, but it's rare that they're exactly suited to any one program.

Researchers at MIT's Computer Science and Artificial Intelligence Laboratory have designed a system that reallocates cache access on the fly, to create new "cache hierarchies" tailored to the needs of particular programs.

The researchers tested their system on a simulation of a chip with 36 cores, or processing units. They found that, compared to its best-performing predecessors, the system increased processing speed by 20 to 30 percent while reducing energy consumption by 30 to 85 percent.

"What you would like is to take these distributed physical memory resources and build application-specific hierarchies that maximize the performance for your particular application," says Daniel Sanchez, an assistant professor in the Department of Electrical Engineering and Computer Science (EECS), whose group developed the new system.

"And that depends on many things in the application. What's the size of the data it accesses? Does it have hierarchical reuse, so that it would benefit from a hierarchy of progressively larger memories? Or is it scanning through a data structure, so we'd be better off having a single but very large level? How often does it access data? How much would its performance suffer if we just let data drop to main memory? There are all these different tradeoffs."

Sanchez and his coauthors—Po-An Tsai, a graduate student in EECS at MIT, and Nathan Beckmann, who was an MIT graduate student when the work was done and is now an assistant professor of computer science at Carnegie Mellon University—presented the new system, dubbed Jenga, at the International Symposium on Computer Architecture last week.

Staying local

For the past 10 years or so, improvements in computer chips' processing power have come from the addition of more cores. The chips in most of today's desktop computers have four cores, but several major chipmakers have announced plans to move to six cores in the next year or so, and 16-core processors are not uncommon in high-end servers. Most industry watchers assume that the core count will continue to climb.

Each core in a multicore chip usually has two levels of private cache. All the cores share a third cache, which is actually broken up into discrete memory banks scattered around the chip. Some new chips also include a so-called DRAM cache, which is etched into a second chip that is mounted on top of the first.

For a given core, accessing the nearest memory bank of the shared cache is more efficient than accessing more distant cores. Unlike today's cache management systems, Jenga distinguishes between the physical locations of the separate memory banks that make up the shared cache. For each core, Jenga knows how long it would take to retrieve information from any on-chip memory bank, a measure known as "latency."

Jenga builds on an earlier system from Sanchez's group, called Jigsaw, which also allocated cache access on the fly. But Jigsaw didn't build cache hierarchies, which makes the allocation problem much more

complex.

For every task running on every core, Jigsaw had to calculate a latency-space curve, which indicated how much latency the core could expect with caches of what size. It then had to aggregate all those curves to find a space allocation that minimized latency for the chip as a whole.

Curves to surfaces

But Jenga has to evaluate the tradeoff between latency and space for two layers of cache simultaneously, which turns the two-dimensional latency-space curve into a three-dimensional surface. Fortunately, that surface turns out to be fairly smooth: It may undulate, but it usually won't have sudden, narrow spikes and dips.

That means that sampling points on the surface will give a pretty good sense of what the surface as a whole looks like. The researchers developed a clever sampling algorithm tailored to the problem of cache allocation, which systematically increases the distances between sampled points. "The insight here is that caches with similar capacities—say, 100 megabytes and 101 megabytes—usually have similar performance," Tsai says. "So a geometrically increased sequence captures the full picture quite well."

Once it has deduced the shape of the surface, Jenga finds the path across it that minimizes latency. Then it extracts the component of that path contributed by the first level of cache, which is a 2-D curve. At that point, it can reuse Jigsaw's space-allocation machinery.

In experiments, the researchers found that this approach yielded an aggregate space allocation that was, on average, within 1 percent of that produced by a full-blown analysis of the 3-D surface, which would be prohibitively time consuming. Adopting the computational short cut

enables Jenga to update its memory allocations every 100 milliseconds, to accommodate changes in programs' memory-access patterns.

End run

Jenga also features a data-placement procedure motivated by the increasing popularity of DRAM cache. Because they're close to the cores accessing them, most caches have virtually no bandwidth restrictions: They can deliver and receive as much data as a core needs. But sending data longer distances requires more energy, and since DRAM caches are off-chip, they have lower data rates.

If multiple cores are retrieving data from the same DRAM cache, this can cause bottlenecks that introduce new latencies. So after Jenga has come up with a set of cache assignments, cores don't simply dump all their data into the nearest available memory bank. Instead, Jenga parcels out the data a little at a time, then estimates the effect on bandwidth consumption and latency. Thus, even within the 100-millisecond intervals between chip-wide cache re-allocations, Jenga adjusts the priorities that each core gives to the memory banks allocated to it.

"There's been a lot of work over the years on the right way to design a cache hierarchy," says David Wood, a professor of computer science at the University of Wisconsin at Madison. "There have been a number of previous schemes that tried to do some kind of dynamic creation of the hierarchy. Jenga is different in that it really uses the software to try to characterize what the workload is and then do an optimal allocation of the resources between the competing processes. And that, I think, is fundamentally more powerful than what people have been doing before. That's why I think it's really interesting."

More information: Jenga: Sotware-Defined Cache Hierarchies. people.csail.mit.edu/sanchez/p … /2017.jenga.isca.pdf