More efficient memory-management scheme could help enable chips with thousands of cores

September 10 2015, by Larry Hardesty

In a modern, multicore chip, every core—or processor—has its own small memory cache, where it stores frequently used data. But the chip

also has a larger, shared cache, which all the cores can access.

If one core tries to update data in the shared cache, other cores working on the same data need to know. So the shared cache keeps a directory of which cores have copies of which data.

That directory takes up a significant chunk of memory: In a 64-core chip, it might be 12 percent of the shared cache. And that percentage will only increase with the core count. Envisioned chips with 128, 256, or even 1,000 cores will need a more efficient way of maintaining cache coherence.

At the International Conference on Parallel Architectures and Compilation Techniques in October, MIT researchers unveil the first fundamentally new approach to cache coherence in more than three decades. Whereas with existing techniques, the directory's memory allotment increases in direct proportion to the number of cores, with the new approach, it increases according to the logarithm of the number of cores.

In a 128-core chip, that means that the new technique would require only one-third as much memory as its predecessor. With Intel set to release a 72-core high-performance chip in the near future, that's a more than hypothetical advantage. But with a 256-core chip, the space savings rises to 80 percent, and with a 1,000-core chip, 96 percent.

When multiple cores are simply reading data stored at the same location, there's no problem. Conflicts arise only when one of the cores needs to update the shared data. With a directory system, the chip looks up which cores are working on that data and sends them messages invalidating their locally stored copies of it.

"Directories guarantee that when a write happens, no stale copies of the

data exist," says Xiangyao Yu, an MIT graduate student in electrical engineering and computer science and first author on the new paper. "After this write happens, no read to the previous version should happen. So this write is ordered after all the previous reads in physical-time order."

Time travel

What Yu and his thesis advisor—Srini Devadas, the Edwin Sibley Webster Professor in MIT's Department of Electrical Engineering and Computer Science—realized was that the physical-time order of distributed computations doesn't really matter, so long as their logical-time order is preserved. That is, core A can keep working away on a piece of data that core B has since overwritten, provided that the rest of the system treats core A's work as having preceded core B's.

The ingenuity of Yu and Devadas' approach is in finding a simple and efficient means of enforcing a global logical-time ordering. "What we do is we just assign time stamps to each operation, and we make sure that all the operations follow that time stamp order," Yu says.

With Yu and Devadas' system, each core has its own counter, and each data item in memory has an associated counter, too. When a program launches, all the counters are set to zero. When a core reads a piece of data, it takes out a "lease" on it, meaning that it increments the data item's counter to, say, 10. As long as the core's internal counter doesn't exceed 10, its copy of the data is valid. (The particular numbers don't matter much; what matters is their relative value.)

When a core needs to overwrite the data, however, it takes "ownership" of it. Other cores can continue working on their locally stored copies of the data, but if they want to extend their leases, they have to coordinate with the data item's owner. The core that's doing the writing increments

its internal counter to a value that's higher than the last value of the data item's counter.

Say, for instance, that cores A through D have all read the same data, setting their internal counters to 1 and incrementing the data's counter to 10. Core E needs to overwrite the data, so it takes ownership of it and sets its internal counter to 11. Its internal counter now designates it as operating at a later logical time than the other cores: They're way back at 1, and it's ahead at 11. The idea of leaping forward in time is what gives the system its name—Tardis, after the time-traveling spaceship of the British science fiction hero Dr. Who.

Now, if core A tries to take out a new lease on the data, it will find it owned by core E, to which it sends a message. Core E writes the data back to the shared cache, and core A reads it, incrementing its internal counter to 11 or higher.

Unexplored potential

In addition to saving space in memory, Tardis also eliminates the need to broadcast invalidation messages to all the cores that are sharing a data item. In massively multicore chips, Yu says, this could lead to performance improvements as well. "We didn't see performance gains from that in these experiments," Yu says. "But that may depend on the benchmarks"—the industry-standard programs on which Yu and Devadas tested Tardis. "They're highly optimized, so maybe they already removed this bottleneck," Yu says.

"There have been other people who have looked at this sort of lease idea," says Christopher Hughes, a principal engineer at Intel Labs, "but at least to my knowledge, they tend to use physical time. You would give a lease to somebody and say, 'OK, yes, you can use this data for, say, 100 cycles, and I guarantee that nobody else is going to touch it in that

amount of time.' But then you're kind of capping your performance, because if somebody else immediately afterward wants to change the data, then they've got to wait 100 cycles before they can do so. Whereas here, no problem, you can just advance the clock. That is something that, to my knowledge, has never been done before. That's the key idea that's really neat."

Hughes says, however, that chip designers are conservative by nature. "Almost all mass-produced commercial systems are based on directory-based protocols," he says. "We don't mess with them because it's so easy to make a mistake when changing the implementation."

But "part of the advantage of their scheme is that it is conceptually somewhat simpler than current [directory-based] schemes," he adds. "Another thing that these guys have done is not only propose the idea, but they have a separate paper actually proving its correctness. That's very important for folks in this field."

More information: "Tardis: Time Traveling Coherence Algorithm for Distributed Shared Memory."
people.csail.mit.edu/devadas/pubs/tardis.pdf

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology