Parallel programming may not be so daunting

March 24 2014, by Larry Hardesty

Credit: Thinkstock

Computer chips have stopped getting faster: The regular performance improvements we've come to expect are now the result of chipmakers' adding more cores, or processing units, to their chips, rather than increasing their clock speed.

In theory, doubling the number of cores doubles the chip's efficiency,

but splitting up computations so that they run efficiently in parallel isn't easy. On the other hand, say a trio of computer scientists from MIT, Israel's Technion, and Microsoft Research, neither is it as hard as had been feared.

Commercial software developers writing programs for multicore chips frequently use so-called "lock-free" parallel algorithms, which are relatively easy to generate from standard sequential code. In fact, in many cases the conversion can be done automatically.

Yet lock-free algorithms don't come with very satisfying theoretical guarantees: All they promise is that at least one core will make progress on its computational task in a fixed span of time. But if they don't exceed that standard, they squander all the additional computational power that multiple cores provide.

In recent years, theoretical computer scientists have demonstrated ingenious alternatives called "wait-free" algorithms, which guarantee that all cores will make progress in a fixed span of time. But deriving them from sequential code is extremely complicated, and commercial developers have largely neglected them.

In a paper to be presented at the Association for Computing Machinery's Annual Symposium on the Theory of Computing in May, Nir Shavit, a professor in MIT's Department of Electrical Engineering and Computer Science; his former student Dan Alistarh, who's now at Microsoft Research; and Keren Censor-Hillel of the Technion demonstrate a new analytic technique suggesting that, in a wide range of real-world cases, lock-free algorithms actually give wait-free performance.

"In practice, programmers program as if everything is wait-free," Shavit says. "This is a kind of mystery. What we are exposing in the paper is this little-talked-about intuition that programmers have about how [chip]

schedulers work, that they are actually benevolent."

The researchers' key insight was that the chip's performance as a whole could be characterized more simply than the performance of the individual cores. That's because the allocation of different "threads," or chunks of code executed in parallel, is symmetric. "It doesn't matter whether thread 1 is in state A and thread 2 is in state B or if you just swap the states around," says Alistarh, who contributed to the work while at MIT. "What we noticed is that by coalescing symmetric states, you can simplify this a lot."

In a real chip, the allocation of threads to cores is "a complex interplay of latencies and scheduling policies," Alistarh says. In practice, however, the decisions arrived at through that complex interplay end up looking a lot like randomness. So the researchers modeled the scheduling of threads as a process that has at least a little randomness in it: At any time, there's some probability that a new thread will be initiated on any given core.

The researchers found that even with a random scheduler, a wide range of lock-free algorithms offered performance guarantees that were as good as those offered by wait-free algorithms.

That analysis held, no matter how the probability of thread assignment varied from core to core. But the researchers also performed a more specific analysis, asking what would happen when multiple cores were trying to write data to the same location in memory and one of them kept getting there ahead of the others. That's the situation that results in a lock-free algorithm's worst performance, when only one core is making progress.

For that case, they considered a particular set of probabilities, in which every core had the same chance of being assigned a thread at any given

time. "This is kind of a worst-case random scheduler," Alistarh says. Even then, however, the number of cores that made progress never dipped below the square root of the number of cores assigned threads, which is still better than the minimum performance guarantee of lock-free algorithms.

More information: Paper: "Are Lock-Free Concurrent Algorithms Practically Wait-Free?" arxiv.org/pdf/1311.3200v2

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