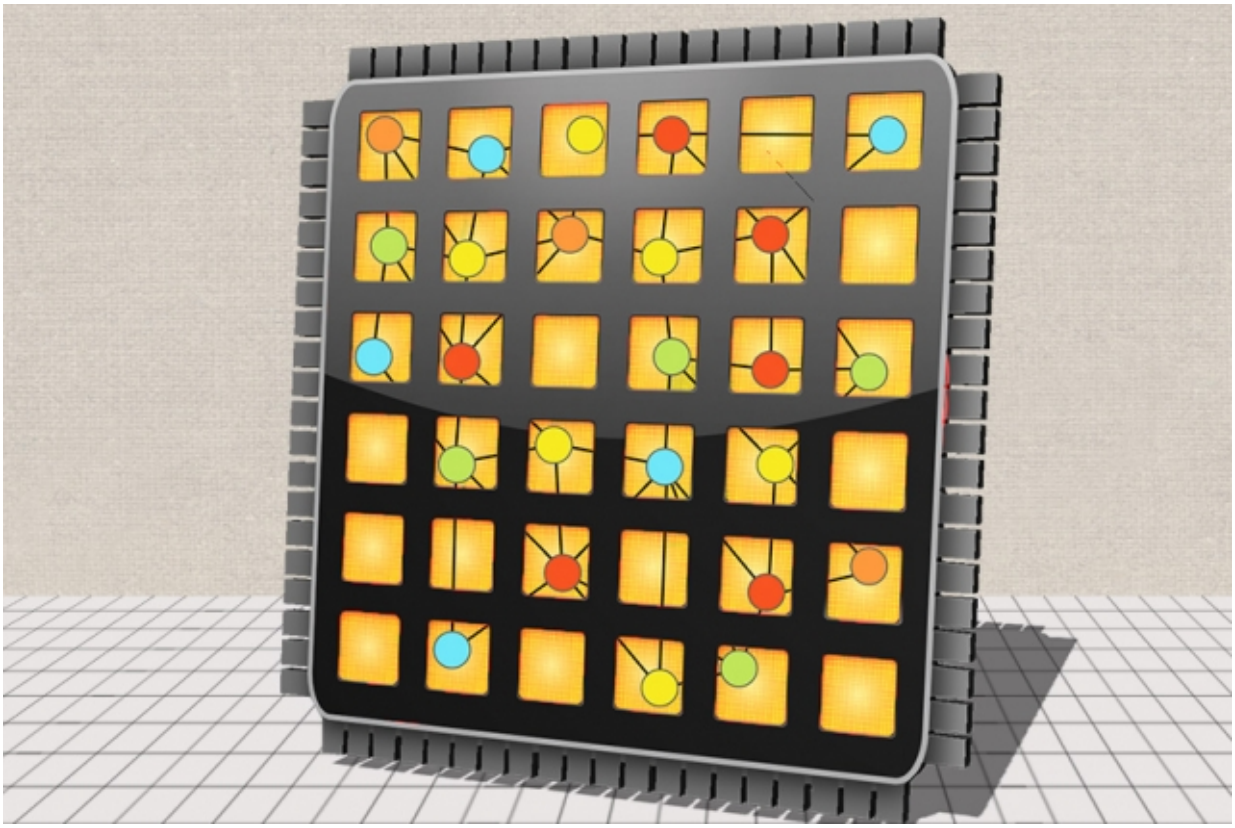
New chip design makes parallel programs run many times faster and requires one-tenth the code

June 20 2016, by Larry Hardesty



“Multicore systems are really hard to program,” says Daniel Sanchez, an assistant professor in MIT’s Department of Electrical Engineering and Computer Science. “You have to explicitly divide the work that you’re doing into tasks, and then you need to enforce some synchronization between tasks accessing shared data. What this architecture does, essentially, is to remove all sorts of explicit synchronization, to make parallel programming much easier.” Credit: Christine

Daniloff/MIT

Computer chips have stopped getting faster. For the past 10 years, chips' performance improvements have come from the addition of processing units known as cores.

In theory, a program on a 64-core machine would be 64 times as fast as it would be on a single-core machine. But it rarely works out that way. Most computer programs are sequential, and splitting them up so that chunks of them can run in parallel causes all kinds of complications.

In the May/June issue of the Institute of Electrical and Electronics Engineers' journal Micro, researchers from MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) will present a new chip design they call Swarm, which should make parallel programs not only much more efficient but easier to write, too.

In simulations, the researchers compared Swarm versions of six common algorithms with the best existing parallel versions, which had been individually engineered by seasoned software developers. The Swarm versions were between three and 18 times as fast, but they generally required only one-tenth as much code—or even less. And in one case, Swarm achieved a 75-fold speedup on a program that computer scientists had so far failed to parallelize.

"Multicore systems are really hard to program," says Daniel Sanchez, an assistant professor in MIT's Department of Electrical Engineering and Computer Science, who led the project. "You have to explicitly divide the work that you're doing into tasks, and then you need to enforce some synchronization between tasks accessing shared data. What this architecture does, essentially, is to remove all sorts of explicit

synchronization, to make parallel programming much easier. There's an especially hard set of applications that have resisted parallelization for many, many years, and those are the kinds of applications we've focused on in this paper."

Many of those applications involve the exploration of what computer scientists call graphs. A graph consists of nodes, typically depicted as circles, and edges, typically depicted as line segments connecting the nodes. Frequently, the edges have associated numbers called "weights," which might represent, say, the strength of correlations between data points in a data set, or the distances between cities.

Graphs crop up in a wide range of computer science problems, but their most intuitive use may be to describe geographic relationships. Indeed, one of the algorithms that the CSAIL researchers evaluated is the standard algorithm for finding the fastest driving route between two points.

Setting priorities

In principle, exploring graphs would seem to be something that could be parallelized: Different cores could analyze different regions of a graph or different paths through the graph at the same time. The problem is that with most graph-exploring algorithms, it gradually becomes clear that whole regions of the graph are irrelevant to the problem at hand. If, right off the bat, cores are tasked with exploring those regions, their exertions end up being fruitless.

Of course, fruitless analysis of irrelevant regions is a problem for sequential graph-exploring algorithms, too, not just parallel ones. So computer scientists have developed a host of application-specific techniques for prioritizing graph exploration. An algorithm might begin by exploring just those paths whose edges have the lowest weights, for

instance, or it might look first at those nodes with the lowest number of edges.

What distinguishes Swarm from other multicore chips is that it has extra circuitry for handling that type of prioritization. It time-stamps tasks according to their priorities and begins working on the highest-priority tasks in parallel. Higher-priority tasks may engender their own lower-priority tasks, but Swarm slots those into its queue of tasks automatically.

Occasionally, tasks running in parallel may come into conflict. For instance, a task with a lower priority may write data to a particular memory location before a higher-priority task has read the same location. In those cases, Swarm automatically backs out the results of the lower-priority tasks. It thus maintains the synchronization between cores accessing the same data that programmers previously had to worry about themselves.

Indeed, from the programmer's perspective, using Swarm is pretty painless. When the programmer defines a function, he or she simply adds a line of code that loads the function into Swarm's queue of tasks. The programmer does have to specify the metric—such as edge weight or number of edges—that the program uses to prioritize tasks, but that would be necessary, anyway. Usually, adapting an existing sequential algorithm to Swarm requires the addition of only a few lines of code.

Keeping tabs

The hard work falls to the chip itself, which Sanchez designed in collaboration with Mark Jeffrey and Suvinay Subramanian, both MIT graduate students in electrical engineering and computer science; Cong Yan, who did her master's as a member of Sanchez's group and is now a PhD student at the University of Washington; and Joel Emer, a professor

of the practice in MIT's Department of Electrical Engineering and Computer Science, and a senior distinguished research scientist at the chip manufacturer NVidia.

The Swarm chip has extra circuitry to store and manage its queue of tasks. It also has a circuit that records the memory addresses of all the data its cores are currently working on. That circuit implements something called a Bloom filter, which crams data into a fixed allotment of space and answers yes/no questions about its contents. If too many addresses are loaded into the filter, it will occasionally yield false positives—indicating "yes, I'm storing that address"—but it will never yield false negatives.

The Bloom filter is one of several circuits that help Swarm identify memory access conflicts. The researchers were able to show that time-stamping makes synchronization between cores easier to enforce. For instance, each data item is labeled with the time stamp of the last task that updated it, so tasks with later time-stamps know they can read that data without bothering to determine who else is using it.

Finally, all the cores occasionally report the time stamps of the highest-priority tasks they're still executing. If a core has finished tasks that have earlier time stamps than any of those reported by its fellows, it knows it can write its results to memory without courting any conflicts.

"I think their architecture has just the right aspects of past work on transactional memory and thread-level speculation," says Luis Ceze, an associate professor of computer science and engineering at the University of Washington. "'Transactional memory' refers to a mechanism to make sure that multiple processors working in parallel don't step on each other's toes. It guarantees that updates to shared memory locations occur in an orderly way. Thread-level speculation is a related technique that uses transactional-memory ideas for

parallelization: Do it without being sure the task is parallel, and if it's not, undo and re-execute serially. Sanchez's architecture uses many good pieces of those ideas and technologies in a creative way."

More information: Mark C. Jeffrey et al. Unlocking Ordered Parallelism with the Swarm Architecture, *IEEE Micro* (2016). DOI: 10.1109/MM.2016.12

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