ACM CAREERS

# Practical Parallelism

By MIT News
July 5, 2017
Comments



MORE NEWS & OPINIONS

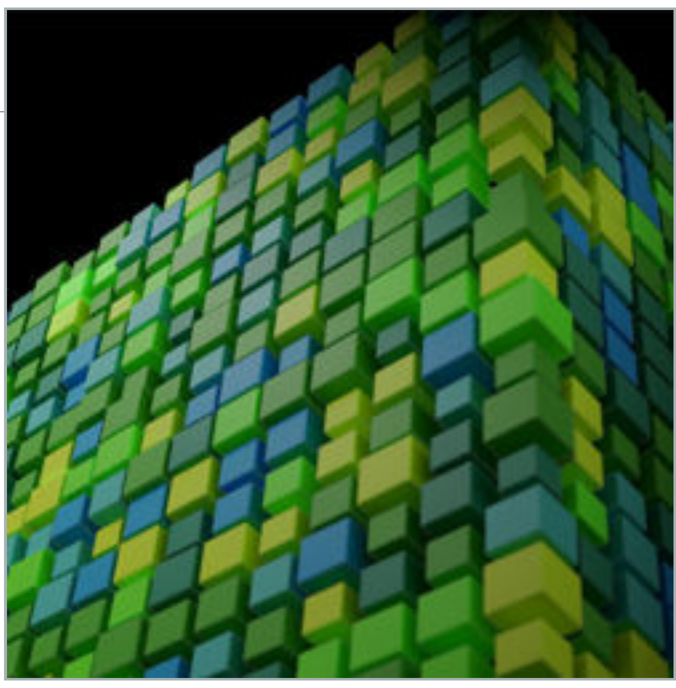
US to Create Independent Military Cyber Command
The Associated Press

Facebook's Belated Awakening
The New York Times

ACM-ICPC Has Outgrown Its Humble Start
Kelsey Sinclair

ACM RESOURCES

Understanding DB2: Learning Visually with Examples
Courses



The chips in most modern desktop computers have four "cores," or processing units, which can run different computational tasks in parallel. But the chips of the future could have dozens or even hundreds of cores, and taking advantage of all that parallelism is a stiff challenge.

Researchers from MIT's Computer Science and Artificial Intelligence Laboratory have developed a new system that not only makes parallel programs run much more efficiently but also makes them easier to code.

In tests on a set of benchmark algorithms that are standard in the field, the researchers' new system frequently enabled more than 10-fold speedups over existing systems that adopt the same parallelism strategy, with a maximum of 88-fold.

For instance, algorithms for solving an important problem called max flow have proven very difficult to parallelize. After decades of research, the best parallel implementation of one common max-flow algorithm achieves only an eightfold speedup when it's run on 256 parallel processors. With the researchers' new system, the improvement is 322-fold — and the program required only one-third as much code.

The new system, dubbed Fractal, achieves those speedups through a parallelism strategy known as speculative execution.

"In a conventional parallel program, you need to divide your work into tasks," says Daniel Sanchez, an assistant professor of electrical engineering and computer science at MIT and senior author on the new paper. "But because these tasks are operating on shared data, you need to introduce some synchronization to ensure that the data dependencies that these tasks have are respected. From the mid-90s to the late 2000s, there were multiple waves of research in what we call speculative architectures. What these systems do is execute these different chunks in parallel, and if they detect a conflict, they abort and roll back one of them."

Constantly aborting computations before they complete would not be a very efficient parallelization strategy. But for many applications, aborted computations are rare enough that they end up squandering less time than the complicated checks and updates required to synchronize tasks in more conventional parallel schemes. Last year, Sanchez's group reported a system, called Swarm, that extended speculative parallelism to an important class of computational problems that involve searching data structures known as graphs.

## 'Atomic' Tasks

Research on speculative architectures, however, has often run aground on the problem of "atomicity." Like all parallel architectures, speculative architectures require the programmer to divide programs into tasks that can run simultaneously. But with speculative architectures, each such task is "atomic," meaning that it should seem to execute as a single whole. Typically, each atomic task is assigned to a separate processing unit, where it effectively runs in isolation.

Atomic tasks are often fairly substantial. The task of booking an airline flight online, for instance, consists of many separate operations, but they have to be treated as an atomic unit. It wouldn't do, for instance, for the program to offer a plane seat to one customer and then offer it to another because the first customer hasn't finished paying yet.

With speculative execution, large atomic tasks introduce two inefficiencies. The first is that, if the task has to abort, it might do so only after chewing up a lot of computational cycles. Aborting smaller tasks wastes less time.

The other is that a large atomic task may have internal subroutines that could be parallelized efficiently. But because the task is isolated on its own processing unit, those subroutines have to be executed serially, squandering opportunities for performance improvements.

Fractal — which Sanchez developed together with MIT graduate students Suvinay Subramanian, Mark Jeffrey, Maleen Abeydeera, Hyun Ryong Lee, and Victor A. Ying, and with Joel Emer, a professor of the practice and senior distinguished research scientist at the chip manufacturer Nvidia — solves both of these problems. The researchers, who are all with MIT's Department of Electrical Engineering and Computer Science, describe the system in "Fractal: An Execution Model for Fine-Grain Nested Speculative Parallelism," presented at the 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture.

With Fractal, a programmer adds a line of code to each subroutine within an atomic task that can be executed in parallel. This will typically increase the length of the serial version of a program by a few percent, whereas an implementation that explicitly synchronizes parallel tasks will often increase it by 300 or 400 percent. Circuits hardwired into the Fractal chip then handle the parallelization.

## Sequential Order

The key to the system is a slight modification of a circuit already found in Swarm, the researchers' earlier speculative-execution system. Swarm was designed to enforce some notion of sequential order in parallel programs. Every task executed in Swarm receives a time stamp, and if two tasks attempt to access the same memory location, the one with the later time stamp is aborted and re-executed.

Fractal, too assigns each atomic task its own time stamp. But if an atomic task has a parallelizable subroutine, the subroutine's time stamp includes that of the task that spawned it. And if the subroutine, in turn, has a parallelizable subroutine, the second subroutine's time stamp includes that of the first, and so on. In this way, the ordering of the subroutines preserves the ordering of the atomic tasks.

As tasks spawn subroutines that spawn subroutines and so on, the concatenated time stamps can become too long for the specialized circuits that store them. In those cases, however, Fractal simply moves the front of the time-stamp train into storage. This means that Fractal is always working only on the lowest-level, finest-grained tasks it has yet identified, avoiding the problem of aborting large, high-level atomic tasks.

*Originally published on MIT News.*
*Reprinted with permission of MIT News.*

---

No entries found

# Comment on this article

Signed comments submitted to this site are moderated and will appear if they are relevant to the topic and not abusive. Your comment will appear with your username if published. View our policy on comments

Notify me via email when subsequent user comments are published with this article.

SUBMIT FOR REVIEW