# RETHINKING CS101

## RUSS MILLER WANTS STUDENTS TO EMBRACE PARALLEL ARCHITECTURES FROM DAY ONE

ISTOCKPHOTO

**Over 350 computer science students** take Russ Miller's Discrete Structures course every fall semester. About 90 percent are freshmen. By week five, they are breaking problems into small chunks and learning ways to solve each chunk at the same time—in parallel.

Today, multicore processors power our laptops and cellphones. Distributed cloud servers or supercomputer clusters process large data sets to improve Facebook news feeds or predict the weather. To take full advantage of these systems, you need parallel algorithms. "It's a parallel world," says Miller, a computer scientist at the State University of New York at Buffalo. "Why is no one teaching a course in parallel algorithms to freshmen?"

Currently, most introductory computer science courses start with sequential programming, in which the computer performs just one instruction at a time. Universities that integrate parallel thinking into their undergraduate curricula tend to offer only an upper-level elective. Others that do spread parallelism throughout the curriculum start no sooner than the second or third course. University needs can vary, but Miller believes that teaching parallel thinking "becomes harder the longer you wait," whereas it can become "second nature" if you do it early enough. So in 2013, Miller changed the State University at Buffalo discrete mathematics course to teach parallel algorithms. It has no prerequisites.

He says that most required discrete mathematics courses teach some material students won't need until their junior or senior year. "That's just the most ridiculous waste of everybody's time," he says. "They just check out." After spending three or four weeks covering the basics of standard logical thinking and divide-and-conquer strategies, Miller dives into parallelism. He gives context by first explaining 1960s- to 2000s-era parallel computing architectures. The rest of the semester he dives into general, hardware-agnostic parallel algorithms for tasks such as searching and sorting. Students learn about such topics as image-segmentation applications.

After each lesson about a new algorithm, students mathematically analyze its theoretical performance on old parallel hardware architectures. They learn some ways to change the algorithm to work on modern, real-world architecture, such as a cloud or grid.

Because it's not about programming per se, Miller can skip thorny implementation details, such as syntax or debugging methods, and have plenty of time to teach students a parallel-first mind set.

"Something like that could work," says Mehran Sahami, a computer scientist at Stanford who cochairs the Association for Computing Machinery steering committee on computing curricula. The ACM and the IEEE jointly introduce new guidelines roughly every 10 years: The latest, issued in 2013, recommend integrating parallel education throughout the curriculum.

But some educators find their ability to deeply embrace parallelism is constrained by other demands. Some instructors of introductory computer science courses, such as Steven Bogaerts at DePauw University in Greencastle, Ind., spend about a week on threads (subsections of a program that can run in parallel) and how to stop them from accessing the same resources at the same time. But to Bogaerts, "it's just already a very full course," so it's hard to do much more.

And some point out that there's more to enhancing code performance than just parallelism. "Of all the ways of getting performance, parallelism is among the hardest," says Charles E. Leiserson, a computer scientist at MIT who teaches a junior- and senior-level course on performance engineering. Leiserson says that parallelizing algorithms doesn't guarantee they'll run faster than sequential algorithms when you implement them on real hardware. He says other factors are important to manage and understand, such as the memory hierarchy or compiler.

Miller, who does not cover the memory hierarchy outside of small examples, is optimistic about his students' futures. He says they have three-and-a-half years to learn how to write efficient code. In his course, they're learning to solve problems at a high level. "I want them to think that the world's open to them," he says. **—ANDREW SILVER**