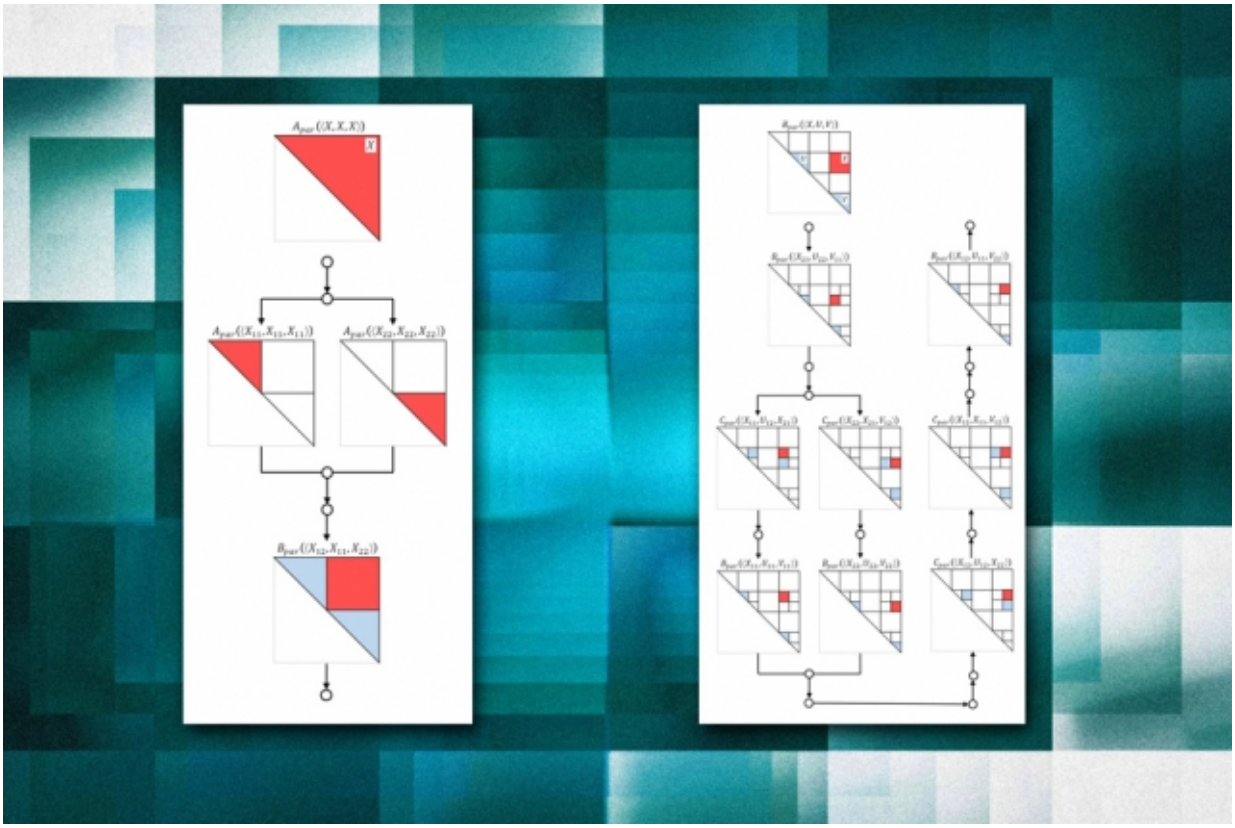
New system lets nonexperts optimize programs that run on multiprocessor chips

November 7 2016, by Larry Hardesty



A system developed by researchers at MIT and Stony Brook University should make it easier for researchers to solve complex computational problems using dynamic programming optimized for multicore chips — without the expertise that such programming typically requires. Credit: MIT News (figures courtesy of the researchers)

Dynamic programming is a technique that can yield relatively efficient solutions to computational problems in economics, genomic analysis, and other fields. But adapting it to computer chips with multiple "cores," or processing units, requires a level of programming expertise that few economists and biologists have.

Researchers from MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) and Stony Brook University aim to change that, with a new system that allows users to describe what they want their programs to do in very general terms. It then automatically produces versions of those programs that are optimized to run on multicore chips. It also guarantees that the new versions will yield exactly the same results that the single-core versions would, albeit much faster.

In experiments, the researchers used the system to "parallelize" several algorithms that used dynamic programming, splitting them up so that they would run on multicore chips. The resulting programs were between three and 11 times as fast as those produced by earlier techniques for automatic parallelization, and they were generally as efficient as those that were hand-parallelized by computer scientists.

The researchers presented their new system last week at the Association for Computing Machinery's conference on Systems, Programming, Languages and Applications: Software for Humanity.

Dynamic programming offers exponential speedups on a certain class of problems because it stores and reuses the results of computations, rather than recomputing them every time they're required.

"But you need more memory, because you store the results of intermediate computations," says Shachar Itzhaky, first author on the new paper and a postdoc in the group of Armando Solar-Lezama, an associate professor of electrical engineering and computer science at

MIT. "When you come to implement it, you realize that you don't get as much speedup as you thought you would, because the memory is slow. When you store and fetch, of course, it's still faster than redoing the computation, but it's not as fast as it could have been."

Outsourcing complexity

Computer scientists avoid this problem by reordering computations so that those requiring a particular stored value are executed in sequence, minimizing the number of times that the value has to be recalled from memory. That's relatively easy to do with a single-core computer, but with multicore computers, when multiple cores are sharing data stored at multiple locations, memory management become much more complex. A hand-optimized, parallel version of a dynamic-programming algorithm is typically 10 times as long as the single-core version, and the individual lines of code are more complex, to boot.

The CSAIL researchers' new system—dubbed Bellmania, after Richard Bellman, the applied mathematician who pioneered dynamic programming—adopts a parallelization strategy called recursive divide-and-conquer. Suppose that the task of a parallel algorithm is to perform a sequence of computations on a grid of numbers, known as a matrix. Its first task might be to divide the grid into four parts, each to be processed separately.

But then it might divide each of those four parts into four parts, and each of those into another four parts, and so on. Because this approach—recursion—involves breaking a problem into smaller subproblems, it naturally lends itself to parallelization.

Joining Itzhaky on the new paper are Solar-Lezama; Charles Leiserson, the Edwin Sibley Webster Professor of Electrical Engineering and Computer Science; Rohit Singh and Kuat Yessenov, who were MIT both

graduate students in electrical engineering and computer science when the work was done; Yongquan Lu, an MIT undergraduate who participated in the project through MIT's Undergraduate Research Opportunities Program; and Rezaul Chowdhury, an assistant professor of computer science at Stony Brook, who was formerly a research affiliate in Leiserson's group.

Leiserson's group specializes in divide-and-conquer parallelization techniques; Solar-Lezama's specializes in program synthesis, or automatically generating code from high-level specifications. With Bellmania, the user simply has to describe the first step of the process—the division of the matrix and the procedures to be applied to the resulting segments. Bellmania then determines how to continue subdividing the problem so as to use memory efficiently.

Rapid search

At each level of recursion—with each successively smaller subdivision of the matrix—a program generated by Bellmania will typically perform some operation on some segment of the matrix and farm the rest out to subroutines, which can be performed in parallel. Each of those subroutines, in turn, will perform some operation on some segment of the data and farm the rest out to further subroutines, and so on.

Bellmania determines how much data should be processed at each level and which subroutines should handle the rest. "The goal is to arrange the memory accesses such that when you read a cell [of the matrix], you do as much computation as you can with it, so that you will not have to read it again later," Itzhaky says.

Finding the optimal division of tasks requires canvassing a wide range of possibilities. Solar-Lezama's group has developed a suite of tools to make that type of search more efficient; even so, Bellmania takes about

15 minutes to parallelize a typical dynamic-programming algorithm. That's still much faster than a human programmer could perform the same task, however. And the result is guaranteed to be correct; hand-optimized code is so complex that it's easy for errors to creep in.

"The work that they're doing is really foundational in enabling a broad set of applications to run on multicore and parallel processors," says David Bader, a professor of computational science and engineering at Georgia Tech. "One challenge has been to enable high-level writing of programs that work on our current multicore processors, and up to now doing that requires heroic, low-level manual coding to get performance. What they provide is a much simpler, high-level technique for some classes of programs that makes it very easy to write the program and have their system automatically figure out how to divide up the work to create codes that are competitive with hand-tuned, low-level coding.

"The types of applications that they would enable range from computational biology, to proteomics, to cybersecurity, to sorting, to scheduling problems of all sorts, to managing network traffic—there are countless examples of real algorithms in the real world for which they now enable much more efficient code," he adds. "It's remarkable."

More information: Deriving Divide-and-Conquer Dynamic Programming Algorithms using Solver-Aided Transformations: people.csail.mit.edu/shachari/dl/oopsla2016.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