Retooling algorithms

February 25 2011, by Larry Hardesty

Charles Leiserson. Photo: Patrick Gillooly

At its most fundamental, computer science is about the search for better algorithms — more efficient ways for computers to do things like sort data or filter noise out of digital signals. But most new algorithms are designed to run on serial computers, which process instructions one after another. Retooling them to run on parallel processors is rarely simple.

As head of MIT's Supertech Research Group, Professor of Computer Science and Engineering Charles Leiserson is an old hand at parallelizing algorithms. Often, he explains, the best approach is to use a technique known as divide-and-conquer. Divide-and-conquer is a recursive technique, meaning that it uses some method to split a problem in half, then uses the same method to split those halves in half, and so on. The advantage of divide-and-conquer, Leiserson explains, is that it enables a computer to tailor an algorithm's execution to the resources available.

Given a computation that requires, say, 10,000 arithmetic operations and a processor with 100 cores, Leiserson says, programmers will frequently

just assign each core 100 operations. But, he says, “let’s say, for example, that one of the processors was interrupted by another job to do something for a moment, so in fact, you had to run on 99 processors. But the software divided it into 100 pieces.” In that case, Leiserson says, “everyone does one chunk; one guy does two chunks. Now you’ve just cut your performance in half.” If the algorithm instead used divide-and-conquer, he says, “that extra chunk that you had could get distributed across all of the other processors, and it would only take 1 percent more time to execute.”

But the general strategy of divide-and-conquer provides no guidance on where to do the dividing or how. That’s something that has to be answered on a case-by-case basis.

In the early 1990s, as a demonstration of the power of parallel algorithms, members of Leiserson’s group designed a chess-playing program that finished second in the 1995 computer chess championship. In a squeaker, the MIT program — the only one of the top finishers not developed by a commercial enterprise — lost a one-game tiebreaker to the eventual winner; the third-place finisher, IBM’s Deep Blue, went on to beat world champion Gary Kasparov two years later.

In large part, a chess program is a method of exploring decision trees. Each tree consists of a move, all the possible responses to that move, all the possible responses to each of those responses, and so on. The obvious way to explore the tree would be to simply evaluate every move and the responses to it to whatever depth time allows. That approach would be easy to parallelize: Each core could just take a different branch of the tree. But some moves are so catastrophically bad that no competent player would ever make them; after a brief investigation, those branches of the tree can simply be lopped off. Parallelizing the pruning of a decision tree is more complicated, since different pathways need to be explored to different depths. The MIT program thus ranked possible

moves in order of likely success and first explored the most promising of them; then it explored the alternative moves in parallel. But it didn't need to explore the alternatives exhaustively, just far enough to determine that they weren't as good as the first move. Exploring the first move, however, meant first evaluating the most promising response to it, and then evaluating the alternative responses in parallel, and so on, down several levels of the tree. Divide and conquer.

The divide-and-conquer strategy means continually splitting up and recombining data, as they're passed between different cores. But this poses its own problems. One common means of storing data, called an array, is very easy to split in two; but combining two arrays requires copying the contents of both into a new array. An alternative is a data storage method called a linked list, in which each piece of data includes a "pointer" that indicates the location of the next piece. Combining linked lists is easy: At the end of one, you just add a pointer to the front of the next. But splitting them is hard: To find the middle of the list, you have to work your way down from the top, following a long sequence of pointers.

So Tao Benjamin Schardl, a graduate student in Leiserson's group, developed a new method of organizing data, which he and Leiserson call a "bag." Though not quite as easy to split up as arrays, bags are much easier to combine; though not quite as easy to combine as linked lists, they're much easier to split up. By using the bag, Schardl and Leiserson developed an algorithm for searching trees that provides "linear speedup," the holy grail of parallelization: That is, doubling the number of cores doubles the efficiency of the algorithm.

The second part of this article will appear tomorrow.

(web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

More information: Computer chips' clocks have stopped getting faster. To maintain the regular doubling of computer power that we now take for granted, chip makers have been giving chips more "cores," or processing units. But how to distribute computations across multiple cores is a hard problem, and this five-part series of articles examines the different levels at which MIT researchers are tackling it, from hardware design up to the development of new programming languages.

Designing the hardware - www.physorg.com/news217669712.html

The next operating system - www.physorg.com/news/2011-02-t … perating-system.html

Provided by Massachusetts Institute of Technology