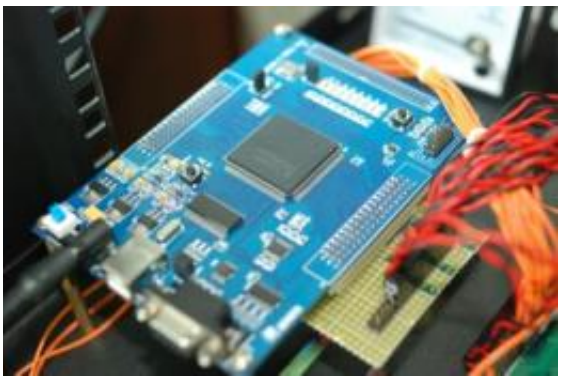
New system makes hardware models of multicore chips more efficient, easier to design, more reliable

April 13 2012, by Larry Hardesty



A photo of an Xilinx field-programmable gate array, or FPGA. Photo: Thomas.L/flickr

Most computer chips today have anywhere from four to 10 separate cores, or processing units, which can work in parallel, increasing the chips' efficiency. But the chips of the future are likely to have hundreds or even thousands of cores.

For chip designers, predicting how these massively multicore chips will behave is no easy task. Software simulations work up to a point, but more accurate simulations typically require hardware models — programmable chips that can be reconfigured to mimic the behavior of multicore chips.

At the IEEE International Symposium on Performance Analysis of Systems and Software earlier this month, researchers from MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) presented a new method for improving the efficiency of hardware simulations of multicore chips. Unlike competing methods, it guarantees that the simulator won't go into "deadlock" — a state in which cores get stuck waiting for each other to relinquish system resources, such as memory. The method should also make it easier for designers to develop simulations and for outside observers to understand what those simulations are intended to do.

Hardware simulations of multicore chips typically use devices called field-programmable gate arrays, or FPGAs. An FPGA is a chip with an array of simple circuits and memory cells that can be hooked together in novel configurations after the chip has left the factory. The chips sold by some small-market manufacturers are, in fact, specially configured FPGAs.

Chip architects using FPGAs to test multicore-chip designs, however, must simulate the complex circuitry found in general-purpose microprocessors. One way to do that is to hook together a lot of the FPGA's simple circuits, but that consumes so many of them so rapidly that the simulator ends up modeling only a small portion of the whole chip design. The other approach is to simulate the complex circuits' behavior in stages — using a partial circuit but spending, say, eight clock cycles on a calculation that, in a real chip, would take only one. Traditionally, however, that's meant slowing down the whole simulation, to allow eight real clock cycles per one simulated cycle.

Go with the flow

For a simulation system they've dubbed Arete, graduate students Asif Khan and Muralidaran Vijayaraghavan; their adviser, Arvind, the

Charles W. and Jennifer C. Johnson Professor of Electrical Engineering and Computer Science; and Silas Boyd-Wickizer, a CSAIL graduate student in the Parallel and Distributed Operating Systems Group, adopted the second approach, but they developed a circuit design that allows the ratio between real clock cycles and simulated cycles to fluctuate as needed. That allows for faster simulations and more economical use of the FPGA's circuitry.

Every logic circuit has some number of input wires and some number of output wires, and the CSAIL researchers associate a little bit of memory with each such wire. Data coming in on a wire is stored in memory until all the operations that require it have been performed; data going out on a wire is stored in memory until the data going out on the other wires has been computed, too. Once all the outputs have been determined, the input data is erased, signaling the completion of one simulated clock cycle. Depending on the complexity of the calculation the circuit was performing, the simulated clock cycle could correspond to one real clock cycle, or eight, or something in between.

The memory associated with the input and output wires and the logic to control it does take up some extra real estate on the FPGA, but "you only do it for select parts of the design, like complex logic, or these memories that you just cannot fit on chip on FPGAs," Khan says. "And remember, through this overhead, you are able to save a lot of resource" — that is, FPGA circuitry. Moreover, it's the memory system that allows the researchers to guarantee that their simulator won't deadlock. Other research groups have developed FPGA multicore-simulation systems that use space and time efficiently, but they don't offer that guarantee.

The big picture

Another advantage of their system, the CSAIL researchers argue, is that it makes it easier for outside observers — and even for chip designers

themselves — to understand what a simulation is intended to do. With other researchers' simulators, it's often the case that "the cycle-level specification for the machine that they're modeling is in their heads," Khan says. "What we're proposing is, instead of having this in your head, let's start with a specification. Let's write it down formally but in a language that is at a very high level of abstraction, so it does not require you to write a lot of details. And once you have this specification that clearly tells you how the entire multicore model is going to behave every cycle, you can transform this automatically into an efficient mapping on the FPGA."

The researchers' high-level language, which they dubbed StructuralSpec, builds on the BlueSpec hardware design language that Arvind's group helped develop in the late 1990s and early 2000s. The StructuralSpec user gives a high-level specification of a multicore model, and software spits out the code that implements that model on an FPGA. Where a typical, hand-coded hardware model might have about 30,000 lines of code, Khan says, a similar model implemented on StructuralSpec might have only 8,000 lines of code.

Kattamuri Ekanadham, a chip researcher at IBM's T. J. Watson Laboratory, is currently building his own implementation of the MIT researchers' simulator. He hasn't used it yet, so he can't characterize its performance, but he says that it does have several features that convinced him to give it a try.

Other FPGA simulators "do the functional evaluation of the various circuits and functions on one machine, and then they will do a software model of the time that each of these circuits takes on a different machine," Kattamuri says. "The problem I find with that approach is, first of all, the clean separation of time versus functionality is difficult, and I don't know how to verify that it is accurately done. And then of course I have to run two systems, and I have to have proper interaction

between the timing simulator and the functional simulator."

With the MIT simulator, on the other hand, "everything is integrated," Kattamuri says. "I don't take care of the timer at all. Everything is taken care of automatically by the subsystem. So that's why I find this more attractive."

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