Reconfigurable Accelerators for Combinatorial Problems

Tailoring hardware to a specific algorithm *and* a specific set of input data can boost execution several fold. One hardware circuit that solves *Boolean satisfiability* improved execution time by a factor of 140,000 over state-of-the-art software solvers.

Marco Platzner
Swiss Federal Institute of Technology (ETH) Zurich

Reconfigurable accelerators can improve process time on combinatorial problems with fine-grained parallelism. Such problems contain a huge number of logical operations (NOT, AND, and OR) that can evaluate simultaneously, a characteristic that varies considerably from problem to problem. Because of this variability, such combinatorial problems are approached using *instance-specific reconfiguration*—hardware tailored to a specific algorithm *and* a specific set of input data.

Boolean satisfiability (see the sidebar) is a common combinatorial problem that exhibits fine-grained parallelism that varies considerably based on the situation. Its solution is thus an ideal candidate for improvement with instance-specific reconfiguration. In fact, simulations of an instance-specific accelerator show potential speedups by a factor of up to 140,000 in execution time over the solution by a software solver. We worked on a prototype in current FPGA technology that leads to an order-of-magnitude speedup in the execution of difficult satisfiability problems. A conference and workshop published the details of that work.[1,2]

IMPLEMENTING SAT IN HARDWARE

The basic architecture for solving Boolean satisfiability (commonly known as "SAT" for short) in hardware, shown in Figure 1, consists of three building blocks: a chain of finite state machines (FSMs), deduction logic, and a global controller. The FSMs are connected in a one-dimensional array—each FSM can activate its two neighboring FSMs, one above and one below. The deduction logic is a combinational circuit that computes the three-valued logic result of the Boolean formula. The global controller starts computation and handles I/O. The architecture is instance-specific because the SAT problem instance we want to solve determines the number of FSMs and the deduction logic.

The hardware architecture follows a simple assign-and-determine procedure. The architecture maps SAT problem variables to three possible values: 0, 1, or X (X denotes an unassigned variable). Each assigned variable corresponds to an FSM holding the variable's value. Initially, all variables are unassigned, and the global controller activates the topmost FSM. An activated FSM assigns 0 to its variable and checks the formula's result.

If the formula evaluates to 1, this partial assignment satisfies the formula and computation stops. An evaluation to 0 contradicts the formula, and the FSM tries the complementary value for its variable. If the formula evaluates to X, the FSM activates its neighbor FSM below. If the FSM tries both value assignments (0 and 1) without success (without evaluating the formula to 1), it relaxes this variable by assigning X, and activates the FSM above.

The act of the first FSM in relaxing its variable and activating the global controller proves the SAT problem to be unsatisfiable. This procedure implements backtracking search with a fixed ordering of variables, a rather simple deduction step that checks each partial assignment for satisfaction or contradiction, and does not perform a diagnosis.

By exploiting fine-grained parallelism in the deduction logic, this process has a huge advantage over software solvers. The deduction logic consists of three-valued logic operators (NOT, AND, and OR), a good match for the fine-

0018-9162/00/$10.00 © 2000 IEEE

grained hardware structures of field-programmable gate arrays. The deduction logic computes the Boolean formula in a single clock cycle. Even newer microprocessors, with clock frequencies 10 to 20 times higher than FPGA-based SAT accelerators, cannot compete, except for trivially small SAT problems. On the other hand, software solvers have more powerful strategies for decision, deduction, and diagnosis that require fewer value assignments to find a solution.

RESULTS

Simulations of the basic SAT architecture with problems from the DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) benchmarks suite showed raw speedups between 10 and 140,000 times the performance of state-of-the-art software SAT solvers. These results reveal the great potential of instance-specific acceleration.

To obtain realistic performance data, including hardware compilation and configuration times, we created an accelerator prototype. This prototype reads a SAT problem and generates an instance-specific circuit, which it compiles into a configuration bitstream for an FPGA. The prototype then loads the bitstream onto the FPGA, starts the computation, and if the FPGA finds a solution, the prototype reads the variable values back from the FPGA. We implemented this prototype on a low-cost system: a Windows-NT-based PC with a Digital/Compaq PCI Pamette board, which contains four Xilinx XC4028 FPGAs.

This prototype speeds up runtime by an order of magnitude for medium-sized SAT problems. As an example, Figure 2 shows the runtimes and the overall speedups for instances of the pigeon hole problem—a well-known benchmark that asks if it is possible to place $n+1$ pigeons in n holes without two pigeons being in the same hole—from the DIMACS benchmarks suite. For small problem sizes (hole6, hole7, and hole8), hardware compilation times dominated the accelerator's runtime. For hole9, we observed a real overall speedup for the first time, measuring hardware compilation and execution time and comparing it to the runtime of a software solver. For hole10, using the reconfigurable accelerator reduced the software runtime from 2 hours and 7 minutes to 17 minutes. Larger instances of this problem require FPGAs with more logic resources than Xilinx XC4028s offer.

We have also implemented extensions to our basic architecture using more powerful deduction strategies based on don't-care variables and Boolean constraint propagation (a deduction mechanism that exploits the fact that a partial assignment can imply values for other variables). Although these variants can use fewer clock cycles to find solutions, they also require more hardware, resulting in slower FPGA designs.

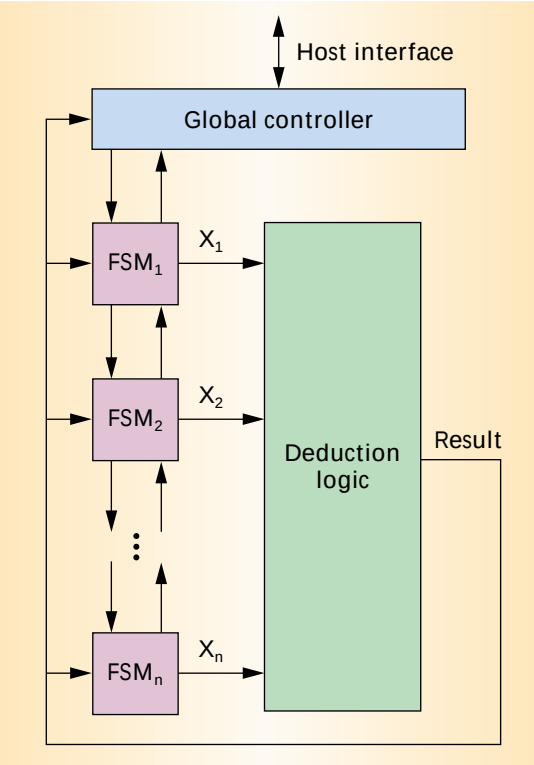

Figure 1. Basic hardware architecture for solving SAT problems with backtracking search.

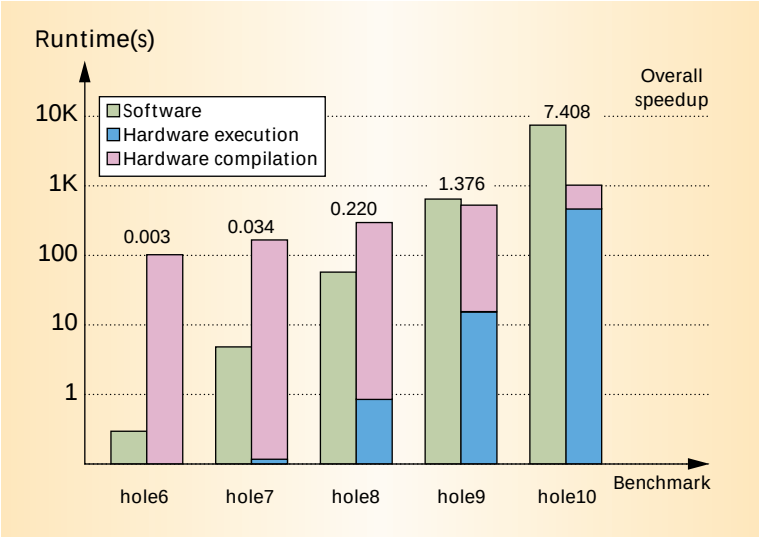

Figure 2. Runtimes and overall speedups for hard SAT problems, measured for the basic SAT architecture running at 20 MHz.

FPGAs will continue to become denser and faster, which will benefit instance-specific accelerators. Additionally, reduced FPGA synthesis and compilation times will enable the promising high raw speedups for algorithms like those for SAT. Using today's FPGAs restricts instance-specific accelerators to hard combinatorial problems with long software runtimes. Our experiments further revealed that a reconfigurable SAT accelerator does not accurately produce speedups over different benchmark classes. Moreover, many SAT problems have very specific properties for which highly optimized software solvers exist. A practical acceleration engine will have to combine an instance-specific accelerator with software SAT solvers. If the software solvers do not ter-

Boolean Satisfiability (SAT)

SAT is a fundamental problem in mathematical logic and computing theory and has many practical applications in automated reasoning, computer-aided design and manufacturing, databases, robotics, machine vision, computer architecture, and networks.[1] For example, engineers use it for the automated generation of test patterns to test digital circuits for faults.

The SAT problem is to find an assignment of truth values (0,1) to a set of Boolean variables, $x_1, ..., x_n$, so that a given Boolean formula evaluates to 1. SAT is NP-complete, which means the worst case runtime of an exact SAT solver grows exponentially with the number of variables.

Most exact SAT solvers rely on backtracking and perform a depth-first search of the problem's search tree. Figure A shows the search tree for the following formula:

$$(\bar{x}_1 \text{ OR } x_2) \text{ AND } (x_1 \text{ OR } x_3) \text{ AND } (x_1 \text{ OR } \bar{x}_2)$$

Starting with all variables unassigned, the procedure iteratively picks a free variable and assigns it a truth value. When the procedure assigns all variables (steps a, b, and c in Figure A), it checks the result of the formula. If the formula evaluates to 1, satisfiability is proven. Otherwise, the procedure goes back until it can assign a variable a value that it hasn't tried earlier (step d). The procedure continues, systematically alternating forward search and backtracking until it finds a satisfying assignment (step e) or scans the whole search tree.

Such a procedure would be very time-consuming, however. Practical SAT solvers use improved strategies for decision, deduction, and diagnosis.[2] The decision step selects a variable for the next assignment, either statically with a fixed variable order or dynamically, depending on information gathered during search. The deduction step infers information from the current partial assignment.

For example, in the sample formula, assigning $x_1 \leftarrow 0$ and $x_2 \leftarrow 1$ already contradicts the premise (the formula cannot evaluate to 1); there is no point in assigning x_3. Such deduced information contributes greatly to a SAT solver's efficiency and sometimes prunes off large parts of the search tree.

A widely used deduction mechanism is *Boolean constraint propagation*. Introduced first in the Davis-Putnam algorithm,[3] this technique exploits the fact that a partial assignment can imply values for other variables. A diagnosis step analyzes the contradiction's cause and uses the inferred knowledge to search more efficiently.

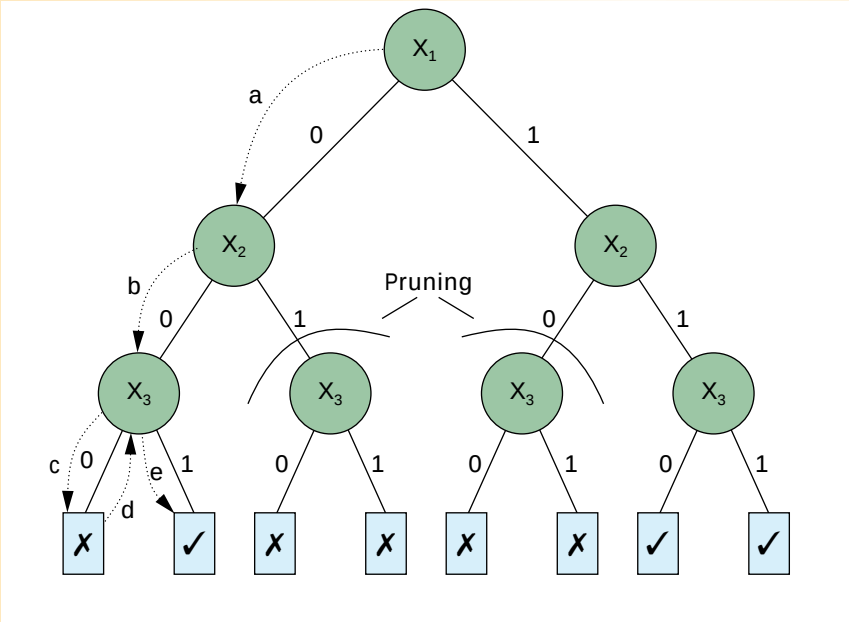

Figure A. Search tree for the sample formula.

References

1. J. Gu et al., "Algorithms for the Satisfiability (SAT) Problem: A Survey," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 35, 1997, pp. 19-151.
2. J.P.M. Silva and K.A. Sakallah, "GRASP: A New Search Algorithm for Satisfiability," *Proc. Int'l Conf. CAD*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 220-227.
3. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, July 1960, pp. 201-215.

minate within a given period, the software can migrate the problem to reconfigurable hardware. We envision future reconfigurable accelerators for combinatorial problems as plug-ins to workstations. These will employ a handful of high-density FPGAs and have a software shell that makes their technology invisible to the user. ❖

References

1. M. Platzner and G. De Micheli, "Acceleration of Satisfiability Algorithms by Reconfigurable Hardware," *Int'l Workshop Field-Programmable Logic and Applications*, Springer-Verlag, Berlin, 1998, pp. 69-78.
2. O. Mencer and M. Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment," *32th Hawaiian Int'l Conf. System Sciences*, IEEE CS Press, Los Alamitos, Calif., 1999.

Marco Platzner is a senior researcher in the Computer Engineering and Networks Lab at the Swiss Federal Institute of Technology Zurich, Switzerland. His research interests include reconfigurable computing, hardware-software codesign, and embedded systems. Platzner earned a PhD in telematics from Graz University of Technology, Austria. He is a member of the ACM and the IEEE. Contact him at platzner@tik.ee.ethz.ch or marco.platzner@computer.org.