Data Parallel Switch-Level Simulation*

Randal E. Bryant
Computer Science Department
Carnegie Mellon University

Abstract

Data parallel simulation involves simulating the behavior of a circuit over a number of test sequences simultaneously. Compared to other parallel simulation techniques, data parallel simulation requires less overhead for synchronization and communication, and it permits higher degrees of parallelism.

We have implemented two data parallel versions of the switch-level simulator COSMOS. The first runs on conventional machines, exploiting the bit parallelism of machine-level logic operations. This version runs 20–30 times faster than sequential simulation on the same machine. The second runs on a a massively-parallel SIMD machine, with each processor simulating the circuit behavior for a single test sequence. The simulator running on a 32,768 processor machine runs up to 33,000 times faster than sequential simulation on a workstation computer.

1 Introduction

Most attempts to exploit parallelism in simulation utilize *circuit parallelism*. In this mode, the simulator extracts parallelism from the circuit evaluation while modeling the behavior for a single test sequence. Such an approach has the advantage that it accelerates the existing way in which simulators are used. However, its performance is limited by the need to control the simulation, as well as by the degree of parallelism found within the circuit. The overhead of keeping the simulation synchronized and communicating values between processing units can overwhelm the savings gained through parallel evaluation. Moreover, there is an upper limit on how much parallelism can be extracted from digital circuit simulation. Various studies [1, 8] indicate very low average parallelism ($\sim$ 10-fold) for event-driven simulation kept in tight synchronization. More clever synchronization techniques and larger circuits might permit higher degrees of parallelism, but it seems unlikely that truly "massive" (e.g., 10,000-fold) parallelism can be achieved.

As an alternative, we have explored a different form of parallel simulation, in which the simulator models the behavior of the circuit over a number of test sequences simultaneously. By exploiting this *data parallelism*, high degrees of parallelism can be achieved even though the behavior of the circuit for a given test sequence is performed sequentially. This approach imposes special requirements on the simulation algorithm, the target machine, and the simulator input patterns. Of these requirements, the first two can be satisfied by the implementor. The final one, however, has a fundamental impact on how the simulator is used. The simulation patterns must be formulated as a large number of short test sequences that can be generated and checked algorithmically.

We have implemented two data-parallel versions of the switch-level simulator COSMOS [4]. The first operates on conventional machines, exploiting the bit-level parallelism of the machine operations to simulate 32 test cases simultaneously. The second implementation operates on a Connection Machine [6], with each processing element simulating the behavior of the circuit for a single test sequence.

This paper describes the general requirements for exploiting data parallel simulation, as well as the method by which we have met these requirements with COSMOS. Experimental results are given to indicate the potential performance.

2 Related Work

The concept of data parallel simulation is not new. Most implementations exploit the bit-level parallelism inherent in conventional machine operations.

*This research was supported in part by the Defense Advanced Research Projects Agency, ARPA Order Number 4976, and in part by the Semiconductor Research Corporation under contract 88-DC-068.

The HSS program [2] developed at IBM simulates a single good or faulty circuit on up to 32 patterns simultaneously. This contrasts with more traditional "parallel fault simulation" in which bit-level parallelism is exploited to simulate a set (e.g., 1 good and 31 faulty) of circuits over a single test sequence. Although the original version of HSS performed only 2-valued, rank-ordered modeling of combinational circuits, the authors mention such extensions as 3-valued modeling, event scheduling, and support for sequential circuits.

The FSS program developed at GEC Research [7] is designed to compute the expected signatures for circuits containing linear feedback shift register signature analyzers. By exploiting the superposition principle of linear systems, this program can divide the test sequence into 32 sections, simulate the sections simultaneously, and then combine the results to determine the signature that would result if the sections were simulated in sequence. Unfortunately, this technique does not extend to general circuits.

The research described here is unique in several respects. It is the first to exploit data parallelism while providing a detailed, switch-level model of the circuit. It is the first reported mapping of data parallel logic simulation onto a massively-parallel processor. Finally, our work on simulation methodology is the first to address the issue of generating patterns for general sequential circuits suitable for data parallel simulation.

3 Simulation Algorithm

For a single control sequence to direct a number of simultaneous simulations, the simulation algorithm must be "oblivious", or at least nearly so. With oblivious computation, the flow of control does not depend on the data being evaluated. Many techniques traditionally used by logic simulators are not oblivious, including:

- *Event scheduling.* The set of logic elements evaluated at each time point depends on the simulated data.
- *Table lookup.* The memory locations referenced depend on the simulated data.
- *Data dependent delays.* The time points at which elements are evaluated depend on the simulated data.
- *Functional models.* Most modeling languages include data-dependent control operations, such as conditional and iteration statements.

Although the above techniques lead to nonoblivious behavior, some can still be utilized in conjunction with data parallel simulation. For example, the simulator can utilize event scheduling by following the convention that a logic element must be scheduled if one of its inputs changes for any of the cases being simulated. This requires the simulator to schedule more events than it would when simulating just a single case. The effectiveness of event-scheduling therefore decreases as the number of parallel cases increases. Our measurements indicate, however, that event scheduling can yield a savings even when as many as 1 million cases are simulated in parallel.

COSMOS utilizes a unique approach to switch-level simulation that makes it suitable for data parallelism. Most switch-level simulators maintain data structures representing the transistor network and apply (nonoblivious) graph algorithms to evaluate the circuit behavior [3]. COSMOS, on the other hand, preprocesses the transistor network to produce a Boolean representation. This representation, produced by the symbolic analyzer ANAMOS, captures such aspects of switch-level networks as: bidirectional transistors, stored charge, charge sharing, different signal strengths, and indeterminate (X) logic values. The 3-valued behavior of the circuit is represented by encoding each node state as a pair of Boolean values. The symbolic representation of the circuit describes how to compute Boolean values encoding the new states of the circuit nodes as a function of the Boolean values encoding the current node states. Hence, only simple 2-valued logic operations are required to compute the circuit behavior.

The methods used by ANAMOS are indicative of the ways in which a complex simulation model can be implemented in a manner suitable for data parallel simulation. Signal encoding allows the circuit behavior to be expressed in terms of Boolean operations. The symbolic analysis captures the subtle details of the switch-level model. This eliminates the need to model strength effects by encoding strengths in the signal algebra or by introducing data-dependent control flow. Similar techniques could be applied to other simulation models to allow data parallel implementations.

4 Implementation Details

COSMOS utilizes a hybrid of compiled-code and event-driven simulation. The preprocessor generates and compiles an update procedure for each DC-connected subnetwork of the circuit based on its Boolean description. The simulator schedules events

at the subnetwork level with a mixed zero/unit delay timing model. On each unit step, the simulator sequences through the subnetworks on the current event list, invoking the subnetwork procedure to compute new states for the nodes and determining which nodes change state. For each node that changes state, it puts the fanout subnetworks onto the new event list and sets the node state to the newly computed value.

Within the innermost loops of the simulation code, each operation either manipulates data or implements the control flow. The primary data operations are: *logic operations* to compute the new states of the nodes, *equality testing* to determine which nodes have changed state, and *data movement* to update the states of the nodes. These are precisely the operations that can exploit data parallelism. The control operations maintain the event lists and trace the fanout connections. These operations must still be performed sequentially, but their cost is amortized over the number of cases being simulated in parallel.

For execution on a conventional machine, the machine performs both the data manipulation and the control flow operations. Two pairs of 32-bit words encode the current and new state of each node for the 32 cases being simulated in parallel. The machine's logic, copy, and comparison operations directly provide the data parallelism. In fact, simulating a circuit sequentially simply wastes 97% of the machine's data processing power.

For execution on a Connection Machine, the parallel processors perform the data manipulation while the host performs the control flow operations. Each processor holds the state of the circuit for one of the test cases being simulated. Two pairs of bits on each processor encode the current and new state of each node. The simulation proceeds much as in the conventional implementation, except for the data manipulations. To perform a logic or data movement operation, the host commands all of the processors to perform the operation on their own copies of the circuit state. To test whether a node has changed state, the host commands each processor to set a flag if its values for the current and new node state differ and then OR's the flag values together.

Three reasons can be found for the high efficiency of our mapping of COSMOS onto the Connection Machine. First, the bit-serial operations of Connection Machine correspond closely to what is required to evaluate the Boolean representation of a circuit. Second, the SIMD parallelism of the Connection Machine closely matches our division between sequential control flow and parallel data manipulation. Finally, all data movement occurs either within the individual processors or between the host and one or more processors. Hence, we avoid using the relatively slow interprocessor communication network altogether.

5 Methodology

Except for the special case of linear signature analyzers, data parallel simulation requires the simulation patterns to be formulated as a number of independent test sequences. For combinational circuits, this restriction imposes no difficulty, because the circuit outputs do not depend on past inputs anyhow. For sequential circuits, on the other hand, this restriction is more severe.

We have developed a simulation methodology for which the resulting patterns naturally partition into a number of independent tests. This methodology arose from using logic simulation to formally verify sequential circuits [5]. In this methodology, each simulation sequence checks a class of state transitions by the underlying sequential system. Each sequence focuses on a small portion of the circuit, showing that for a specific set of values on a small number of input and internal nodes, correct values will result on a small number of output and internal nodes. During simulation, those nodes that should not affect an operation remain set to the unknown value X. If the simulation produces a 0 or 1 on some output or state node, then we can be assured that this value would result for any assignment of 0's and 1's to the nodes set to X.

As an extreme case, we have shown that an N-bit static RAM can be fully verified by evaluating $O(N \log N)$ independent test cases, where each test case involves simulating one cycle of circuit operation. For example, verifying the memory read operation involves evaluating 2 test cases per memory location. Evaluating a test case for memory location i and bit value b involves simulating the circuit with the two nodes in memory cell i initialized to b and $\overline{b}$, with the address input set to the bit representation of i, and with the read/write control line set for a read operation. All other nodes are initialized to X. At the end of the cycle, the verifier checks that the two nodes in memory cell i still equal b and $\overline{b}$ (non-destructive read) and that the data output equals b. Because all other memory locations are initialized to X, this single test effectively checks the circuit for 2^{N-1} different initial memory states. The remaining test cases verify that a write operation causes the input data be stored into the appropriate memory cell and that any operation on one memory location does not affect the value stored in another location.

Many of the ideas from this methodology can be

applied to informal verification as well. That is, simulation tests can be designed with the following characteristics:

- They focus on the operation of only small regions of the circuit.
- They assume a minimal initialization of the circuit. There is no attempt to force all nodes out of X.
- They directly manipulate and examine the states of internal nodes, rather than just observing the input–output behavior of the circuit.
- They simulate only short sequences of circuit operation.
- They can be generated and checked algorithmically.

The resulting patterns provide a more rigorous test of the design, because they are more likely to detect unexpected sequential dependencies. Furthermore, they can be used by a data parallel simulator.

6 Test Generation and Analysis

When using a data parallel simulator, it becomes very tedious to describe the input data and evaluate the simulation outputs by hand. Instead, we have found it preferable to write procedures to generate test patterns and to check that the resulting outputs match the expected values. COSMOS includes a set of library routines to assist in this programming.

For execution on a conventional machine, the routines allow the user to generate test patterns by looping through an indexed set of test cases. Within the loop, the code repeatedly packs together 32 sets of inputs, simulates the block, fetches and unpacks the results, and then checks the results. Typically, information is printed on the terminal only if an incorrect result occurs, or after all tests have been applied.

For execution on a Connection Machine, the input patterns should be generated and the results checked using the parallelism of the processing elements. Otherwise the time spent downloading patterns and uploading results can dominate the simulation time. Fortunately, the general-purpose nature of the Connection Machine makes it possible to do all of the input preparation and result analysis on the machine. For example, the input stimuli can be created in parallel by generating random bit patterns, or by generating patterns based on the processor number.

Machine	Parallel	Time	Rel. Speed
MicroVax-II	1	81 da.*	1
MicroVax-II	32	103 hr.*	19
CM-2	32,678	212 sec.	33,000
10 Mhz. Circuit	1	3.2 sec.	2,200,000

* – Estimated by extrapolation

Table 1: Times to evaluate 32M patterns for 16-bit nMOS ALU

The results can be checked by computing the desired responses based on the input patterns and comparing these to the circuit outputs.

7 Experimental Results

Table 1 indicates the time required to evaluate a 688 transistor, 16-bit nMOS ALU set to perform addition for 33,554,432 (32M) different input data patterns. In each case, the resulting output was checked to make sure it equaled the sum of the inputs. This circuit utilizes a precharged, Manchester carry chain, but functionally behaves as a combinational circuit.

Two machines were used for this experiment: a Digital Equip. Corp. MicroVax-II, and a 32K-processor Thinking Machines Corp. CM-2 with a Digital Equp. Corp. VAX/8800 as host. The times marked with an asterisk (*) were estimated by simulating the circuit for a smaller number of input patterns and extrapolating. They should be accurate to within 10%. Even in sequential mode, COSMOS outperforms its predecessor, MOSSIM II, by over a factor of 10. MOSSIM II would require an estimated 2.5 years on a MicroVax-II to perform the same simulation! Simulating 32 cases in parallel yields another factor of 19 in speed. This speed up falls short of the ideal value of 32 due to the cost of packing and unpacking the blocks of patterns, and due to the larger number of events that must be simulated.

The circuit was simulated on the CM-2 configured as 32 virtual processors per physical processor. The total simulation run involved simulating 32 batches of 1M test cases each. As can be seen, the resulting speed up approaches the maximum realizable, considering that the cycle time of a CM-2 is only slightly less than that of a MicroVax-II. Our measurements show that simulating 1M inputs in parallel only requires 2.1 times more events than does simulating a single input. Thus, the overhead incurred by data parallel simulation is quite small. Observe that the effective speed of the simulation is within 2 orders of magnitude of the actual circuit speed, assuming the

Machine	Parallel	Time	Rel. Speed
MicroVax-II	1	47.9 hr.	1
MicroVax-II	32	93 min.	31
CM-2	24,577	1 min.	2,900
10 MHz. Circuit	1	0.0025 sec.	7×10^7

Table 2: Times to Evaluate 24,577 patterns for 1K CMOS SRAM

circuit operates with a 10 MHz clock.

Table 2 indicates the times required to create, simulate, and analyze a set of 24,577 patterns giving a rigorous verification of a 1K-bit CMOS static RAM. In this case, the parallel implementation on a conventional machine yields a nearly ideal speed up of 31. The regions of activity in this circuit are less sensitive to the data being simulated, and the time required to pack and unpack the data is small compared to the time required for simulation.

Mapping this simulation onto a 32K processor Connection Machine yields another factor of 90 in speed. Although this is a significant speed up, it is not as large as for the ALU benchmark. Several reasons can be found for this performance. First, there were not enough patterns to utilize the full capacity of the machine. Using virtual processors, the machine has enough memory to simulate 128K patterns in parallel for this circuit, but less than 1/5 this number is required for full verification. Second, the program to set up the patterns and to analyze the results can only partially exploit the parallelism of the Connection Machine. As a result, less than 50% of the CPU time was spent running the actual simulation.

8 Conclusions

The data presented clearly indicate the performance potential of data parallel simulation. Without changing hardware or modeling level, the simulator operates 20–30 times faster than before. Moving to a general purpose parallel processor yields another 2–3 orders of magnitude improvement. This degree of improvement makes possible simulations that would otherwise be totally impractical.

References

[1] M. L. Bailey, L. Snyder, "An Empirical Study of On-Chip Parallelism," *25th Design Automation Conference*, 1988.

[2] Z. Barzilai, *et al*, "HSS—A High Speed Simulator", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-6, No. 4 (July, 1987), pp. 601–617.

[3] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. on Computers*, Vol. C-33, No. 2 (February, 1984), pp. 160–177.

[4] R. E. Bryant, *et al*, "COSMOS: a Compiled Simulator for MOS Circuits," *24th Design Automation Conference*, 1987, pp. 9–16.

[5] R. E. Bryant, "Verifying a Static RAM Design by Logic Simulation," *Fifth MIT Conference on Advanced Research in VLSI*, 1988, pp. 335–349.

[6] W. D. Hillis, *The Connection Machine*, MIT Press, 1985.

[7] S. B. Tan, *et al*, "A Fast Signature Simulation Tool for Built-In Self-Testing Circuits", *24th Design Automation Conference*, 1987, pp. 17–25.

[8] K. F. Wong, *et al*, "Statistics on Logic Simulation", *23rd Design Automation Conference*, 1986, pp. 13–19.