

Parallel Logic Simulation of Million-Gate VLSI Circuits

Lijuan Zhu, Gilbert Chen, and Boleslaw K. Szymanski
Rensselaer Polytechnic Institute, Computer Science Department
zhul4,cheng3,szymansk@cs.rpi.edu

Carl Tropper
McGill University, School of Computer Science
carl@cs.mcgill.ca

Tong Zhang
Rensselaer Polytechnic Institute, ECSE
tzhang@ecse.rpi.edu

Abstract

The complexity of today's VLSI chip designs makes verification a necessary step before fabrication. As a result, gate-level logic simulation has become an integral component of the VLSI circuit design process which verifies the design and analyzes its behavior. Since the designs constantly grow in size and complexity, there is a need for ever more efficient simulations to keep the gate-level logic verification time acceptably small.

The focus of this paper is an efficient simulation of large chip designs. We present the design and implementation of a new parallel simulator, called DSIM, and demonstrate DSIM's efficiency and speed by simulating a million gate circuit using different numbers of processors.

1. Introduction and background

The development process of a hardware unit may take several months or even years, and the costs of instrumentation needed for its fabrication may reach several billions of dollars. Therefore circuit simulations done before fabrication have become an important and necessary step to avoid design errors.

Hardware designs are supported by hardware description languages, or HDLs, such as VHDL (Very High Speed IC Hardware Description Languages) [10] and Verilog [2]. By using a HDL, one can describe arbitrary digital hardware at any level. Chips are designed either in bottom-up or top-down fashion. The preferred style of most Verilog based designs is top-down. HDLs support different ways of describing a chip. Verilog, for example, provides three levels of abstraction: behavioral level, register-transfer level and gate level. Accordingly, circuit simulation can be classified into four groups [13]: behavioral or functional simulation in which circuit elements are modeled as functional

blocks, gate-level logic simulation in which a circuit is modeled as a collection of logic gates, switch-level simulation in which circuit elements are modeled as transistors, and circuit level simulation in which resistors and wires with propagation delays are also represented.

Gate-level simulations can be classified into two categories: oblivious and event-driven [13]. In the former, every gate is evaluated once at each simulation cycle, while in the latter, a gate is evaluated only when any of its inputs has changed. For large circuits, event-driven simulation is more efficient because fewer logic gates are evaluated at any instant of the simulated time. Still, very large and complex systems take substantial amounts of time to simulate, even utilizing event-driven simulators [5]. Consequently, parallel simulation has become a necessity for such circuits.

1.1. A Viterbi decoder design

As our benchmark, we selected the circuits implementing a state-parallel RE Viterbi decoder whose block diagram is shown in Figure 1(a). The decoder contains three functional blocks: branch metric unit (BMU) which calculates all the branch metrics; add-compare-select (ACS) units which update the accumulative survivor path metrics; and a survivor memory unit (SMU) which stores the survivor paths and generate the decoder output. For a trellis with N states, a state-parallel decoder implements N ACS units that operate in parallel.

As extensively discussed in the literature (e.g., [6, 7]), the SMU is conventionally designed in two different styles, register exchange (RE) and trace back (TB), targeting different power/complexity versus throughput trade-offs. Basically speaking, RE can easily support very high decoding throughput (e.g., hundreds Mbps or even several Gbps) but requires a large number of transistors and consumes a lot of power. TB decreases implementation complexity and is

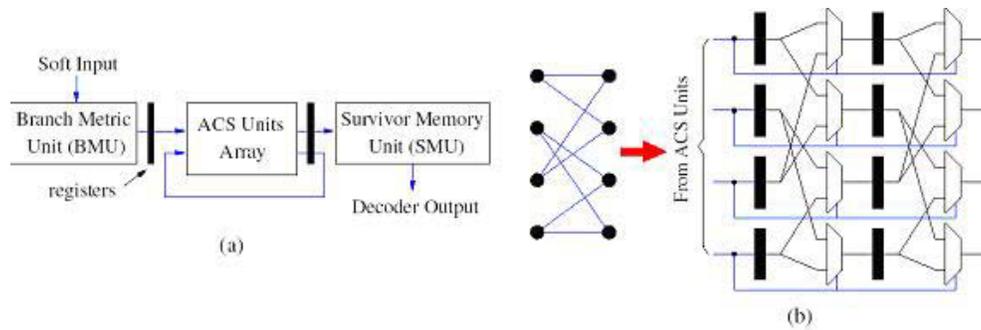


Figure 1. (a) The block diagram of a state-parallel Viterbi decoder, and (b) a sample four-state register exchange structure for SMU

quite power-efficient but cannot support very high decoding throughput. In the RE Viterbi decoder, as illustrated in Figure 1, the decoder output is obtained by a simple register shift operation and the critical path typically lies in ACS recursion. On the other hand, in the TB Viterbi decoder, certain number of memory accesses are required to obtain each decoder output, which often results in the trace back being the critical path. One important parameter in both RE and TB Viterbi decoders is the decoding decision depth, which is the length of the memory path. For convolutional codes, the decision depth selection has been well discussed [3]. We simulate an RE Viterbi decoder with a constraint length of 11, corresponding to 1.2 million NAND gates.

1.2. Parallel Discrete Event-driven Simulation

To speed up simulations, parallel discrete event simulation (PDES) for large circuits has been advocated and used. In this approach, the model is composed of some disjoint submodels [16], each represented by the so-called Logical Process (LP). Each processor takes charge of simulating one or more LPs, and each LP interacts with all of the LPs which are influenced by the given LP's local changes. These interactions are enforced by messages carrying events between LPs. Each message is timestamped with the simulation time at which its event should execute. In parallel simulation of circuits, each gate is modeled as an LP. A gate propagates its output signals to the connected gates. If the LP of a connected gate resides on a different processor, the output generates a messages sent to that processor. Each processor maintains a future event queue from which events are selected for execution. To achieve a correct simulation, the order of the final execution of events must preserve causality in the simulated process. This is the main challenge in parallel simulation. Two major techniques were developed to address this challenge: conservative and optimistic [9]. We focus on optimistic protocols in

this paper.

Many researchers have developed parallel simulators to speed up logic simulation. Meister gave a good albeit bit dated review of parallel logic simulation in [16].

Lungeanu and Shi [17] developed a parallel simulator for VHDL designs, using both a conservative and an optimistic protocols. Williams [18] developed Icarus Verilog, an open-source Electronic Design Automation (EDA) sequential Verilog simulator. In [14, 15], Li *et al* designed and implemented DVS, an objected-oriented framework for distributed Verilog simulation. The distributed simulation engine is based on the Clustered Time Warp (CTW) [4].

All these parallel logic simulators simulated circuits of quite modest size of about several thousands gates. The simulator described in this paper has been developed with the explicit goal of simulating large circuits, having millions of gates.

The rest of the paper is organized as follows. Section 2 contains a description of our circuit simulator. Section 3 provides performance results for our simulations of the Viterbi decoder. The final section contains our concluding remarks and plans for future work.

2. Verilog Simulation

The simulator which we have designed and implemented consists of a translator, a parser and a simulator proper.

To enhance the design's modularity and encapsulation, Verilog defines modules in a hierarchical structure. However, this structure is difficult to process by a simulator. The goal of the translator is to flatten the hierarchical modules into a netlist without a hierarchical structure, and to generate/output the source file of the netlist with the flattened structure. It is composed of the following components:

- Parsing: The translator first reads in the source file

in Verilog format, performing the syntax and semantic checking as well as storing each module in lists of gates, wires, inputs, and outputs.

- **Flattening:** The translator parses the source file again, but now, each time there is a module instantiation, the translator expands it with the original module definition, renaming all of the gates and wires.
- **Outputting:** Using the information stored for the root module (normally, the last module processed), the translator outputs the netlist of this module.

For simulation proper, we use DSIM, a new generation Time Warp simulator developed to support efficient parallel simulations on distributed clusters with thousands of processors [8]. DSIM features an efficient and scalable GVT (Global Virtual Time) algorithm, referred to as the Time Quantum GVT (TQ-GVT) algorithm, which does not require message acknowledgments and relies on short messages with constant length as oppose to using variable length messages with vectors.

In addition to the new GVT algorithm, DSIM uses a modified fossil collection mechanism called Local Fossil Collection, in which fossil collection is done separately by each LP individually, right before an LP attempts to process a new event. DSIM also employs an efficient event management system. For each type of event, it pre-allocates a memory buffer, whose size can be dynamically increased. As a result, the event allocation is a constant complexity operation.

On 1024 processors, DSIM achieved a 379 speedup on a simulation of a quarter million spiking neurons, yielding an event processing rate of 351 million events per second.

2.1. Building a circuit simulation in DSIM

In our gate-level circuit simulation, gates, primary inputs, and clocks are modeled as individual Logical Processes (LPs). A primary input as well as a clock can be considered as a gate, in which the output replicates the input. Primary inputs to the simulator, are in the form of a list of vector (in hex format, with digits of 0-9 and letters a/A-f/F). Decomposing a vector into bits can produce individual bits for each primary input.

The simulation starts with the LP that models the primary input. It recursively reads a vector from the input list and decomposes it to get the corresponding bits as its own input (we model it as a gate replicating its input). The time interval to read the vector is either the time interval of the data supplied, referred to as the data interval or is defined as a parameter of the simulator. The LP which models a clock works similarly to the one that models the primary input. The clock LP inputs a 0 or 1 bit alternately each clock interval.

LPs which model gates execute the gate behavior and schedule new events according to their outputs. An event consists of three items: the identifier of the LP to which the event is sent, the bit (0 or 1) representing the output of the gate (LP) sending this event, and the index of the port in the receiving gate (that is the port that is directly connected with the gate sending the event). Each event is also timestamped with the simulation time at which the event should be executed.

At the start of the simulation, an initialization stage activates the primary input LPs that initialize events (with the current simulation time) at its subordinate LPs from the first input vector in the list. They also schedule events destined to themselves with a time-stamp equal to the current simulation time plus the input data interval. The latter events, when executed, will simulate the arrival of the next input vector from the input list.

After the initialization stage, the simulator enters the main simulation loop. In the body of this loop, messages from other processors are received, if any, and the received events are placed in the future event queue. If there are stragglers, a rollback will occur, otherwise the first event at the head of the future event queue is dequeued. The time-stamp of this event becomes the current simulation time and the event is executed, potentially generating new events which are added to the queue. If the current simulated time reaches the predefined total simulation time or there is no more input vector (end of the list), the simulation stops. Otherwise, if the time quantum is reached, the TQ-GVT algorithm is invoked. If this is not the case, the the simulation loop body is executed again.

For circuit partitioning, we use a tool called hMeTiS developed at the University of Minnesota [11]. hMeTiS is a tool for partitioning large hypergraphs, especially those used in circuit design. The algorithms used by hMeTiS are based on multilevel hypergraph partitioning described in [12].

3. Simulation experiments and their results

We used the synthesized netlist of the Viterbi decoder obtained through the Synopsys [1] design compiler, which converts a design source code to a netlist file. The simulations were executed on a cluster. Each node of this cluster has 2 800-MHz Intel Pentium III processors with 512 MB memory, connected by a fast Ethernet. The Viterbi decoder circuit that we simulated consists of about 1.2M gates, with 6 primary inputs. The input supplied to our simulation is a list with 500 or 1500 vectors.

There are three factors affecting the simulation time: the total number of events committed, the percentage of the inter-processor events, and the ratio of rollbacks.

A sequential simulation of this circuit was not done, because none of the cluster nodes had sufficient memory for such a run. However, in parallel simulations, the memory usage is distributed to all of the nodes. Hence a node needs less memory than that in sequential simulation. In DSIM, one processor is used as the GVT master, so the results (averaged over 3 consecutive runs and shown in Figure 2) are for 2, 4, 8, 16, and 32 processors. Since the sequential simulation was not run, we calculated the speedup with 2 processors. We observed superlinear speedup when the number of processors changed from 3 to 5, from 5 to 9, and from 9 to 17. These speedups are attributed to a decrease in memory and cache required by the simulation at each processor as the number of processors increases. When the available memory and cache exceeds the needs, this effects disappears, so there is no superlinear speedup between 17 and 33 processors. The speedup between 17 processors and 33 processors is 1.80 for simulations with 1500 input vectors, and 1.63 for simulations with 500 input vectors. However the speedup between 3 processors and 33 processors is as high as 27.84 and 22.63 for 1500 and 500 input vectors, respectively. Hence, by increasing the number of processors by the factor of 11, we speed up the computation by the factor of 27 (or 22), a clear sign of a superlinear speedup resulting from improved memory system performance.

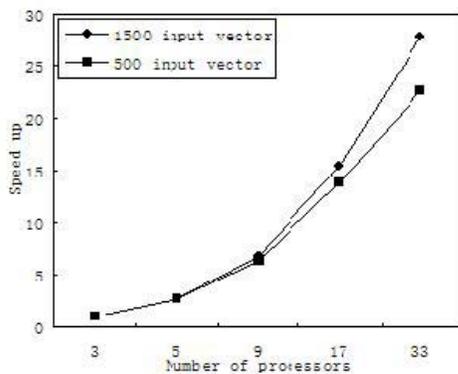


Figure 2. Simulation speedups

4. Conclusions and future research

In this paper, we presented a parallel logic simulator of a million-gate VLSI circuit using a new simulation engine called DSIM. Results show that this simulator is capable of efficiently simulating a synthesized netlist of 1.2 million gates circuit with a high speedup. Superlinear speedup is achieved for up to 17 processors. The speedup achieved when the number of processors is increased from 3 to 33 is about 28.

A good partitioning algorithm is central to the success of distributed circuit simulation, as witnessed by our own (and others) experiments. Iterative exchange algorithms such as hMeTiS, used in this paper, while effective, can become costly as circuits increase in size. Hence heuristics to decrease their execution time or the use of dynamic load balancing [4] provide important venues for our continued research.

Acknowledgment. This work was partially supported by the NSF Grant NGS-0103708. The authors thank Fei Sun for providing them with the Viterbi decoder gate netlist.

References

- [1] www.synopsys.com.
- [2] Verilog hardware description language standard. IEEE 1364-2001, 2001.
- [3] J. B. Anderson and K. Balachandran. Decision depths of convolutional codes. In *IEEE Transactions on Information Theory*, volume 35, pages 455–459, March 1989.
- [4] H. Avril and C. Tropper. Scalable clustered time warp and logic simulation. In *VLSI design*, pages 1–23, 1998.
- [5] M. L. Bailey, J. Jack V. Briner, and R. D. Chamberlain. Parallel logic simulation of vlsi systems. In *ACM Computing Surveys*, volume 26, September 1994.
- [6] P. J. Black and T. H. Meng. Hybrid survivor path architectures for viterbi decoders. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 433–436, April 1993.
- [7] E. Boutillon and N. Demassieux. High speed low power architecture for memory management in a viterbi decoder. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 284–287, May 1996.
- [8] G. Chen and B. Szymanski. Dsim: Scaling time warp to 1,033. In *Proc. Winter Simulation Conference, WSC05*, 2005.
- [9] R. M. Fujimoto. Parallel discrete event simulation. In *Communications of the ACM*, volume 33, pages 30–53, 1990.
- [10] IEEE Std. 1076-2002. *IEEE Standard VHDL Language Reference Manual*, 2002 edition.
- [11] G. Karypis and V. Kumar. Hmetis, a hypergraph partitioning package.
- [12] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning.
- [13] H. K. Kim. *Parallel Logic Simulation of Digital Circuits*. Phd thesis, Wright State University, 1998.
- [14] L. Li, H. Huang, and C. Tropper. Towards distributed verilog simulation. *I.J. of SIMULATION*, 4(3–4):44–54.
- [15] L. Li, H. Huang, and C. Tropper. Dvs: An object-oriented framework for distributed verilog simulation. In *Proc. Seventeenth Workshop on Parallel and Distributed Simulation (PADS'03)*, 2003.
- [16] G. Meister. A survey on parallel logic simulation. Technical report, Department of Computer Science, University of Saarland, 1993.
- [17] D. L. Richard. Parallel and distributed vhdl simulation.
- [18] S. Williams. Icarus verilog. [Http://icarus.com/eda/verilog](http://icarus.com/eda/verilog).