1994

Logic Simulation using an Asynchronous Parallel Discrete-Event Simulation Model on a SIMD Machine

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Lee Gowen
University of Nebraska-Lincoln

Matt Payne
University of Nebraska-Lincoln

Don Sylwester
University of Nebraska-Lincoln

doi: 0.1109/ICVD.1994.282635

Logic Simulation using an Asynchronous Parallel Discrete-Event Simulation Model on a SIMD Machine

Sharad Seth, Lee Gowen, Matt Payne, and Don Sylwester
Department of Computer Science and Engineering
University of Nebraska - Lincoln

Abstract

The Chandy-Misra-Bryant (CMB) model has been applied to logic simulation of synchronous sequential circuits using a massively parallel SIMD computer, a CM-2 Connection Machine. Several methods of reducing message traffic in a logic simulation have been adapted to the SIMD architecture of the CM-2, with the result that each method of reducing message traffic actually decreases the speed of the simulation. This suggests that communication costs required to support logic simulation are small compared to the cost of deciding which messages need not be sent.

1 Introduction

Logic simulation on a parallel computer is an intriguing idea; the goal being to match the parallelism possible among the elements of a logic circuit with the parallelism of the processor supporting the simulation.

Several new models for asynchronous parallel discrete-event simulation (PDES) have been proposed [4] which distribute the simulation over a network of parallel processors that exchange messages to communicate state changes. The Chandy-Misra-Bryant [3, 2, 5] approach is a conservative model, restricting processors from simulating past the latest time for which correct results can be guaranteed.

The asynchronous distributed simulation model has the following features [4, 5]. The physical system is divided into a set of processes which are mapped in some fashion to the available parallel processors. Each process is able to operate asynchronously with respect to the other processes, and may either send or receive messages from the other processes. Messages may take an arbitrary time to traverse the communication channel, but messages sent along a specific channel must arrive in the same order in which they were sent. Each message is time stamped with the local time of the process sending the message. A process that receives messages on several input channels can correctly deduce its own behavior as a result of these messages up to the time given by the minimum of the times of the latest message received on each channel.

The major problem with the asynchronous model is the possibility of deadlocks [6]. It is not generally necessary that each message corresponding to some communication in the actual physical system be sent in the simulation. In fact, restricting the number of messages sent is important in achieving maximum speed in the simulation since communications resources are usually a critical resource in parallel computer systems. It is unfortunately possible that if some messages are not sent then some processes will be unable to advance their local clocks and a deadlock might occur. A variety of techniques have been suggested for avoiding or preventing deadlocks.

Our work parallels that of Soule and Gupta [7] in applying the CMB algorithm, but we use a different architecture, a massively parallel SIMD machine. We find that attempts to reduce the number of messages sent, essential for the machine model used by Soule and Gupta, are counterproductive for the specific architecture of CM-2, a massively parallel SIMD machine.

2 Parallel Asynchronous Simulation

We have applied an asynchronous PDES distributed processing model to the simulation of a sequential logic circuit using a massively parallel SIMD computer. Our physical process will be an individual gate, the communications channels between processes are the wires connecting the pins on the gates, and the state changes of input and output pins will be our messages.

Apart from those aspects of the implementation specifically related to the parallel implementation, the simulation is straightforward. Three valued logic is used throughout, unit delays are assigned to each gate and zero delay to inter-gate connections. Gates are evaluated in parallel across all processors using table

0-8186-4990-9/94 $3.00

lookup.

Critical, then, to the parallel implementation are the selection of a communication mechanism on the CM-2, and selection of a method for handling the potential for deadlocks in the simulation, adapted to a parallel architecture.

In the simulator a small number of gates, typically one, are mapped to a logical processor. The wired connection between the output of one gate and the input of another is modeled as a message path between the logical processors storing the gates. Logical processors evaluate their gates when they receive input messages and send appropriate messages to the gates on their fanout lists. The maximum speedup possible for a synchronous simulation is bounded by the maximum circuit activity [1]. This bound may not apply to an asynchronous simulator.

2.1 Communication Mechanism

The Connection Machine allows several different communication mechanisms, with the expected tradeoff between ease of use and efficiency. General communication, the least restrictive, allows processors to send data to or receive data from any other processor by specifying its location. Hardware is responsible for routing all messages over the limited number of communication paths available. Grid communication restricts communication to processors separated by a fixed distance in a given operation but allows all processors in the system to simultaneously communicate. In simple experiments designed to compare these two mechanisms general communication took approximately 3 times as long as grid communication. However, given the irregular topology of a logic circuit, it seems difficult to map the gates to the logical processors to insure that communication always occurs between processors separated by a fixed distance. If there are several different separation distances among the processors that must communicate in a given step of the simulator then the grid communication mechanism requires that each distance be handled in a separate operation. As a result we have selected the general communication mechanism for our simulator. This choice also simplified the problem of partitioning gates among processors. With the general communication mechanism gates could be partitioned randomly among the available processors.

2.2 Minimization of Message Traffic and Deadlocks

The message bandwidth on the Connection Machine, or any other parallel processor, is a limiting resource, and minimization of the message traffic is expected to be a major factor in the speedup possible in an asynchronous simulation. If a gate changes state, a message must be sent to all gates on the fanout list. This message is time stamped with the local clock value of the sending gate, and receipt of this message possibly updates the local clock of the receiving gate. If a gate receives an input event and is evaluated, but the output state does not change then it is not immediately necessary to send a message to the gates on the fanout list, since no immediate change in their state would result. Such a message, which only communicates an updated time stamp and not a state change, is called a *null message*. If a null message is always sent then the simulation proceeds normally to completion but the number of total messages sent is a maximum. If all null messages are suppressed then, while the number of total messages is a minimum, it is likely that the simulation will deadlock. Deadlocks might occur because the local clock for a gate cannot advance past the minimum of the time stamps of the most recent messages received on its input pins. To advance past this time would be equivalent to predicting the future. Several approaches to minimizing the message traffic have been suggested in the literature [3, 7]. We have adapted several of these methods to the SIMD architecture of the CM2, with descriptions following in section 3.

3 Deadlocks: Avoidance, Detection and Resolution

We have implemented a variety of methods for handling deadlocks, ranging from methods that prevent them from occurring to methods that allow the deadlock to occur and then take action to recover. Deadlock resolution methods differ in the way they handle null messages. A null message does not change the logic value but it may carry a more recent time stamp. Since a gate can only advance its own clock to the minimum of the time stamps of its input messages, plus the delay of the gate, if it does not receive a null message it might be prevented from accepting a non-null message from another gate, resulting in a deadlock. It is clear that all non-null messages, which reflect real changes in logic values, must be sent. Presumably, restricting the number of null messages, and thereby reducing the message traffic on the CM2 computer, will increase the speed of the simulation.

3.1 Deadlock Avoidance

All null messages: If all null messages are sent with no delay then all local clocks are advanced as often as possible and a deadlock cannot occur.

Null messages after a delay: If null messages are sent only after a delay of one or more cycles, then it is possible that a real message may be generated in that gate during those next few cycles, removing the need to send the null message and reducing the message traffic. Since all null messages, or replacement real messages, are always eventually sent a deadlock cannot occur.

Null messages assumed: Upon receiving any message, real or null, a gate evaluates, effectively assuming it has received null messages with the same time stamp on all inactive input pins. This assumption might be initially wrong, leading to temporarily incorrect value on the output line and a violation of the CMB model, but when the circuit is finally stable all lines must have correct values.

3.2 Deadlock Detection and Resolution

The following three methods suppress all null messages, wait for a deadlock to occur and then recover from it by forcing selected null messages to be sent. It is reasonable to expect that there is some relatively small set of gates which can resolve the deadlock by sending null messages. It is not easy to determine which gates are in this set. These three methods select the set in different ways. During this deadlock resolution phase each gate is required to send a null message once if it receives one, an action it would not take during the normal communication phase.

Demand-driven: Presumably those gates with minimum local clock times are the focus of the deadlock. They have not received null messages to advance their own clocks, and are prevented from sending null messages to their fanout list. We identify the gates with minimum local clock values, move several levels back up their fanin cone, typically 2 levels, and mark these gates to send null messages. These gates must have local clock times greater than the set of gates with minimum clock times (otherwise they would be in that set) so the null messages must eventually cause the gates in the minimum clock value set to advance.

Source gates send null messages: The source gates, the D flip-flops and the input gates, have internal clocks that are advanced with the global system clock. Normally D flip-flops and input gates send messages only when a new input vector is accepted and the system is clocked. To resolve the deadlock we require that each source gate send a null message containing the current global time.

All gates send null messages: When a deadlock is detected, all gates send a null message. While this method generates more message traffic than the previous two methods the time required to mark the gates is minimal since all gates are marked and this marking is a single parallel operation on the CM2.

4 Results

Table 1 displays performance data for three test circuits for each of the six deadlock avoidance/resolution methods. Test circuits were selected primarily for size, s5378 and s35932 occupy most of one and four sequencers, respectively, when one gate is assigned to each physical processor.

Table 1: Gate Evaluations per Second for Three Test Circuits.

Circuit	s344	s5378	s35932
Test Vectors	88	469	295
Tot. Gate Evaluations	13681	436415	3673197
Avoidance			
Null Messages Sent			
With Delay 0	252	848	3336
With Delay 1	261	980	3250
With Delay 5	66	*	*
Nulls Assumed			
With Nulls Sent	239	1281	10152
Without Nulls Sent	638	1869	15819
Resolution			
Demand Driven	84	*	*
Source Gates Send	114	*	*
All Gates Send	134	*	*

* Time limits on runs exceeded. Performance can be assumed to be worse than the slowest performance in the column.

In all cases, the deadlock avoidance schemes performed better than the deadlock resolution schemes. The best performance was achieved for the s35932 circuit: 3336 gate evals/sec with null messages sent without any delay and 15819 gate evals/sec when null messages are assumed and not sent.

In absolute terms, this performance is no better than that achievable by conventional logic simulation on a state-of-the-art workstation, bearing in mind, however, that the CM-2 hardware does not represent state-of-the-art. In a companion experiment a synchronous parallel event-driven simulator running on the same CM-2 hardware achieved a performance of 9738 gate evaluations per second.

4.1 Timing

The performance results suggest that efforts to reduce message traffic, possibly incurring the need to resolve deadlocks, is counterproductive since the overhead required to handle communications does not seem to vary much with the size of the circuit. To verify this we performed a simple experiment, artificially activating groups of gates to isolate and time the three phases of each cycle of the main simulation loop: evaluation, deadlock resolution, and communication, denoted by E, D, and C in the following discussion. The results are shown in Table 2, where G is the number of gates assigned to one CM-2 processor.

Note that the time for the C-phase, while not constant, is increasing very slowly in proportion to the size of the circuit. This suggests that the message traffic required to do logic simulation is well within the bandwidth of the CM2, and it may not be worthwhile to attempt to reduce message traffic at the expense of executing additional code in the deadlock resolution phase.

The two timings for s35932 have two and four gates assigned to one CM-2 processor, respectively. Some of the external message traffic between CM-2 processors in the 2:1 case is converted to message traffic internal to one CM-2 processor in the 4:1 case, and the speed of the simulation increases slightly as the communication time decreases slightly.

Table 2: Timing Data

Circuit	G	Cycles	Tot E	Tot C	C/cycle
s344	1	1290	15 s	17 s	.013 s
s5378	1	10571	123 s	187 s	.018 s
s35932	2	7264*2	169 s	345 s	.024 s
s35932	4	7264*4	338 s	584 s	.020 s

5 Conclusion

Deadlock resolution in CMB on a CM-2 is more expensive than deadlock avoidance.

When we began this implementation of a logic simulator using the CMB approach we expected the simple variant that sends all possible null messages to be the slow benchmark against which other methods would be compared. However, the methods which minimize the sending of null messages and then resolve the resulting deadlocks when they occur by sending selected null message are substantially slower. The time required to execute additional instructions in the SIMD processor array to decide which messages need not be sent exceeds the time required to send those messages.

Random partitioning of gates to processors is sufficient.

The experiments described in section 4.1 indicate that the communication load required by logic simulation, due to the generally low and irregular gate activity, and due to the sparse nature of the resulting message traffic, is not a limiting factor in the speed of a PDES logic simulator on a CM-2. Further, our results suggest that the partitioning problem, the mapping of gates to processors, is satisfactorily solved by random partitioning.

Finally, it must be pointed out that any SIMD simulation will have a certain amount of sequential code which places absolute limits on its performance according to the Amdahl's law.

Acknowledgments

The authors gratefully acknowledge the assistance of the following: V.D. Agrawal, Bell Labs; S. S. Thatte, (formerly at UNL); A. Grothe, UNO; and P. Kenyon, W. Mahoney, and V. Sivaramakrishnan, UNL.

References

[1] V. D. Agrawal and S. T. Chakradhar. Logic simulation and parallel processing. *27th Design Automation Conference*, 1990.

[2] R. E. Bryant. Simulation of packet communications architecture computer systems. Technical Report MIT-LCS-TR-188, MIT, 1977.

[3] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *CACM*, 24:198–206, November 1981.

[4] R. Fujimoto. Parallel discrete event simulation. *CACM*, 33:30–53, October 1990.

[5] J. Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39–60, March 1986.

[6] D. A. Reed and A. D. Malony. Parallel discrete event simulation: The Chandy-Misra approach. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):8–13, July 1988.

[7] L. Soule and A. Gupta. Parallel distributed-time logic simulation. *IEEE Design & Test of Computers*, pages 32–48, December 1989.