

PARALLEL LOGIC LEVEL SIMULATION OF VLSI CIRCUITS

Rajive Bagrodia, Zheng Li, Vikas Jha, Yuan Chen, and Jason Cong

Department of Computer Science
University of California, Los Angeles, CA 90024

ABSTRACT

Interest in the exploitation of parallelism in circuit simulation has been increasing steadily. In this paper, we study parallel logic level simulation of combinational VLSI Boolean networks using both conservative and optimistic simulation algorithms. In particular, we describe a logic level circuit simulator that uses an acyclic multi-way network partitioning algorithm to decompose Boolean networks and an algorithm-independent simulation language that allows a discrete-event simulation model to be executed using a variety of simulation algorithms. The simulator has been implemented on an IBM SP1 supercomputer and was used to simulate a set of combinational Boolean circuits from the ISCAS85 benchmark suite. Our results show that it is feasible to obtain speedups for even relatively small circuits using both conservative and optimistic methods.

1. INTRODUCTION

Circuit simulation is a critical bottleneck in the design of complex VLSI chips. Even though the clock speed and processing power of general purpose workstations is improving constantly, the increase in size and complexity of VLSI chips indicates that this problem is likely to get worse. The simulation complexity is such that it routinely requires a few days or even weeks to simulate processor level designs on contemporary desktop workstations. It has been reported that the final verification of a supercomputer developed by Ardent required two weeks using two workstations [Soule, 1992]. Further, as the size of the simulated system grows, the memory requirements may increase beyond the limits of the main memory that is typically available on a single workstation. The need for efficient and cost-effective circuit simulation is hard to overstate.

A promising approach for this problem is the use of general purpose parallel machines for circuit simulation. This approach is referred to as parallel circuit simulation and may be used to execute both functional and circuit-

level models on parallel architectures. Parallel circuit simulations have significant advantages when compared with the use of special-purpose hardware: first, unlike the hardware solutions, these techniques are not inherently restricted to work with specific types of elements or timing models, or to circuits of specific size. Second these techniques are inherently scalable; the software approach can directly track the processor technology curve and derive immediate benefits from advances in processor network designs. Further, as the interconnection technology advances and it becomes feasible to provide low-latency connectivity among an increasingly larger number of processors, this increase in computational power can be immediately harnessed by the parallel simulation approach. Lastly, the software approach can directly support mixed-mode circuit simulations, where different sub-circuits may be simulated using functional, logic, or switch level models.

For parallel circuit simulation to be effective, the circuit must contain inherent parallelism, which may be available as either synchronous or asynchronous parallelism. Synchronous parallelism exploits concurrency if the circuit has a large number of simultaneous events. However, previous measurements by Wong [1986], and Bailey and Snyder [1988] have reported small concurrency levels that are unlikely to yield significant and scalable performance improvements.

Asynchronous parallel circuit simulation introduces two significant problems: circuit partitioning and synchronization. The former refers to the decomposition of the original circuit into a number of subcircuits, such that the computation among the subcircuits is approximately balanced and the communication is minimized. Although a large number of partitioning methods have been devised, relatively few have been designed specifically for parallel circuit simulation. For instance, most partitioning algorithms ignore the direction of signal propagation and model the network as an undirected graph. As we demonstrate in this paper, if signal direction is included in the partitioning algorithm, it is possible to construct acyclic partitions that reduce synchronization overheads for some

synchronization algorithms. The synchronization is required to ensure that incoming signals at each subcircuit are processed in their correct global order. Three primary synchronization mechanisms have been used: the conservative parallel discrete-event simulation (PDES) protocols [Misra, 1986], the optimistic PDES protocols [Jefferson, 1985], and the recently proposed adaptive protocol that combines the hitherto disparate protocols [Jha and Bagrodia, 1994].

In this paper, we present experimental results on logic level simulation of circuits using a new acyclic partitioning algorithm [Cong, Li, and Bagrodia, 1994] where the parallel simulation of the partitions is synchronized using both conservative and optimistic algorithms. A number of circuits from the ISCAS85 benchmark suite were selected for simulation. Parallel execution was effective in reducing the simulation time for even the small circuits containing about 1500 gates. The next section describes related work in the area of parallel circuit simulation. Section 3 gives a brief description of the parallel simulator and its implementation. Section 4 describes the acyclic partitioning algorithm. Section 5 describes the benchmark circuits used in the experiments and presents the results of both sequential and parallel simulation of these circuits. Section 6 is the conclusion.

2. RELATED WORK

Previous work on the application of asynchronous techniques have been used primarily for logic simulation, rather than switch-level simulations. Perhaps the earliest comprehensive experimental study was by Su and Seitz [1989], who studied the effectiveness of six variants of the conservative simulation protocol based on deadlock-avoidance using null messages. They reported a factor of 2 speedup on 32 nodes and between 5-10 on 128 nodes. Subsequently, Soule [1992] applied the conservative techniques to the logic level simulation of a number of benchmark circuits ranging from a small 5,000 element 16-bit multiplier to a large 24,600 element multiprocessor directory cache controller. The models were executed on a 16-processor Encore Multimax and on the Stanford DASH multiprocessor. The researchers noted only slight speedups for the various circuits and noted that the deadlock resolution overheads limited the speedups.

Optimistic techniques have also been used for logic simulations. The most successful parallelizations have been obtained by Briner, et al. [1991], who evaluated the speedups for logic simulation of circuits using an optimistic protocol. Reported speedups measured on 32 nodes of a BBN Butterfly ranged from 7 for a 3,360 transistor

circuit to almost 25 for a 30K transistor circuit. Sporrer and Bauer [1993] used TimeWarp on a network of workstations for logic simulation of the ISCAS89 benchmarks using a hierarchical, cluster-based partitioning algorithm. On a network of 20 workstations they reported a peak speedup of about 8, *relative to a 1 processor implementation of the distributed simulator*.

3. SIMULATION ENVIRONMENT

All sequential and parallel simulations reported in this paper were executed using the Maisie simulation environment. Maisie is among the few existing languages that supports the execution of a discrete-event simulation model with multiple algorithms and provides constructs to reduce the simulation overheads with both conservative and optimistic parallel algorithms. The simulation algorithms currently supported by Maisie include a sequential algorithm, parallel conservative algorithms based on null messages [Misra, 1986] and conditional events [Chandy and Sherman, 1989], a new conservative protocol that combines null messages with conditional events [Jha and Bagrodia, 1993], and a parallel optimistic algorithm [Bagrodia, Chandy, and Liao, 1992]. A complete definition of Maisie may be found in [Bagrodia and Liao, 1992]. An overview of Maisie is also given in a companion paper in this volume [Bagrodia, 1994]. Conservative implementations of Maisie with three different protocols including the null message protocol, the conditional event algorithm, and a new protocol that combines the preceding approaches has been described in [Jha, Bagrodia, 1993]. The implementation of Maisie with the optimistic space-time algorithm is in [Bagrodia, Liao, 1992].

4. CIRCUIT PARTITIONING AND SIMULATION

To simulate a circuit on a parallel architecture, the design must be partitioned into a number of subcircuits, such that each subcircuit can be simulated on a separate processor. The partitioning algorithm can have a significant impact on the performance of the parallel implementation, as a sub-optimal partitioning may imply frequent and (possibly unnecessary) communications among the processors, forcing synchronization to occur much more frequently than is necessary for the given circuit.

4.1. Acyclic Multiway Partitioning

A number of recent algorithms have been proposed that attempt to optimize the partitioning for circuit simulations. Most existing partitioning algorithms model the circuit as

an undirected graph and ignore signal direction during the partitioning process. However, analyzing signal direction during the partitioning process can be useful. For instance, it is possible to use signal direction information to construct acyclic partitions. Acyclic networks have been shown to have much better performance than networks with feedbacks for queueing systems, and the same is likely to hold for circuit simulations. In this section, we briefly describe a class of new acyclic multi-way partitioning algorithms [Cong, Li, and Bagrodia, 1994] that were used for parallel execution of the circuit models.

We begin with an extension of the Fiducia-Mattheyses algorithm [Fiducia and Mattheyses, 1982], referred to as the K-FM algorithm. This algorithm starts with a randomly generated, balanced, initial partition and subsequently moves gates among the partitions. At each step, gates that can be moved to another partition are identified. A gate is feasible for moving, if moving it to another partition will not violate the balance constraint. Among all feasible gates in a partition, the gate with the highest gain is selected, where the *gain* of a gate is defined to be the amount of reduction in the cut size that results from moving the gate to the other partition. When all feasible moves have been inspected, the best partition encountered during the current pass is saved as the initial partition for the next pass. The algorithm terminates when a pass makes no improvement to the partitioning solution.

The preceding partitioning algorithm can be used to generate acyclic partitionings by imposing additional constraints that maintain an acyclic dependency graph. This extension, referred to as K-AFM uses an initial acyclic partitioning based on a random topological ordering of the initial circuit. The criteria for choosing a feasible move in the K-AFM algorithm is modified to select a gate that maximizes gain under both acyclic and area constraints. However, experimental results on the ISCAS benchmarks indicate that this algorithm did not yield significant improvements on the cut sizes for the partitions [Cong, Li, and Bagrodia, 1994].

The next algorithm used clustering prior to the partitioning phase to improve the cut size. The circuit was first clustered using the maximum fan-out free cone (MFCC) [Cong and Ding, 1993] strategy and the preceding K-AFM algorithm was then applied to the clustered network. The resulting algorithm is referred to as the K-MAFM algorithm. Compared to existing algorithms, the K-MAFM algorithm was found to produce significant reductions in the cut size of the partitions.

4.2. Gate Level Simulation

The ISCAS benchmark includes simple circuits that contain only combinational elements; each element is assigned a constant rise and fall time, with all instances of a given element assigned the same rise and fall times. A given circuit is decomposed into a set of partitions as described in the previous section. The input lines in each partition either correspond to primary inputs or outputs from another partition. In the simulator, each partition is programmed as a Maisie entity called partition. A **driver** entity is also defined which is responsible for initiating the simulation, creating the partition entities and terminating the program. The simulation is executed for a given number of input vectors; for each vector, the driver packs the signals that correspond to the primary inputs for a given partition into a single message and sends the message to the corresponding entity. The partition entity is described next.

The gates and lines that belong in a given partition are represented by appropriate data structures within the corresponding entity. Each entity contains two event-lists, called the *in_event_list* and *out_event_list*, respectively. At each simulation instance say t , the entity first processes events with timestamp t in the *in_event_list*. Each event corresponds to a change in the signal value for a given line; if the new value changes the output of the corresponding gate, an event is scheduled at the appropriate future time. If the output line belongs to the same partition, the event is inserted into the *in_event_list* and is inserted in the *out_event_list* otherwise. When the entity has exhausted all events with timestamp t from its internal list, it checks to determine if the *out_event_list* contains any outgoing events with timestamp t ; if so, it propagates them to their destination entities. All outgoing events from one partition to another are packed together and sent as a single message. Note that if the underlying simulation algorithm was known to be optimistic, the output signals could be transmitted at the time they are generated (and subsequently canceled using rollbacks if necessary; the impact of this modification has not yet been evaluated). After processing events from its *out_event_list*, the entity schedules a conditional timeout event corresponding to the earliest timestamp in its *in_event_list*. Note that it is possible for the entity to receive an input message from some other partition prior to receiving the timeout event, in which case the input event is processed as described above and the timeout event is rescheduled.

5. EXPERIMENTAL RESULTS

The benchmark examples were selected from the ISCAS85 suite, a set of combinational circuits that include the signal direction information. Table 1 shows the characteristics of the benchmark circuits in terms of the number of gates, number of PIs, number of nets, and number of edges. Although these circuits are relatively small, they were easily available; experiments with larger circuits are in progress.

The sequential and parallel results reported in this paper were executed on 16 nodes of an IBM 9076 SP1 parallel computer. The IBM SP1 is a distributed memory machine that consists of a set of RS/6000 workstation processors connected by a high speed switch. Each node has a main memory of 128 megabytes and is connected to a 1 Gigabyte disk that is used for temporary file storage. The sequential experiments were executed on a single node of the IBM SP1 using a global event-list implementation of

Circuit	No. of gates	No. of PIs	No. of nets	No. of edges
C2670	1193	233	1426	1983
C3540	1667	50	2167	2911
C5315	2307	178	2485	4331
C6288	2418	32	2450	4800

Table 1. Benchmark Circuits.

	partition	c5315	c6288
unpartitioned	1	64.1	122.8
K-MAFM	2	34.5	69.0
	4	22.0	37.4
	8	19.4	26.2
	16	15.1	17.4
K-FM	2	36.7	64.3
	4	22.4	56.0
	8	20.0	35.5
	16	16.6	29.9
K-AFM	4	34.2	38.1
	8	25.4	26.2
	16	20.4	14.8
Conservative	1	60.4	113.2
Optimistic	1	58.3	141.4

Table 2. Execution Time (secs) (Sequential Algorithm)

Maisie, where the event-list was implemented as a splay tree. The parallel experiments were executed using the conservative and optimistic algorithms described in the previous section.

5.1. Sequential Simulation Experiments

The benchmark circuits were partitioned using each of the three algorithms described previously and the execution time for the sequential runs was measured as a function of k , the number of partitions, for $k=2,4,8$, and 16, respectively. Each experiment was executed for 250 simulation time units. Table 2 shows the impact of the partitioning algorithm on the execution time for two of the benchmark circuits.

As seen from Table 2, the execution time decreases almost monotonically as the number of partitions increase. The speedup obtained for the sequential implementation was as much as 1.86 for 2 partitions, 3.6 for 4 partitions, 4.68 for 8 partitions, and 8.3 for 16 partitions. As the number of partitions for a given circuit increases, the size of *in_event_list* for each partition decreases and the total time spent in list-management activities including insertion, deletion, and search decreases leading to improvements in the execution time. A similar phenomenon was noticed in the simulation of the shark's world model, where model decomposition was found to improve the performance of the sequential implementation [Bagrodia and Liao, 1990].

For a given number of partitions, the acyclic clustered partitioning (K-MAFM) yielded the best performance. The superior performance of this algorithm was primarily because it generated partitions with the smallest cut-sizes, and it had the fewest signals sent across the partitions. The undirected K-FM partitions resulted in solutions with a larger cut-size, but each message exchanged between the partitions was packed with fewer signals than with the K-MAFM partition resulting in more total message exchanges than in the K-MAFM algorithm. This caused a significant degradation in the performance of the K-FM implementation. Although the cut size for the acyclic K-AFM algorithm was much larger than that of the K-MAFM algorithm, its performance is better than the cyclic K-FM algorithm because it typically had a higher number of signals packed in each message. The larger cut-size of the K-AFM also implied that the internal event-list in each partition contained a larger number of entries, which degraded its performance in comparison to that of the clustered acyclic algorithm. Note that the topology of the original circuit also influenced the relative performance of the two acyclic algorithms.

5.2. Parallel Simulation Experiments

The Maisie model was subsequently refined for parallel execution. The only change required to the sequential model was to distribute the partition entities among available processors.

We begin with a detailed look at the two largest circuits in the benchmark suite, namely, c6288 and c5315. The circuits were partitioned using the K-MAFM, K-FM and K-AFM algorithms and the timings were measured for parallel implementations using 4, 8, and 16 partitions. The results are tabulated in Tables 3 and 4 and displayed graphically in figures 1a-e. The legend for the graphs is given separately in Figure 1a. The timings clearly demonstrate the considerably better performance of the acyclic partitionings. For conservative implementations, acyclicity implies less blocking for null messages and for the










- K-MAFM, 4 
- K-FM, 4 
- K-AFM, 4 
- K-MAFM, 8 
- K-FM, 8 
- K-AFM, 8 
- K-MAFM, 16 
- K-FM, 16 
- K-AFM, 16 

Figure 1a Legend Used for Figures 1b-e

partition		c5315 on n nodes			c6288 on n nodes		
		n=4	n=8	n=16	n=4	n=8	n=16
K-MAFM	k=4	8.88	-	-	15.49	-	-
	k=8	10.41	4.7	-	9.64	6.47	-
	k=16	20.27	13.91	4.44	10.05	6.354	3.1
K-FM	k=4	12.66	-	-	22.76	-	-
	k=8	20.97	11.28	-	29.51	27.12	-
	k=16	48.97	31.56	7.01	49.43	33.44	11.78
K-AFM	k=4	10.704	-	-	18.05	-	-
	k=8	19.21	6.c2	-	10.25	6.33	-
	k=16	21.94	13.94	4.83	8.28	5.89	3.64

Table 3. Execution Time using Conservative Algorithm

partition		c5315 on n nodes			c6288 on n nodes		
		n=4	n=8	n=16	n=4	n=8	n=16
K-MAFM	k=4	16.59	-	-	34.67	-	-
	k=8	41.01	7.47	-	34.7	12.59	-
	k=16	56.88	25.97	6.91	22.68	14.25	6.20
K-FM	k=4	43.18	-	-	61.39	-	-
	k=8	51.28	25.95	-	66.99	58.77	-
	k=16	43.06	36.97	23.09	38.33	38.88	24.13
K-AFM	k=4	15.70	-	-	33.61	-	-
	k=8	52.32	8.29	-	35.83	9.81	-
	k=16	75.91	38.23	9.32	26.92	16.53	6.56

Table 4. Execution Time using Optimistic Algorithm

optimistic implementations, it reduces the probability of rollbacks leading to better performance. The table also includes that given n processors, an n -partition yielded the best performance. Thus, unlike the sequential implementation, having multiple partitions on a processor does not yield better performance.

Figure 2 plots the self-relative speedup for both conservative and optimistic algorithms for the c5315 and c6288 circuits. The self-relative speedup was defined as $t_{par}(n,n)/t_{par}(1,1)$, where $t_{par}(n,n)$ is the execution time for the n -partition n -processor parallel implementation of a given circuit and $t_{par}(1,1)$ is time for the 1-processor implementation for the corresponding conservative or optimistic model. The times for the 1-processor conservative and optimistic implementations are included at the bottom of Table 2.

Table 5 shows the performance of the two smaller circuits in the benchmark. The table only reports the performance of the n -partition, n -processor mapping using the K-MAFM partitioning algorithm, as this configuration was found to be the most efficient for the larger circuits. For both conservative and optimistic runs, the best speedups are obtained using 8 processors. The speedups for the smaller circuits are better than those for the larger circuits described earlier, particularly for the optimistic implementation. The primary reason for this is the considerably lower state saving overhead for the small circuits.

In Figure 3, we plot the speedups obtained for the k -partition parallel implementation. The speedup is defined as $t_{par}(n,n)/t_{seq}(n)$, where $t_{par}(n,n)$ is the execution time for the n -partition n -processor parallel implementation of

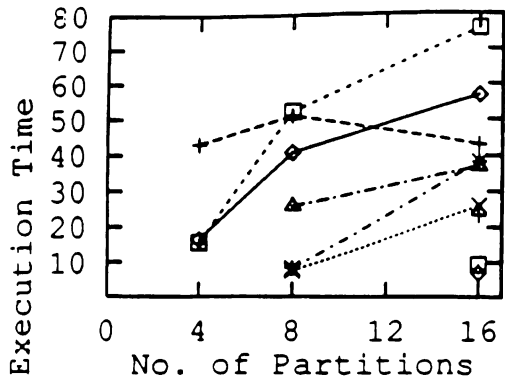


Figure 1b Execution Time for c5315 (Optimistic)

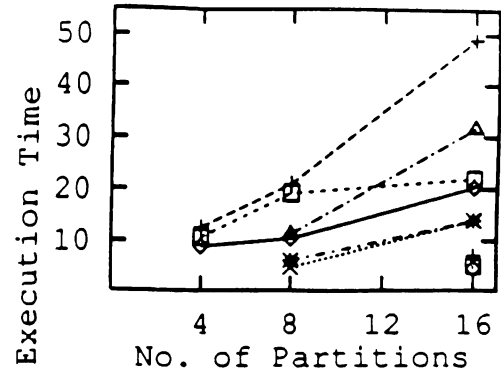


Figure 1c Execution Time for c5315 (Conservative)

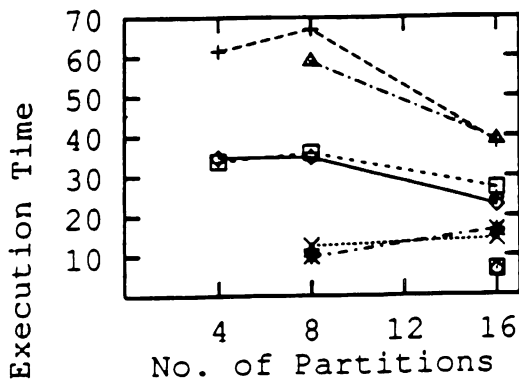


Figure 1d Execution Time for c6288 (Optimistic)

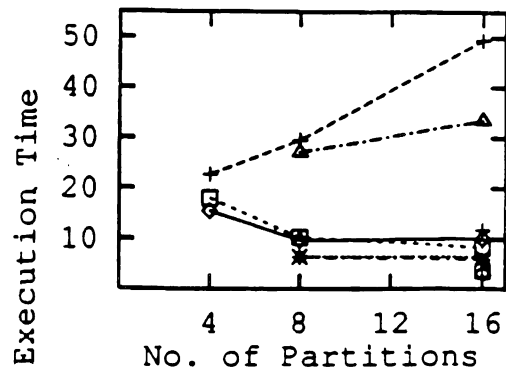


Figure 1e Execution Time for c6288 (Conservative)

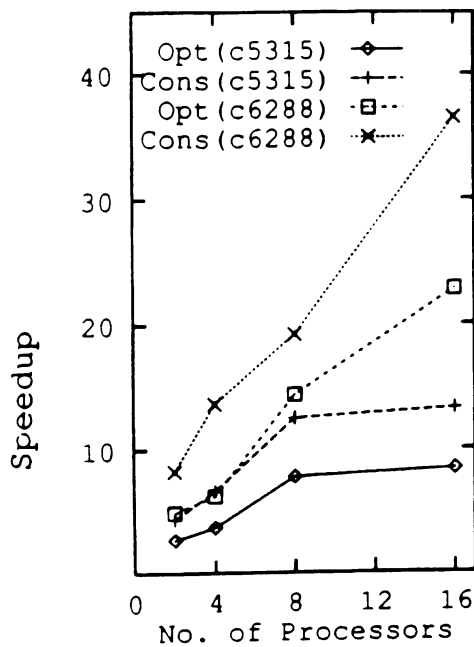


Figure 2 Self-relative Speedup for c5315 and c6288

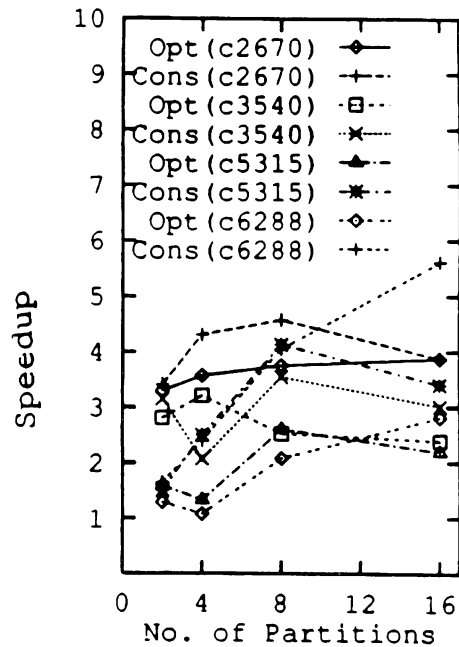


Figure 3 Speedups w.r.t Corresponding Sequential Runs (using K-MAFM)

Partition	Sequential		Conservative		Optimistic	
	c2670	c3540	c2670	c3540	c2670	c3540
1	29.2	50.3	12.229	21.873	11.92	23.742
2	17.2	28.3	5.05	8.97	5.21	10.07
4	10.9	20.1	2.53	5.26	3.05	6.26
8	11.3	13.0	2.47	3.19	3.01	5.16
16	9.9	13.4	2.56	3.29	5.89	5.61

Table 5. Execution Time for c2670 and c3540 (using K-MAFM, one entity on one processor)

a given circuit and $t_{seq}(n)$ is the execution time for the corresponding n -partition sequential implementation. As noted from the graph, for the bigger circuits, the speedup improves monotonically as the number of processors is increased to 16. For the two smaller circuits, the speedup increases initially, but decreases beyond 8 processors because of the additional communication overhead.

Table 6 summarizes the speedups obtained using both conservative and optimistic algorithms as compared with the best sequential implementation. For each circuit, the lowest parallel execution times were identified for both conservative and optimistic methods. This time is included in the column titled $t_{par}(n)$; the number in the parenthesis refers to the number of processors that were used to obtain the optimal time. The column titled $t_{best}(n, algo)$ includes the best time obtained for the sequential execution of that circuit across all sequential implementations considered in this paper. The parentheses indicate the partitioning algorithm and the number of partitions that yielded the best sequential time. We note that overall the conservative algorithm performed better than the optimistic implementations.

As checkpointing costs were found to be the biggest source of overhead for the optimistic implementations, we experimented with a number of alternative checkpointing

frequencies. For the acyclic partitioning using K-MAFM and K-AFM, checkpointing after every 15 events was found to yield the best performance, whereas for the undirected partitioning, checkpointing after every 5 events was found to yield the best performance. The reported times used the checkpointing frequency that optimized the performance.

6. CONCLUSION AND FUTURE RESEARCH

In this paper, we have demonstrated the feasibility of obtaining speedups in parallel logic simulation of circuits on distributed memory architectures using both conservative and optimistic methods. The experiments reported in this paper were restricted to relatively small circuits with less than 3000 gates. Extensions of this work to larger circuits, including circuits that contain sequential elements, is in progress. The impact of algorithm-specific optimizations, including programmer-specified lookahead and aggressive optimistic execution is also being investigated.

ACKNOWLEDGMENTS

This work is partially supported by the California MICRO program with Cadence, Hewlett-Packard, and Zycad, by ARPA/CSTO under Contract J-FBI-93-112, and by the NSF Young Investigator Awards under ASC-9157610 and MIP-9357582.

REFERENCES

- Bailey M. and Snyder L. 1988. An Empirical Study of On-Chip Parallelism *Proceedings of the 25th ACM/IEEE Design Automation Conference*
- Bagrodia R. 1994. Language Support for Parallel Discrete-Event Simulations *1994 Winter Simulation Conference*
- Bagrodia R., Chandy K., and Liao W. 1992. A Unifying Framework for Distributed Simulations *ACM*

Circuits	$t_{best}(n, algo)$	Conservative		Optimistic	
		$t_{par}(n)$	speedup	$t_{par}(n)$	speedup
c2670	9.9(16,K-MAFM)	2.47(8)	4.01	3.01(8)	3.29
c3540	13.0(8,K-MAFM)	3.19(8)	4.08	5.16(8)	2.52
c5315	15.1(16,K-MAFM)	4.44(16)	3.40	6.91(16)	2.18
c6288	14.8(16,K-MAFM)	3.1(16)	4.77	6.20(16)	2.39

Table 6. Speed-up w.r.t. the Best Sequential Execution Time

Transactions on Modeling and Computer Simulations

- Bagrodia R., and Liao W. 1992. Transparent Optimizations for Optimistic Simulations *Proc. 1992 Winter Simulation Conference*
- Bagrodia R., and Liao W. 1990. Parallel Simulation of the Sharks World Problem *1990 Winter Simulation Conference*
- Briner J., Ellis J., and Kedem G. 1991. Breaking the Barrier of Parallel Simulation of Digital Systems *Proceedings of the ACM/IEEE Design Automation Conference*
- Chandy K. and Sherman R. 1989. The Conditional Event Approach to Distributed Simulation *Distributed Simulation Conference*
- Cong J. and Ding Y. 1993. On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping *Proc. 30th ACM/IEEE Design Automation Conference*
- Cong J., Li Z., and Bagrodia R. 1994. Acyclic Multi-Way Partitioning of Boolean Networks *Proc. ACM/IEEE 31st Design Automation Conference*
- Fiduccia C., and Mattheyses R. 1982. A Linear Time Heuristic for Improving Network Partitions *ACM/IEEE Design Automation Conference*
- Jefferson D. 1985. Virture Time *ACM Transactions on Programming Languages and Systems* 7, 3, page 404-4
- Jha V., and Bagrodia R. 1993. Transparent Implementations of Conservative Algorithms in Parallel Simulation Languages *Winter Simulation Conference*
- Jha V., and Bagrodia R. 1994. A Unified Framework for Conservative and Optimistic Distributed Simulation *8th Workshop on Parallel and Distributed Simulation*
- Misra J. 1986. Distributed-Discrete Event Simulation *ACM Computing Surveys* 18, 1, page 39-65
- Soule L. 1992. Parallel Logic Simulation: An Evaluation of Centralized-Time and Distributed-Time Algorithms *PhD thesis, Stanford University*
- Sporrer C. and Bauer H. 1993. Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits *7th Workshop on Parallel and Distributed Simulation*
- Su W. and Seitz C. 1989. Variants of the Chandy-Misra-Bryant Distributed Simulation Algorithm *1989 Simulation Multiconference: Distributed Simulation*
- Wong K. 1986. Statistics on Logic Simulation *Proceedings of ACM/IEEE Design Automation Conference*

AUTHOR BIOGRAPHIES

RAJIVE BAGRODIA is an Associate Professor of Computer Science at UCLA. He obtained his M.A. and Ph.D. degrees in Computer Science from the University of Texas at Austin, in 1983 and 1987 respectively. His research interests include distributed algorithms, parallel languages, programming methodology and performance evaluation. He has served as the Program Chair and the General Chair for the Workshop on Parallel and Distributed Simulation. He was selected as a Presidential Young Investigator by NSF in 1991. He is also the recipient of the 1992 TRW Outstanding Young Teacher award.

ZHENG LI received a B.S. in Computer Science from Fudan University, Shanghai, China in 1991. She is expecting her M.S. degree in Computer Science from UCLA, Fall, 1994. Her research interests include VLSI circuit partitioning and parallel simulation.

VIKAS JHA is a Ph.D. student in Computer Science at UCLA. He obtained a B.S. in Computer Science from Indian Institute of Technology, Delhi, in 1989, and an M.S. in Computer Science from UCLA in 1991. His research interests include parallel simulation and distributed algorithms.

YUAN CHEN received B.S. and M.S. degrees in Computer Science from National Taiwan University, Taiwan in 1987 and 1989, respectively. Currently, he is a Ph.D student in Computer Science at UCLA. His research interests are Distributed Discrete Event Simulation and Parallel Programming.

JASON CONG is an associate professor of Computer Science at UCLA. His research interests include computer-aided design of VLSI circuits, fault-tolerant design of VLSI systems, and design and analysis of efficient combinatorial and geometric He has served on the program committees of a number of VLSI CAD conferences. He was the chairman of the 4th ACM/SIGDA Physical Design Workshop. He received the NSF Research Initiation Award in 1991 and the NSF Young Investigator Award in 1993. He received the Northrop Corporation Outstanding Junior Faculty Research Award from the UCLA School of Engineering and Applied Sciences in 1993.