SIMD 컴퓨터상에서 효율적인 병렬처리 논리 시뮬레이션

정 연 모†

요 약

VLSI 회로의 복잡도 및 집적도가 증가함에 따라서 이들의 검증에 사용되는 논리 시뮬레이션을 위해서 시간이 많이 걸린다. 본 논문에서는 SIMD 병렬처리 컴퓨터 상에서 빠른 논리 시뮬레이션의 구현을 위한 병렬처리 기법, 자료구조, 알고리듬을 제시한다. 대표적인 병렬처리 컴퓨터인 CM-2상에서 수행한 결과를 제시하고 이를 분석하고자 한다.

Efficient Parallel Logic Simulation on SIMD Computers

Yunmo Chung †

ABSTRACT

As the complexity of VLSI circuits has increased, a lot of simulation time for verifying their correctness has been required. This paper presents efficient parallel logic simulation protocols, data structures, algorithms to implement fast logic simulation on SIMD parallel processing computers. The performance results of the presented schemes on CM-2 are given and analyzed.

1. Introduction

This paper presents efficient simulation paradigms and data structures for parallel logic simulation based on the characteristics of gate-level logic simulation. The event-driven simulation protocols investigated are synchronous simulation, conservative simulation, and optimistic simulation. Some variations are considered to implement the simulation techniques efficiently on massively parallel SIMD(Single Instruction Multiple Data) computers.

Efficient event queue manipulations are critical on massively parallel SIMD computers since array access operations are, in general, slow. Circular FIFO (First-In-First-Out) lists are used as event queues in both synchronous and conservative simulation. As an efficient event queue structure for optimistic simulation, we present a circular binary search queue structure which allows binary search on a circular list. Advantage and disadvantages of the proposed technique will be discussed.

The logic simulation protocols were implemented in the CM-2 with 32K processors. Some ISCAS'85 and ISCAS'89 benchmark circuits are used as test circuits.

2. Related Work

A simple approach to parallel implementation of logic simulation is to use vector machines. However, it has been shown in [6, 17] that vectorization

† 정 회 원:경희대학교 전자공학과
논문접수:1995년 8월 29일, 심사완료:1996년 1월 5일

techniques for logic simulation can achieve very limited parallelism. Distributed event-driven simulation techniques offer the most promise.

In the well-known Chandy-Misra conservative simulation algorithm, deadlock management is a major problem in MIMD (Multiple Instruction Multiple Data) environments, and has been investigated in [9, 10, 17, 18]. Soule and Gupta [18] classified the types of deadlocks in digital circuit simulation to reduce deadlock occurrence, and reported simulation results on a Multimax. They observed that conservative simulation with null messages is inefficient on the MIMD machine. In [20], a new conservative simulation algorithm, called YADDES, is proposed, which uses a dataflow network to avoid deadlocks.

In Time Warp, several techniques have been proposed in [13, 15] to reduce the storage overhead and rollback problems. Moving Time Window (MTW) is proposed as a semi-optimistic approach to reduce the rollback frequency and the space overhead of Time Warp.

Comparisons of the performance of synchronous simulation, conservative simulation, and optimistic simulation techniques have been reported based on actual implementation on MIMD machines: Transporter-based multiprocessor with 8 nodes [16] and BBN Butterfly with 64 processors [11]. Fujimoto [11] reported that Time Warp outperforms the Chandy-Misra algorithm in his implementation. Lin et al., proposed a performance comparison model of parallel logic simulation in a MIMD environment. They showed that Time Warp always outperform the Chandy-Misra algorithms under the assumptions of zero overhead of rollback and state saving. It has been known that simulations using Time Warp on MIMD machines give promising performance in battlefield simulation [12], digital hardware simulation [3], and producer/consumer simulation workloads [1].

When optimistic simulation is implemented on MIMD machines, fast computation of Global Virtual Time (GVT) is very important, but it is not easy. Preiss suggests in [16] a token-ring calculation of GVT. Briner presents in [4] a GVT approximation algorithm using a tree of processors. Baldwin et al. present a GVT computation algorithm with overlapping window concept in [2].

A few results are available for parallel logic simulation in SIMD environments. Result have been published on parallel simulation of queuing models [14], circuit simulation [19], and switch-level simulation on the Connection Machine [5]. We have investigated logic simulation schemes on a massively parallel SIMD machine. Since queue structures for optimistic simulation on massively parallel SIMD machines are so important, we have developed several schemes as event queue structures. For example, a data parallel queue scheme was presented in [7]. In this scheme, an event is assigned to an event processor and event evaluation is done based on a data parallel approach. Next, a single queue scheme was presented which assigns an event queue to a gate in [8]. In the implementation, a variation of Time Warp was presented which uses a single queue at each process and computes a lower bound of rollback based on immediate cancellation to reduce space overhead. The analysis of local clock advancement in a SIMD environment, and the effect of moving time window size on execution time were also reported in [8]. In other words, each simulation technique can be characterized by how much each process is allowed to advance its simulation clock beyond the current global virtual time.

3. Basic Data Structures

As basic data structures for logic simulation on massively parallel SIMD machines, we consider memory layout, event queue schemes, and table lookup.

3.1 Memory Layout for Parallel Simulation

Without loss of generality, we can assume that one gate is assigned to one processor since massively par-

allel SIMD machines have a large number of physical processors and allow users to define as many virtual processors as possible within available memory capacity. A process (gate) contains the following information:

[memory layout]

- Local Virtual Time (LVT): the smallest simulation time of unprocessed events in the event queues of the process.
- Gate information: the information about the process, such as pointers to its successors.
- Function table: the table for computing output signals from input signal values.
- Input signals: the current input signals at the LVT.
- Input buffer: the buffer for storing just arrived events before putting them into its event queues.
- Event queues: the queues for storing all events to be processed.
- Other local variables.

In the above memory layout, other local variables include general temporary variables, such as variables for an active bit, an output signal, etc. as well as simulation technique-dependent variables, such as link clocks and minimum link clocks in conservative simulation. In addition, a global virtual time (GVT), the smallest timestamp of unprocessed events in all processes, is computed in the front-end processor.

3.2 Event Queue Assignments

In our simulation, a distributed event queue scheme is used in which each input port of a gate has its own event queue. In SIMD, the maximum queue size over all processors should be estimated and allocated in advance since dynamic allocation in not allowed, in general.

3.3 Table Lookup

To evaluate different function of all active gates at the same time, the table lookup method is used as in [7]. Each gate contains a table for the corresponding function. For example, if there are at most 3 inputs to each gate, 8 bits are enough to store each operation as shown in 〈Table 1〉. In the table, "x" represents "don't care" signal.

〈Table 1〉 Function tables for lookup operations

inbits(b_1 b_2 b_3)	000	001	010	011	100	101	110	111
3-input AND	0	0	0	0	0	0	0	1
3,2 or 1-input OR	0	1	1	1	1	1	1	1
2-input AND	0	0	0	1	x	x	x	x
1-iput AND	0	1	x	x	x	x	x	x
Invert	1	0	x	x	x	x	x	x

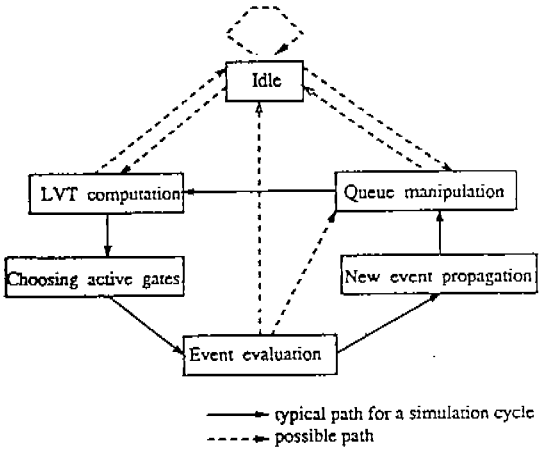
4. Parallel Logic Simulation Protocols

In this paper, several variations of distributed event-driven logic simulation protocols are considered. A logic circuit can be considered to be a directed graph in which nodes represent gates called processes, and arcs (links) indicate connections between the gates. Each gate propagates its output signals by sending event messages. Each event is time-stamped with the simulation time at which it should be executed.

Definition 1 *Simulation Cycle*: All processors on massively parallel SIMD machines are synchronized. Therefore, each process repeats the same procedure, which consists of LVT computation, choosing active gates, event evaluation, new event propagation, and queue manipulation. Optimistic simulation may require an additional step for rollback management. The time period to perform this procedure is called a *simulation cycle*.

The diagram for possible state changes of a process is given in (Fig. 1). The idle state in the figure represents that a process is not involved in any of the

above activities. Each simulation cycle is synchronized in SIMD environments.



(Fig. 1) Diagrams for possible state changes of a process

4.1 Distributed Synchronous Logic Simulation

Distributed synchronous simulation is a synchronous simulation with distributed event queue. In distributed synchronous logic simulation, each input port of a process receives events in non-decreasing timestamp order. A circular FIFO queue, rather than a priority queue, can be used as an event queue of each input port, facilitating queue manipulation. During every simulation cycle, the following algorithm is performed.

[Algorithm SYNCHRONOUS]

1. Each process computes its LVT.
2. GVT is computed by talking the minimum LVT of all processes.
3. Each process whose LVT is equal to the GVT performs event evaluation and event propagation.
4. Each receiving process performs event insertion.

4.2 Conservative Logic Simulation

In conservative simulation, a process executes events only if it is certain that no event with an earlier timestamp can arrive. Link clocks and minimum link clock are used as defined in conservative simulation. A link clock of each input port of a process is defined to be the timestamp of the most recently arrived event along that input link. The minimum link clock of a process is the minimum of link clocks of all its input ports.

Since the delay of each process is fixed and there is no preemption in logic simulation, events arrive at an input link in nondecreasing order of timestamps. A circular FIFO list is assigned to each input port of a process as an event queue. The following modified input/output waiting rules of the Chandy-Misra algorithm are proposed which exploit gate-level logic simulation properties: a process executes the events with timestamps equal to its LVT when its minimum link clock is grater than or equal to the LVT; the output events generated can be immediately sent to their destinations.

Null messages in Chandy-Misra algorithms are used to avoid deadlock. In contrast to MIMD environments, where null messages may significantly decrease the performance of Chandy-Misra algorithms [18], the null messages in SIMD environments can be efficiently used because all null messages are sent at the same time real events are propagated.

In addition, GVT can also be used to further reduce simulation cycles in conservative logic simulation. All evens whose timestamps are equal to GVT are executed even though the Chandy-Misra input waiting rule is not satisfied. On a SIMD machine, GVT can be computed easily.

The proposed algorithm exploits features of the SIMD architecture: easy computation of GVT and negligible overhead of null messages. For each simulation cycle, the following steps are performed for each process:

[Algorithm CONSERVATIVE]

1. Each process computes its LVT.

2. GVT is computed.
3. Each process executes the events with timestamps equal to its LVT when
   a) its minimum link clock is not less than LVT or
   b) the GVT is equal to LVT;
4. Each active gate performs event propagation. Each gate which has an updated minimum link clock and does not propagate any event sends a null message with the updated minimum link clock.
5. Each receiving process sets the corresponding link clock to the timestamp of the just arrived message along the link. If the message is non-null, event insertion is performed.

There are three things to be considered to obtain good performance when we implement conservative logic simulation using null messages and GVT. First, all processes whose minimum link clocks are updated by null messages or events need to propagate new null messages or events to update link clocks of its successors in time. Second, at every simulation cycle, all link clocks should be updated to GVT if they are smaller than GVT. Finally, null messages with infinite timestamps need to be sent at the end of each input vector to inform successors that no further events will be generated.

The proposed simulation technique can also be used for logic simulation on MIMD processing environments. Since GVT computation on MIMD machines is expensive, Step 2) and 3b) in the above algorithm might be discarded.

4.3 Optimistic Logic Simulation

Optimistic logic simulation uses LVT differently from conservative logic simulation. The difference is that all events whose timestamps equal LVT are executed in optimistic simulation, while the same events may not be executed in conservative simulation if the execution is not satisfied. Optimistic logic simulation does not need any link clocks. A simulation cycle of optimistic simulation performs the following steps:

[Algorithm OPTIMISTIC]

1. Each process computes its new LVT.
2. If the new LVT is not greater than its previous LVT, then rollback procedure, i.e. state restoration and cancellation, is performed.
3. Each process with unprocessed events performs event evaluation and propagation.
4. Each receiving process performs an event insertion operation.
5. Fossil collection is performed if necessary.

The following difficulties must be considered in implementing the optimistic simulation on a massively parallel SIMD machine. First, optimistic simulation has a storage overhead problem because information about all events whose timestamps are grater than GVT must be kept. Moreover, each process has three queues: input, output, and state queues. Second, processors of most massively parallel SIMD machines have relatively small local memory capacity. Finally, array access time on the CM-2, our target machine, is slow because its processors are bit-serial. Therefore, efficient storage management is very important.

4.3.1 Immediate Cancellation

Aggressive and lazy cancellation techniques have been proposed to undo incorrect event propagation. However, these are difficult to implement on massively parallel SIMD machines for following two reasons. First, there is not enough memory space for both state and output event queues. Second, additional simulation cycle are required to send antimessages, i. e. one simulation cycle for each antimessage.

To cope with the problems, we have proposed the "immediate cancellation" technique which eliminates use of antimessages and does not require both state and output queues. In this cancellation technique, a

... rollback immediately propagates a replace-... event to its successors. Immediate cancellation is ... variation of aggressive cancellation which sends antimessages for all incorrect event propagation. Since logic simulation has fixed event propagation routing and no preemption, one replacement event is enough to nullify all incorrect event propagation. With the immediate cancellation scheme, we can achieve significant reduction in space and make both queue manipulation and rollback management fast and easy.

4.3.2 Circular Binary Search Queue

The FIFO queue structure which has been used for both synchronous simulation and conservative simulation is not suitable for optimistic logic simulation because timestamps of events along each input link are not monotonic. Several existing queue structures for optimistic simulation have been suggested : a splay tree based on a self adjusting binary tree ; a timing wheel using a priority queue structure. In case of rollback, however, all the events in the queue may have to be searched to determine which to remove [4].

As an attempt to achieve fast queue manipulation, we use an event queue scheme, called CBSQ (Circuit Binary Search Queue). A CBSQ is a data structure which allows a binary search on a circular list. A CBSQ has two pointers, Front and Rear, to indicate events in non-decreasing order of timestamp contained in the queue. Front points to the element with the smallest timestamped event, while Rear points to the next available element in the queue. An algorithm for finding a key a CBSQ Q with n elements numbered 0..n-1 is described as follows :

[Algorithm CBSQ_search(key)]

if(Front < Rear) then perform binary search from Front to (Rear-1)
else if (Front > Rear) then
    if (key < Q(n-1)) then perform binary search from Front to (n-2)
    else if (key > Q(n-1)) then perform binary search from 0 to (Rear-1)
    else Found
else the queue is empty

A CBSQ is assigned to each port of a process as an event queue. In addition to the two pointers, another pointer called SimPtr is used to point to the unprocessed event which has the smallest timestamp and is ready to be evaluated for simulation. The CBSQ structure can be used for event queues since immediate cancellation, which does not need use of antimessages and state queues, is used.

The following basic operations can be done on CBSQs with O(log n)

- find an event with timestamp t,
- Insert an event into a queue.
- delete all events with timestamps greater than t,
- delete all events with timestamps less than t, and
- find the next event after the event pointed to by SimPtr.

5. Performance of Parallel Logic Simulation

In this paper, we present experimentation and performance of the logic simulation protocols presented. The logic simulation protocols were implemented on the CM-2 with 32K processors. Some ISCAS'85 and ISCAS'89 benchmark circuits are used as test circuits. The performance of the protocols are measured for 000 randomly generated input vectors.

5.1 Performance Metrics

As performance metrics, we use the number of simulation cycles, parallelism, maximum queue size, and execution times.

- Number of Simulation cycles

As defined previously, a simulation cycle is

synchronized for processors in SIMD. The execution time is proportional to the number of simulation cycles. That is, a large number of simulation cycles means slow speed. Based on the relationship between the number of simulation cycles and execution time, we can measure the average time taken per simulation cycle.

· Parallelism

The degree of parallelism (sometimes called activity level [18]) is the ratio of active processors to assigned processors at a given simulation cycle. An active gate at a simulation cycle is defined as a gate which has at least one event to be executed during the simulation cycle. As the number of active gates increases, the concurrency becomes higher. An active processor is defined as a processor which contains at least one active gate.

Let H be the number of assigned processors. To measure concurrency, we define degree of parallelism and parallelism as follows.

Definition 2 *Degree of parallelism* (D_i) at simulation cycle i is the ratio of the total number of active processors to the total number of assigned processors. That is,

$$D_i = C_i / H,$$

where C_i is the number of active processors at simulation cycle i.

When a processor simulates one gate only, the number of active gates is equal to the number of active processor.

Definition 3 *Parallelism* is the average ratio of the total number of active proccssors to the total number of assigned processors.

Parallelism can be computed as follows:

$$\text{Parallelism} = \frac{1}{S} \sum_{i=1}^{i=s} D_i$$

Where S is the number of simulation cycles.

· Maximum Queue Size

The maximum queue size is related to the memory requirement and speed. If there is a queue overflow, simulation cannot continue. In SIMD, the maximum queue size over all processors should be estimated and allocated in advance. In optimistic simulation using CBSQ data structures, fossil collection[3] is performed whenever there is at least one queue which exceeds a certain limit. The frequency of fossil collection significantly affects performance.

· Execution Times

Some results in this paper are not ideal (or not general) on massively parallel SIMD machines because some constraints of evaluation advancement were applied to limit queue size. In other words, if we had used a massively parallel SIMD machine with enough local memory size, different performance would have been obtained.

5.2 Performance Evaluation

Experimental results on the performance of the considered simulation protocols have been obtained for the benchmark circuits on the CM-2 with 32K processors. Although simulation with multi-delay can be done, a unit-delay was assigned to each gate for the consistency of experimental results. In the measurements, Moving Simulation Bound (MSB) is applied to both conservative simulation and optimistic simulation technique to prevent queue overflow[8].

For performance evaluation 1,000 input vectors were used. We gave 200 and 512 as timestamp intervals between successive input vectors for ISCAS'85 and ISCAS'89 benchmark circuits, respectively.

5.2.1 Number of Simulation Cycles

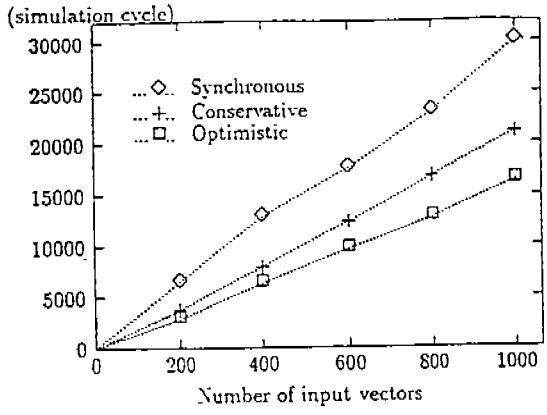
〈Table 2〉 shows the number of simulation cycles for 1,000 randomly generated input vectors. The MSB sizes to get the corresponding numbers of simulation cycles are also given in the table. According to the experimental results, synchronous simulation requires the largest number of simulation cycles while optimistic simulation needs the least. Synchronous simulation activates only the slowest processes whose LVTs are equal to GVT at each simulation cycle. Therefore, a lot of simulation cycles are needed because a relatively small fraction of gates are involved in event evaluation. On the other hand, optimistic simulation advances the local clock of each process as far as possible. If there is an event whose timestamp is equal to or less than LVT of a gate, the gate is rollbacked immediately without loss of any simulation cycles. When rollback does not occur, the process gets as much gain as possible. In conservative simulation, event evaluation at a gate is done based on information derived from its ancestors.

〈Table 2〉 Number of simulation cycles

Circuits	Synchronous	Conservative		Optimistic	
	Cycles	Cycles	MSB	Cycles	MSB
C1355	58117	6686	∞	6678	2000
C1908	68345	5675	28000	5717	20000
C6288	11736	67658	2500	67527	1500
C7552	83784	13304	10000	13844	7500
S9234	31886	21072	∞	16310	∞
S13207	41886	30297	∞	33568	40000
S15850	58874	22794	∞	1320	∞
S35932	26062	16044	∞	1550	∞

In combinational circuits, synchronous simulation has many more simulation cycles than the other two techniques, while the numbers of simulation cycles for conservative simulation and optimistic simulation are close. On the other hand, optimistic simulation for sequential circuit has fewer simulation cycles than the other two techniques. The reason is that event propagation is not frequently performed in sequential circuits since flip-flops control event flow. Flip-Flops execute received events according to the event scheduling policy of a technique, but they send messages only at clock times. In this case, link clocks cannot be properly updated in conservative simulation. For example, consider a gate which has a D flip-flop as its parent. In conservative simulation, the input link clock of the gate from the flip-flop is not updated in time because local clock computation must wait until the flip-flop sends a message to the gate. In optimistic simulation, the gate can execute event evaluation as soon as possible and go ahead with virtual time advancement.

(Fig. 2) shows the number of simulation cycles for S9234 as a function of the number of input vectors. According to our experimental results, in general, optimistic simulation has the smallest number of simulation cycles, while synchronous simulation has the largest number of simulation cycles.



(Fig. 2) Number of simulation cycles for S9234

5.2.2 Parallelism

〈Table 3〉 shows the parallelism for 1,000 randomly generated input vectors. Optimistic simulation has the highest parallelism since it includes unnecessarily

active gates which will be rolled back later. Synchronous logic simulation has the lowest parallelism. The reason is as follows. The time difference between successive input vectors is larger than the critical path length of the circuit being simulated. Since only the gates with the smallest LVTs are involved in event evaluation, any two successive vectors cannot be overlapped. In other words, input vectors are processed one by one. Therefore, parallelism is very low.

〈Table 3〉 Parallelism

Circuits	Synchronous	Conservative		Optimistic	
	Parallelism	Parallelims	MSB	Parallelism	MSB
C1355	0.0230	0.1900	∞	0.3240	20000
C1908	0.0230	0.2791	28000	0.3800	20000
C6288	0.0860	0.1976	2500	0.2600	1500
C7552	0.0160	0.1053	10000	0.1330	7500
S9234	0.0020	0.0033	∞	0.0240	∞
S13207	0.0018	0.0021	∞	0.0068	40000
S15850	0.0024	0.0027	∞	0.0490	∞
S35932	0.0140	0.0230	∞	0.2400	∞

Combinational circuits have higher parallelism than sequential circuits since flip-flops in sequential circuit control (or reduce) the flow of events. We can see, in the table, that parallelism becomes low if MSB is used.

5.2.3 Queue Sizes

〈Table 4〉 presents the required queue size of a gate at the possible maximum MSB size for each test circuit with different simulation techniques. The measurement of the maximum queue size considers all event queues during the entire simulation. The smallest queue size is required in synchronous simulation compared with the other two techniques since parallelism is low and simulation is done for input vectors one by one. On the other hand, optimistic simulation needs a large maximum queue size since events must be stored in the queue to copy with rollback. In sequential circuits, however, conservative simulation might need large queues than optimistic simulation since conservative simulation must wait to execute events due to a race event(or message) propagation.

In actual simulation, to avoid frequent fossil collection and improve performance, the following strategy can be used: fossil collection is performed if the queue size exceeds the limit. But the experimental results in Table 3.5 were obtained by applying fossil collection at each simulation cycle.

〈Table 4〉 Maximum queue sizes (in events)

Circuits	Synchronous	Conservative		Optimistic	
	Q Size	Q Size	MSB	Q Size	MSB
C1355	2	647	∞	346	20000
C1908	2	630	28000	473	20000
C6288	2	632	2500	419	1500
C7552	2	436	10000	431	7500
S9234	2	427	∞	337	∞
S13207	2	284	∞	434	40000
S15850	2	386	∞	258	∞
S35932	2	271	∞	289	∞

5.2.4 Execution times

〈Table 5〉 compares the execution times of parallel logic simulation techniques for different test circuits with 1,000 randomly generated input vectors. In the measurement, only the period for simulation is considered. The figures listed do not include the time required for translating circuit description, reading vector, or printing output.

We observed that conservative simulations are faster than synchronous and optimistic simulations for combinational circuits even though optimistic simulation requires fewer simulation cycle. One reason is that, each cycle of optimistic simulation involves more time-consuming operations, such as rollback and long queue manipulation. This observation contrasts

〈Table 5〉 Execution times(in seconds)

Circuits	Synchronous	Conservative		Optimistic	
	Q Size	Q Size	MSB	Q Size	MSB
C1355	214	55	∞	189	20000
C1908	265	41	28000	184	20000
C6288	735	437	2500	1667	1500
C7552	324	89	10000	402	7500
S9234	119	100	∞	443	∞
S13207	154	142	∞	814	40000
S15850	227	111	∞	39	∞
S35932	100	81	∞	37	∞

with the claims in [34, 48] that Time Wrap outperforms the Chandy-Misra algorithms. But, as shown in the experimental results, optimistic simulation may be better than any other techniques for sequential circuits. But, for circuit with fewer flip-flops, such as S9234, Time Wrap might be slow because optimistic logic simulation has only slightly fewer simulation cycles than conservative logic simulation. Even through each simulation cycle of synchronous simulation is very simple, it usually takes more time than the two techniques since it needs many more simulation cycles.

Soule and Gupta [18] found that deadlock avoidance with null messages on the Encore Multimax (shared memory MIMD machine with 16 needs) is highly inefficient. They claimed that actual run times range from 30 times slower to more than 100 times slower than conservative simulation with deadlock detection and recovery. But conservative simulation with deadlock detection and recovery is difficult to implement on SIMD machine. As we can from the table, conservative simulation with null messages works very fast.

6. Conclusions

This paper studied logic simulation as one application of distributed event-driven simulation on massively parallel SIMD processing machines. Data structures to implement several simulation techniques efficiently in SIMD environments were discussed. Some variations of distributed event-driven simulation have been proposed to improve performance of logic simulation.

We have experimentally analyzed the effect on performance of logic simulation depending on several factors, such as target machine used, simulation techniques applied, event queue structures implemented, and test circuit simulated.

We observed that, despite theoretical arguments to the contrary, optimistic simulation such as Time Wrap is not the best technique for all applications on massively parallel SIMD machines. This is attributed to its inherent rollback and queue management overhead. We also observed that, in contrast to MIMD environments, conservative simulation with null message works very fast on massively parallel SIMD machines. Finally, we conclude massively parallel SIMD machines can be efficiently used for parallel logic simulation if we utilize the limited local memory efficiently.

References

[1] [1] J. R. Agre. Simulations of time wrap distributed simulations. In Proceedings of the SCS Multiconference on Distributed Simulation, pages 85-90, March 1980.

[2] R. Baldwin, M. J. Chung, and Y. Chung. Overlapping window algorithm for computing GVT in Time Wrap. In Proceedings of the 11th International Conference on Distributed Computing Systems, pages 534-541. IEEE, May 1991.

[3] D. Ball and S. Hoyt. The adaptive Time Wrap concurrency control algorithm. In Proceedings of the SCS Multiconference on Distributed Simulation, page 174-177, January 1990.

[4] J. Briner. Parallel Mixed-Level Simulation of Digital Circuits Using Virtual Time. PhD thesis,

Duke University, 1990.

[5] R. E. Bryant. Data parallel switch-level simulation. In Proceedings of the 1988 International Conference on Computer Aided Design, pages 354-357, 1988.

[6] A. Chandak and J. C. Browne. Vectorization of discrete-event simulation. In Proceedings of the 1983 International Conference on Parallel Processing, pages 359-361, august 1983.

[7] M. J. Chung and Y. Chung. Data parallel simulation using Time Wrap on the Connection Machine. In Proceedings of the 26th Design Automation Conference, pages 98-103. ACM/IEEE, June 1989.

[8] M. J. Chung and Y. Chung. An experimental analysis of simulation clock advancement in parallel logic simulation on an SIMD machine. In Advances in Parallel and Distributed Simulation, volume 23, pages 125-132, January 1991.

[9] R. C. De Vries. Reducing null messages in Misra's distributed discrete event simulation model. IEEE Transactions on Software Engineering, 16 (1):82-91, January 1990.

[10] R. M. Fujimoto. Lookahead in parallel discrete event simulation. In Proceedings of 1988 International Conference on Parallel Processing, volume 3, pages 34-41, 1988.

[11] R. M. Fujimoto. Time Wrap on a shared memory multiprocessor. In Proceedings of 1989 International Conference on Parallel Processing, volume 3, pages 242-249, 1989.

[12] J. B. Gilmer. An assessment of Time Wrap parallel discrete event simulation algorithm performance. In Proceedings of the SCS Multiconference on Distributed Simulation, pages 45-49, July 1988.

[13] Y. Lin, E. D. Lazowaska, and M. L. Bailey. Comparing synchronization protocols for parallel logic-level simulation. In Proceedings of the 1990 international Conference on Parallel Proceedings, volume 3, pages 223-227, August 1990.

[14] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple-loop networks. Communications of the ACM, 32(1):111-123, January 1989.

[15] V. Madisetti, J. Walrand, and D. Messerschmitt. WOLF:A rollback algorithm for optimistic distributed simulation systems. In Proceedings of the 1988 Winter Simulation Conference, December 1988.

[16] B. R. Preiss. Performance of discrete event simulation on a multiprocessor using optimistic and conservative synchronization. In Proceedings of the 1990 International Conference on Parallel Processing, pages 218-222, August 1990.

[17] D. A. Reed, A. D. Malony, and B. D. McCredie. Parallel discrete event simulation using shared memory. IEEE Transaction on Software Engineering, 14(4):541-553, April 1988.

[18] L. Soule and A. Gupta. Characterization of parallelism and deadlocks in distributed digital logic simulation. In Proceedings of the 26th Design Automation Conference, pages 81-86. ACM/IEEE, June 1989.

[19] D. M. Webber and A. Sanggiovanni-Vincentelli. Circuit Simulation on the Connection Machine. In Proceedings of the 24th Design Automation Conference, pages 108-113. ACM/IEEE, June 1987.

[20] M. Yu, S. Ghosh, and E. DeBenedictis. A non-deadlocking conservative asynchronous distributed discrete event simulation algorithm. In Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, pages 39-43. ACM/IEEE/SCS, January 1991.

정 연 모

1980년 경북대학교 졸업(학사)
1982년 KAIST 졸업(석사)
1992년 미국 미시간주립대학교 졸업(박사)
1992년~현재 경희대학교 전자공학과 부교수
관심분야: CAD 및 VLSI, 병렬처리, 컴퓨터 구조