TR-646

Evaluation of Parallel Logic Simulation System using the Time Warp Mechanism

by

Y. Matsumoto & K. Taki

May, 1991

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Evaluation of Parallel Logic Simulation System using the Time Warp Mechanism

Yukinori Matsumoto and Kazuo Taki
Instisute for New Genaration Computer Technology

Abstract

This paper focuses on parallel logic simulation. An efficient logic simulation system on a large-scale multiprocessor is targeted. The Time Warp mechanism, an optimistic approach, was experimented and evaluated though it has been said that rollback processes costed much. The system is implemented on the Multi-PSI, a distributed memory multiprocessor. It includes several new ideas to enhance the performance, such as local message scheduler, antimessage reduction mechanism and load distribution scheme. In our experiment, using 64 processors, about 48-fold speedup was attained and the performance of the whole system amounted to about 60 k events/sec that is fairly good as a full software simulator. Then this paper reports the empirical comparison between the Time Warp mechanism and two conservative mechanisms: an asynchronous approach using null messages and a synchronous approach. The comparison shows that the Time Warp mechanism will be the most efficient of the three, and could be the most suitable for large-scale multiprocessors.

1 Introduction

Logic simulation is usually treated as a typical application of discrete event simulation. The time management mechanism is at the heart of problems of parallel discrete event simulation. The mechanisms broadly fall into two categories: conservative approaches and optimistic approaches.

Conservative approaches are classified into two groups: synchronous mechanisms and asynchronous mechanisms. Synchronous mechanisms need global synchronization which reduces the parallelism included in the problem by nature. For this reason, it is hard to apply to the highly parallel processing, especially on distributed memory multiprocessors[4]. On the other hand, the asynchronous conservative mechanisms can exploit much parallelism, but they tend to deadlock when circuits have loop structures. So a lot of computation power is needed to avoid it[2].

We are targeting an efficient logic simulation system on large-scale multiprocessors, most of which will be distributed memory machines. Optimistic mechanisms are basically asynchronous. However, they cannot deadlock, though they do spend some computation power on rollback processes [1, 8]. We expected the optimistic mechanism that can exploit much parallelism without paying significant cost for the rollback. That is, it could be the most suitable mechanism for logic simulation on large-scale multiprocessors.

We have examined a parallel logic simulation system using an optimistic approach, the Time Warp mechanism [1], on the Multi-PSI — a distributed memory multiprocessor, an experimental parallel inference machine [3]. Efficiency of the system and advantages of the Time Warp mechanism have been measured and evaluated.

This paper firstly overviews our system. New ideas to enhance the efficiency, such as local message scheduler, antimessage reduction mechanism and load distribution scheme are mentioned. Secondly, fairly good performance and speedup of the system in actual execution are reported. Finally we compare the performance with systems based on two other ways: the asynchronous conservative mechanism using null messages and the synchronous way. This comparison clearly shows that the Time Warp mechanism is more efficient than the conservative ways in the problem domain of logic simulation.

2 The Time Warp Mechanism

Event simulation can be modeled as that several objects change their states by communicating with each other. An object is a state-automaton. A message has information of an event whose occurrence time is stamped on the message (time-stamp).

Jefferson proposed the Virtual Time paradigm and its implementation, the Time Warp mechanism [1]. The Time Warp mechanism consists of the local control mechanism and the global control mechanism.

2.1 The Local Control Mechanism

In the Time Warp mechanism, assuming that messages might arrive chronologically, each object is usually executing and also recording the history of messages and states. But when a message arriving at an object in the wrong order, the object rewinds its history (this process is called rollback). Then it executes again from the time at which the message should have arrived. If there are messages which should not have been sent, the object also sends antimessages in order to cancel those messages.

2.2 The Global Control Mechanism

If all objects were to keep their full histories until the execution termination, an enormous amount of memory would be needed and the Time Warp mechanism would be infeasible.

The global control mechanism works in order to get GVT (global virtual time). The time GVT must satisfy the following two conditions.

1. GVT is not greater than the minimum of simulation time at any object.

2. GVT is not greater than the minimum of time-stamp values in the messages that have been sent but not yet received.

After the global control mechanism renews GVT, it notifies all objects of the new GVT. As no objects rewind their histories of the past before GVT, the memory area occupied by histories before GVT can be released.

3 System Overview and Implementations

3.1 Hardware and Language

This system is written in a concurrent logic language KL1 [9] on the Multi-PSI. The Multi-PSI[3] is a MIMD machine where 64 processing elements (PEs) are connected to each other by a 2-dimensional mesh network. A PE has a wide micro instruction architecture. The machine cycle time is 200 nsec.

The Multi-PSI is a distributed memory machine, so it costs rather a lot to access data in other PEs, but it is also easy to scale up. In the Multi-PSI, since the cost for inter-PE communication depends very slightly on the distance between PEs, we need not be sensitive to the connection topology.

KL1 supports the dynamic memory allocation mechanism and garbage collection mechanism similar to LISP. It allows programmers to be free from troublesome memory management (e.g. history recording area of the Time Warp mechanism).

3.2 System Specification

The system simulates combinatorial circuits and sequential circuits that have loop structures. It handles three values: Hi, Lo, and X (unknown). A different delay time can be assigned to each gate (non-unit delay model).

Functions are the minimum set for the experiment, but they can be easily expanded (handling more signal values, etc.).

3.3 Local Message Scheduler

In our system, a gate corresponds to an object that is mentioned in section 2. Since there are usually many more objects than PEs, each PE has to take charge of several objects. In this environment, the bigger time-stamp a message has, the more likely the message is to be rolled back. For this reason, message scheduling in each PE is expected to reduce rollback frequency effectively.

Our system has a message scheduler for each PE. When a message is spawned, the message is not sent to its destination object directly but is first sent to the scheduler which the destination object belongs to. The scheduler is a kind of sorter. It sorts messages according to their time-stamps, and sends the messages to their destination objects at an appropriate time.

The scheduler consists of slots that form a chain. Each slot corresponds to each unit of time in simulation. When the scheduler receives a message, it resisters the message in the slot corresponding to the time-stamp of the messages. The current time of a scheduler is defined as the smallest time-stamp among all the messages that are registered at that moment. The scheduler sends messages that have the time-stamps equal to the scheduler's current time.

3.4 Reduction of Antimessages

As long as we follow the original Time Warp mechanism proposed by Jefferson, when rollback occurs, as many antimessages must be generated as the number of messages that need be canceled (Figure 1).

However, the number of antimessages can be reduced when we assume the following condition.

Order-preserved Condition

- For any objects A and B, messages sent from A to B arrive at B in the same oder as they are sent by A.

Under this condition, Fukui proposed that a series of messages $M_1, M_2, ..M_n$ that must be canceled can be canceled with only one antimessage AM [7]. The basic idea is briefly described below.

Assume that $M_1, M_2, .., M_n$ are messages and AM is an antimessage. Also assume $M_1, M_2, .., M_n$ are all the messages that satisfy the following three conditions:

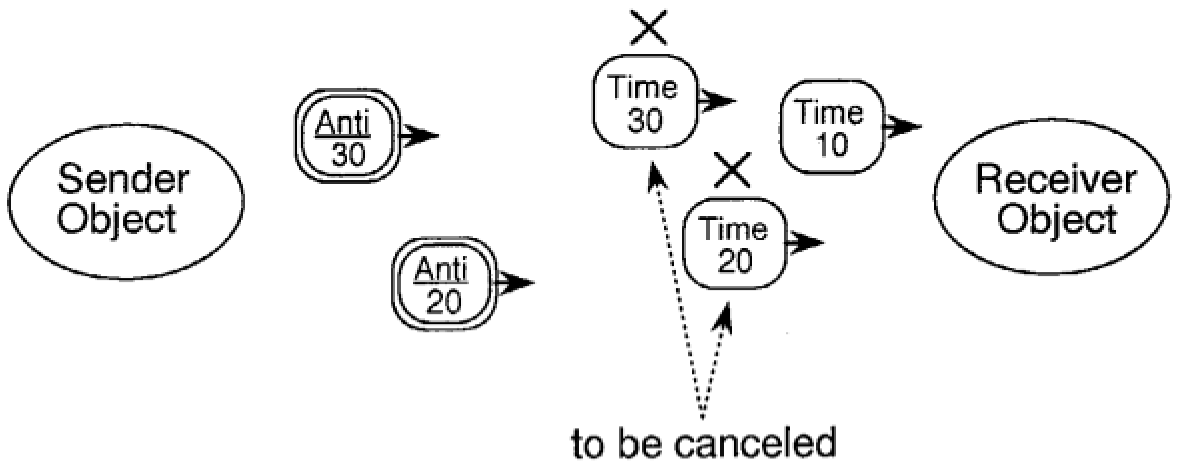

Figure 1: Sending antimessages

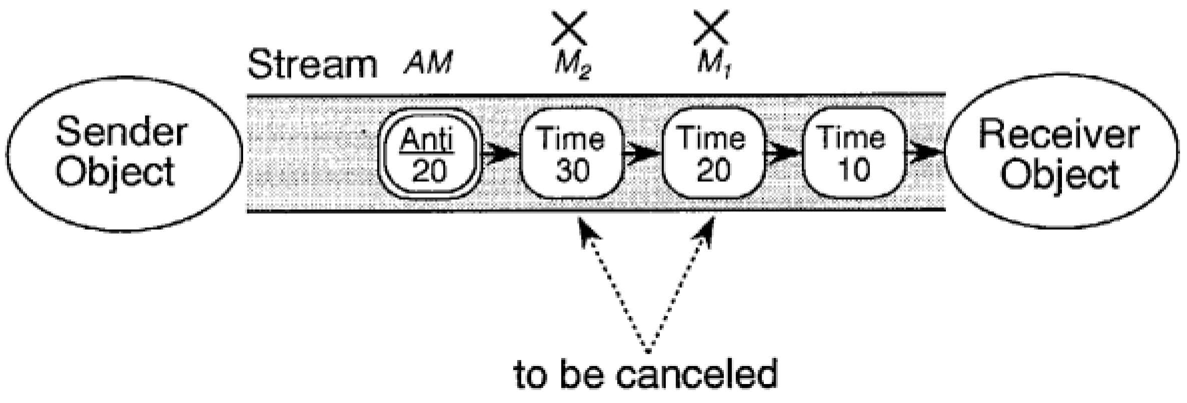

Figure 2: Reduction of antimessages(1)

- $M_1, M_2, .., M_n$ were sent before AM.
- $M_1, M_2, .., M_n$ were sent along the same channel that AM is sent along.
- $M_1, M_2, .., M_n$ have time-stamp values greater than or equal to AM.

Then it is clear that $M_1, M_2, .., M_n$ must be canceled. No other messages must be canceled. Only one antimessage that corresponds to the canceled message with the smallest time-stamp value need be sent (Figure 2).

We improved this idea further as described below. Assume that a sender object(SO) has to cancel messages $M_1, M_2, .., M_n$ that are already sent in this order, and at the same time SO knows that it will send a new message M_{new}, whose time stamp is equal to or less than that of M_1.

In this case, SO need not send any antimessages. Imagine that SO sends M_{new} but no antimessage. When a receiver object (RO) receives M_{new} with a smaller time-stamp than M_n that the destination object received just before, RO can easily notice that an invalid situation occurs. RO knows that $M_1, M_2, .., M_n$ must be canceled (Figure 3).

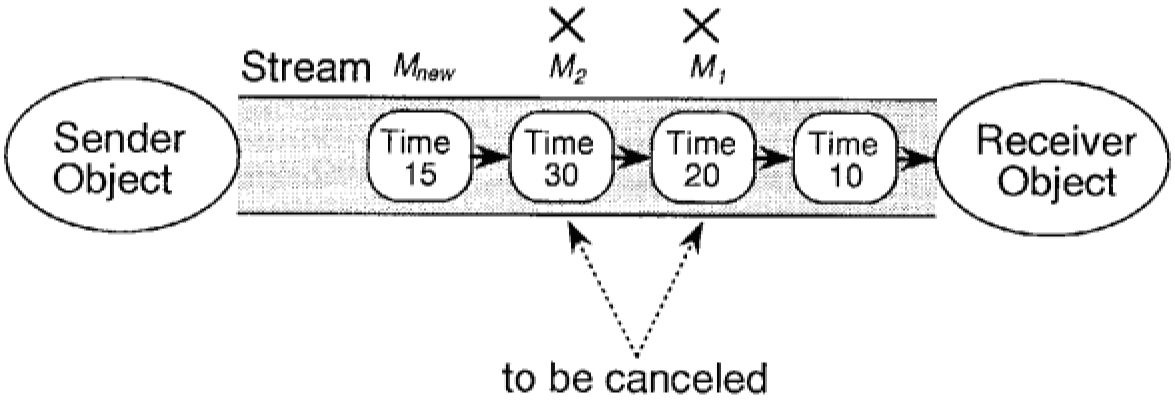

Figure 3: Reduction of antimessages(2)

Eventually objects can cancel the series of messages that must be canceled with only one antimessage, or sometimes without any antimessages, by checking the time-stamp order of messages when they arrive.

In our system, messages flow in the streams of KL1. KL1 keeps the order in which data were sent in the stream when the data are received. As our system satisfies the Order-preserved Condition, we adopted the above optimization for reducing antimessages.

3.5 Load Distribution Scheme

In parallel logic simulation, since the amount of work per message at an object is very small (the grain size of processing is very small), overhead of inter-PE communication cannot be ignored. Besides, as our system uses the Time Warp mechanism, rollback happens sometimes. The rollback procedure also may be one of the overheads.

Since rollback is basically caused by insufficient parallelism and load imbalancing, in partitioning process a given circuit network, we have to consider the following three points so that simulation will be executed efficiently on a distributed memory machine.

1. Reducing inter-PE communication
2. Deriving a lot of parallelism
3. Load balancing

For getting an optimal solution, we should define an evaluation function that expresses the above three requirements adequately. Then we have to find a partitioning solution that attains the best function value. Since such a problem is NP hard, some heuristic algorithms are required.

We propose a new partitioning strategy named "the Cascading-Oriented Partitioning" (in short COP). This strategy is expected to satisfy the above three points tolerably. It is also expected to require only a little computation time.

Generally speaking, circuits have sufficient tree parallelism of the multiple fanouts of gates. And COP intends to exploit parallelism of the multiple fanouts. Besides, this algorithm guarantees that

1. *Select gates connected to input terminals of the circuit. Those gates will be the start points of grouping.*
2. *Form the gates into a queue*
3. *Choose a new gate from the queue. If the gate does not belong to any clusters, the gate is called the current gate and will be used as a start point. If the gate belong to a cluster, then go to 3. If there is no such gate in the queue, then finish.*
4. *Allocate some memory area ready to hold a new cluster*
5. *Assign the current gate to the memory area*
6. *List all gates that connect the output of the current gate, except the gates already included in other clusters. If there is no gate listed, go to 3.*
7. *Pick an arbitrary gate from the gates listed in 6, and regard that gate as the current gate.*
8. *Enqueue the rest of the gates as new start points of grouping, and go to 5.*

Figure 4: Algorithm of COP

Table 1: Target circuits

Circuits	s1494	s5378	s9234	s13207
No. of gates	683	3,853	6,965	11,965
No. of signal lines	1,490	6,588	10,957	19,983

a gate G has, at least, one adjacent gate in the same cluster where G is. This nature contributes reducing inter-PE communication.

The algorithm is shown in Figure 4. The time complexity of this algorithm is apparently $O(n)$, where n is the number of gates.

After the partitioning process is finished, there may be small clusters that contain very few gates. They should be merged into adjacent large clusters. Conversely, extremely long cascade-formed clusters should be cut into several small clusters so that they do not cause load imbalancing.

Finally clusters are assigned to PEs randomly; the only constraint is that each processor should contain a roughly equal number of gates. This process is expected to attain load balancing.

4 Measurement Results and Discussions

We executed several experimental simulations on the Multi-PSI. Sequential circuits s1494, s5378, s9234 and s13207 were simulated in our experiment. These circuit data were published by ISCAS'89. The number of gates and signal lines in these circuits are written in Table 1.

In our experiments, the clock cycle was fixed to be a constant value. Signals to the other input

Table 2: Performance using the Time Warp mechanism (events/sec)

PEs	s1494	s5378	s9234	s13207
1	1,460	1,410	1,313	1,246
2	2,240	2,624	2,642	2,570
4	3,613	4,929	4,715	5,424
8	5,702	8,731	7,605	10,753
16	7,498	16,024	11,294	20,385
32	8,857	25,210	13,958	36,930
64	8,980	40,034	21,133	**59,974**

terminals were given randomly. They changed synchronously with the clock rising time.

In the following subsections, we firstly report the system performance and speedup. We make some comments on them. Secondly we report the frequency and cost of rollback. Finally we report the frequency and cost of inter-PE communication.

4.1 Performance and Speedup

Table 2 shows the system performance when the circuits were simulated using various numbers of PEs. Figure 5 indicates speedup.

Our system showed good performance and speedup in certain cases like s13207 and s5378. The best case was about 60 k events/sec performance and about 48-fold speedup using 64 PEs.

But in the case s1494, comparatively poor performance and speedup were measured. In order to ascertain the cause of the poor performance, we estimated the upper limit of speedup of each simulation problem, which was the parallelism of the problem, when the same load distribution scheme and the same input vectors were applied as used in the actual simulation. Assuming all the costs except the message processing, that is essential to the simulation, are negligible, the limit of the problem s1494 was only about 19-fold, whereas the limit of s13207 was about 43-fold. We can conclude that the cause of the poor performance is the lack of parallelism, and our system could show good performance as long as target problems have sufficient parallelism. The detail is written in [10].

Besides, we notice that super-linear speedup was obtained using 2, 4, 8 and 16 PEs when the simulation of s13207 was executed. The super-linear speedup is caused by the computation for releasing unnecessary history area after GVT is renewed. The detail is also described in [10].

4.2 Rollback

In this subsection, we focus on the experiment about s13207 using 64 PEs. We measured the frequency of rollback f_r and the average depth of rewinding history d_r. f_r and d_r are defined as follows.

$$f_r = R/E$$

$$d_r = H_r/R$$

where R is the number of rollback occurrences, E is the number of real events, and H_r is the number of messages that were rolled back. In our measurement, f_r was 0.0227 and f_d was 7.31.

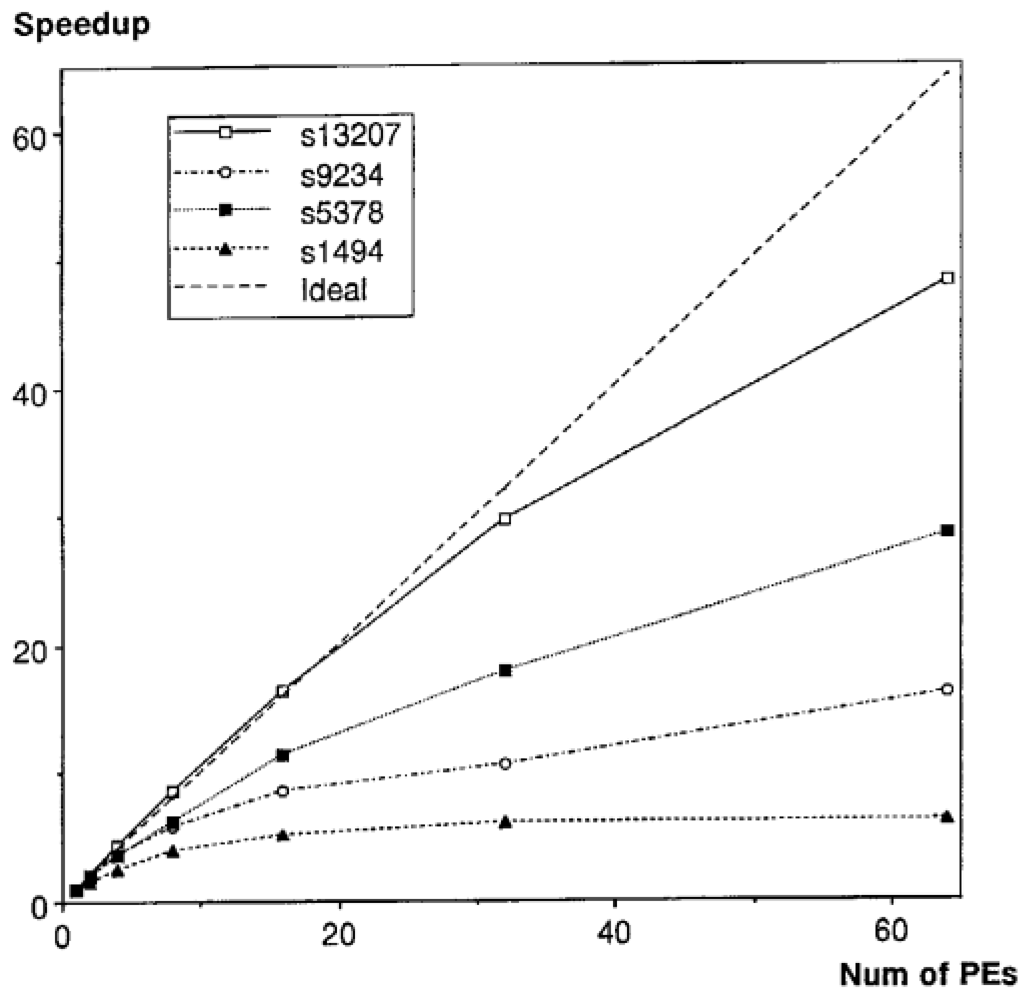

Figure 5: Speedup

The cost for one rollback process also attracts our interest. As the computation time for rollback t_r is roughly proportional to the number of messages rewound, t_r is represented by the next equation.

$$t_r = k_r h_r + C_r$$

where h_r is number of messages rewound in the history, C_r is a constant. Our measurement resulted that k_r was 0.027 msec, and C_r was 0.367 msec.

The average cost for rollback can be calculated using f_d and the above equation. It amounts to 0.564 msec. As the number of real events was 2,341,374, the average computation time for rollback per PE was only about 0.468 sec. On the other hand, the execution time using 64 PEs was 39.040 sec.

These results shows that rollback did not happen so often, and when it did, the cost for rollback was not seriously high. Note that rollback does not always affect the system performance badly. Only when a PE receives a message from the PE that is farthest behind does rollback affect the performance directly.

4.3 Inter-PE Communication

We measured f_c : the frequency of inter-PE communication, in case of s13207, using 64 PEs. We also measured the cost for inter-PE communication per message. f_c is defined as described below.

$$f_c = M_c / M_{all}$$

where M_c is the number of messages that crossed the processor boundaries, M_{all} is the number of all the messages generated in the simulation.

We got 0.256 msec as the cost for inter-PE communication per message and 0.0222 as f_c.

The value of f_c represents that our load distribution scheme worked efficiently and the frequency of inter-PE communication was very low. We can certainly say that the inter-PE communication cost had only a slight effect on the system performance in this experiment.

5 Empirical Comparison between the Time Warp Mechanism and Conservative Ways

We constructed two other logic simulation systems: one is based on an asynchronous conservative mechanism using null messages and the other is based on a synchronous mechanism. In this section, we report the performances of the two systems and compare them with the system that uses the Time Warp mechanism.

5.1 Asynchronous Conservative Mechanism

Asynchronous conservative mechanisms for time management have been proposed [2].

In asynchronous conservative mechanisms, deadlock is the most significant problem. Using null messages is one solution for avoiding deadlock. But then there arises another problem that a large number of null messages are generated.

Some mechanisms for reducing null messages have been proposed [2]. For example, consecutive null messages at a same gate input can be reduced into single null messages. We adopted this mechanism for the experimental implementation.

5.1.1 Results and Discussion

We simulated s13207 under the same conditions (load distribution, input vectors, etc.) as in Section 4.

Table 3 shows the system performance when the circuit s13207 was simulated using various numbers of PEs. Also the numbers of all messages and null messages are written in Table 3.

This system indicated good speedup but poor performance: using 64 PEs, only about 1.7 k events/sec performance was obtained. Table 3 denotes that most messages generated in the simulation were null messages. There is no doubt that too many null messages caused the poor performance.

Generally speaking, the average number of fanouts of a gate in circuits is more than 1. Without the mechanism for reducing null message, once an object receives a message, new messages are generated and sent to all the following objects. So the number of messages does increase exponentially as the simulation proceeds.

Table 3: Performance and number of messages using an asynchronous conservative mechanism

PEs	No. of all messages	No. of null messages	Performance (events/sec)
2	94,783,392	92,442,018	91
4	98,768,309	96,426,935	164
8	101,437,325	99,095,951	291
16	102,163,983	99,822,609	503
32	102,589,874	100,248,500	953
64	104,182,696	101,841,322	1,684

Table 4: Performance using a synchronous mechanism

PEs	Performance (events/sec)
1	2,261
2	4,210
4	8,307
8	13,756
16	19,748
32	18,493
64	11,320

Of course, in practice, the number of messages has an upper limit. Since the simulation time is discrete, the number of times at which events may occur in the simulation is finite. Note that two events never take place at the same time on the same signal line. So the limit is calculated by (Number of signal lines) $\times$ (Simulation span).

Our system has a mechanism for reducing null messages, but the number of messages amounted to 104,182,696 using 64 PEs. That is about half of the limit, 199,830,000 !

From the above consideration, it is apparent that logic simulation based on the asynchronous conservative mechanism using null messages is quite impractical unless an invention of such a mechanisms that can reduce null messages dramatically.

5.2 Synchronous Conservative Mechanism

We constructed another simulation system using a synchronous mechanism.

In the synchronous mechanism, a centralized time management mechanism called a time-wheel is used. The synchronous mechanism is efficient for single processor execution, but may be difficult to scale up to a system that uses many PEs. The way to parallelize the synchronous mechanism in our system is roughly described below.

Each PE has a time-wheel. The time-wheel manages gates assigned to the PE where the time-wheel is. The time-wheels communicate with each other at every tick in order to synchronize and advance their clocks one step further.

5.2.1 Results and Discussions

We simulated s13207, also under the same conditions as section 4, using the same load distribution scheme, inputting the same vectors. Table 4 shows the system performance when the circuit was simulated using various numbers of PEs.

This system indicated good performance using comparatively few PEs. In particular, when using 2 to 8 PEs, the performance was superior to using the Time Warp mechanism. But the performance peaked at 16 PEs.

Two things limited the performance.

1. Synchronization overhead

2. Low parallelism

Synchronization overhead is, as already mentioned, caused by the high frequency of synchronization operation: all time-wheels have to synchronize at every tick. With respect to parallelism, it is bounded by the number of events at the same clock. And hence the static load distribution scheme limits the parallelism much more than using the Time Warp mechanism.

6 Summaries and Conclusions

We examined the Time Warp mechanism, as an efficient time management mechanism for logic simulation on a large-scale multiprocessors. Implementation on the Multi-PSI, a distributed memory multiprocessor, showed that the system attained good performance and speedup: about 60 k events/sec and 48-fold speedup using 64 PEs. The performance obtained here is fairly good for a software logic simulator.

Also we compared this system with others that used the asynchronous conservative mechanism with null messages and the synchronous mechanism. The comparison showed that the Time Warp mechanism was the most efficient of the three when many PEs were used. We conclude that the Time Warp mechanism could be the most suitable for logic simulation on large-scale multiprocessors.

References

[1] D.R.Jefferson, "Virtual Time", ACM Transactions on Programming Languages and Systems,Vol.7,No.3, 1985, pp. 404-425

[2] J.Misra, "Distributed Discrete-Event Simulation", ACM Computing Surveys,Vol.18,No.1, 1986, pp. 39-64

[3] K. Taki, "The parallel software research and development tool: Multi-PSI system", Programming of Future Generation Computers, North-Holland, 1988, pp. 411-426

[4] L. Soule', T. Blank, "Parallel Logic Simulation on General Purpose Machines", Proceedings of 25th Design Automation Conference, 1988, pp. 166-170

[5] S. Shimogori, T. Kage "Parallel Logic Simulation using A Message-Driven Approach", IEICE Technical Report, CAS88-110, 1989, pp. 23-30 (in Japanese)

[6] B.D.Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks", Communications of the ACM, Vol.32, No.1, 1989, pp. 111-131

[7] S. Fukui, "Improvement of the Virtual Time Algorithm", Transactions of Information Processing Society of Japan, Vol.30,No.12, 1989, pp. 1547-1554 (in Japanese)

[8] R.M.Fujimoto, "Parallel Discrete Event Simulation", Communications of the ACM, Vol.33, No.10, 1990, pp. 30-53

[9] K. Ueda, T. Chikayama, "Design of the Kernel Language for the Parallel Inference Machine", The Computer Journal, Vol.33, No.6, 1990, pp. 494-500

[10] Y. Matsumoto, K. Taki, "Parallel Logic Simulation base on Virtual Time", Proceedings of Joint Symposium on Parallel Processing '91, 1991, (in Japanese) to appear