PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Parallel Logic Simulator based on Time Warp and its Evaluation

Yukinori Matsumoto and Kazuo Taki

Institute for New Generation Computer Technology
1-4-28, Mita, Minato-ku, Tokyo 108, Japan

yumatumo@icot.or.jp

Abstract

This paper focuses on parallel logic simulation. An efficient logic simulator on a large-scale multiprocessor is targeted. The Time Warp mechanism, an optimistic method for time-keeping, was experimented and evaluated.

Synchronous mechanisms and conservative mechanisms for time-keeping have been examined already, and their inefficiency on large-scale distributed memory machines has been noted. There have been few reports, however, on evaluation of the Time Warp mechanism although rollback processes have been presumed to be heavy. We aim at evaluating the efficiency of this mechanism. Several devices, such as a local message scheduler, an antimessage reduction mechanism and a load distribution scheme, are added in order to reduce rollback overhead.

The simulator is implemented on the Multi-PSI, a distributed memory multiprocessor. The simulator is written in concurrent logic language KL1. KL1 is expected to be suitable for parallel programming because it supports data-flow synchronization and global name space across the processor boundary.

Experiments were done so that the speedup, performance and influences of various overheads could be measured. Using 64 processors, 48-fold speedup and 99K events/sec performance were obtained. The overhead measurements revealed that rollback processes slightly affected performance. These results showed that the simulator had fairly good performance as a full-software logic simulator and that the Time Warp mechanism worked efficiently.

1 Introduction

Logic simulation is used in order to verify not only the functions of designed circuits but also the timing of signal propagation. Since logic simulation is currently one of the most time-consuming stages in LSI design, faster simulators are urgently needed. A parallel logic simulator is one likely way of producing quick simulation.

Parallel logic simulation is usually treated as a typical application of parallel discrete event simulation (PDES). In PDES, performance essentially depends on the time-keeping mechanism.

The mechanisms broadly fall into three categories: synchronous, conservative and optimistic mechanisms. Since synchronous mechanisms require global synchronization, they, apparently, do not work efficiently on distributed memory multiprocessors[Soulé and Blank 1988]. Furthermore, conservative mechanisms tend to deadlock when circuits have feedback loops. A lot of computation power is needed to avoid this[Lubachevsky 1989, Misra 1986, Shimogori and Kage 1989]. On the contrary, optimistic mechanisms cannot deadlock, however, they do expend some computation power on rollback processes [Fujimoto 1990, Jefferson 1985]. Only a few experiments with the optimistic mechanism were reported [Chung 1989, Briner *et al.* 1991] but the details have not been evaluated yet.

We are targeting an efficient logic simulator on large-scale MIMD multiprocessors, most of which will be distributed memory machines. We adopted the Time Warp mechanism, an optimistic mechanism, because the overheads of the mechanism were considered to be reduced using some devices such as a local message scheduler, an antimessage reduction mechanism and a load distribution scheme. By adding them to the Time Warp mechanism, we expected that it would become suitable for logic simulation on large-scale MIMD machines.

We have implemented a parallel logic simulator on the Multi-PSI[Taki 1988] — an experimental parallel inference machine, a distributed memory multiprocessor. The simulator was written in concurrent logic language KL1. KL1 provides several advantages for quickly programming parallel applications. Data-flow synchronization, global name space and dynamic memory allocation are expected to remove the causes of many bugs.

Several benchmark circuits have been simulated on the

simulator in order to evaluate the efficiency of the Time Warp mechanism. Performance, speedup, rollback overhead and inter-PE (processor element) communication overhead have been measured.

This paper firstly overviews our system. Remarkable devices to enhance efficiency, such as a load distribution scheme, a local message scheduler and an antimessage reduction mechanism are mentioned. Secondly, KL1 and the Multi-PSI are briefly introduced. Then, fairly good performance and speedup in actual execution are reported. Finally, we refer to the examination that revealed the main causes affecting performance in our simulator.

2 The Time Warp Mechanism

Event simulation can be modeled so 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[Jefferson 1985]. He suggested that the Time Warp mechanism would be useful as the time-keeping mechanism for PDES.

In the Time Warp mechanism, each object usually acts according to received messages and also records the history of messages and states, assuming that messages arrive chronologically. But when a message arrives at an object out of time-stamp order, the object rewinds its history (this process is called rollback), and makes adjustments as if the message had arrived in correct time-stamp order. After rollback, ordinary computation is resumed. If there are messages which should not have been sent, the object also sends antimessages in order to cancel those messages.

In addition to the above, a global control mechanism sometimes works to update GVT (global virtual time) which is used for memory management. GVT must satisfy the following two conditions.

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

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

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

3 System Overview

3.1 System Specification

The system simulates combinatorial circuits and sequential circuits that have feedback loops. It handles three values: Hi, Lo, and X (unknown). A different delay time can be assigned to each gate (non-unit delay model). Since this system only treats gates, flip-flops and other functional blocks should be completely decomposed into gates.

The supported functions are the minimum set for experiments, but they can be easily expanded (e.g. to handle more signal values).

3.2 Load Distribution Scheme

For efficient execution of parallel logic simulation on a distributed memory machine, the scheme of load distribution is important at the following three points: load balancing, keeping inter-PE communication frequency low and deriving a lot of parallelism.

In our simulator, target circuits are partitioned statically in the preprocessing phase. We propose a new partitioning strategy called "Cascading-Oriented Partitioning" (COP, for short) for high-quality load distribution.

COP makes several clusters by grouping gates that are connected to each other in a cascade form. A grouping operation starts from the primary input of the circuit. By tracing a path of the gate connection straightforward, subsequent gates are included in a cluster. If there are several candidates to be included, only one gate is selected and the others are left to be the starting points for other grouping operations.

After partitioning, small clusters that contain very few gates are merged into adjacent large clusters. Conversely, extremely long cascade-formed clusters are cut into several smaller 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.

COP intends to exploit parallelism of the multiple fanouts. COP also guarantees that a gate has at least one adjacent gate in the same cluster. So, COP is effective in keeping the communication locality, that is, reducing inter-PE communication. The random distribution of clusters attains load balancing.

In COP, the smaller each cluster, the better load balancing but the higher inter-PE communication frequency. There is a performance trade-off between good load balancing and the low frequency of inter-PE communication. It is necessary to decide the appropriate size of a cluster according to the number of gates in the

target circuit and the number of PEs[1].

COP may look similar to Agrawal's algorithm, however, COP is different from it in the next two points.

- Agrawal's algorithm basically assumes simulation using the synchronous time-keeping mechanism. According to the gate delay value, the algorithm estimates the number of messages generated in each cluster at each tick for the purpose of load balancing.

  Conversely, an estimation such as the above is slightly beneficial to our simulator because messages with different time-stamps can be evaluated simultaneously in the Time Warp mechanism.

- In Agrawal's algorithm, once a cluster is generated, it will never be decomposed into smaller ones. Therefore, the load is sometimes imbalanced.

  In COP, however, clusters that are too large are cut into several adequately sized clusters. This enables the system not only to be flexible for various numbers of PEs but also it to exploit more parallelism (i.e. pipeline parallelism).

3.3 Local Message Scheduler

In the simulation, there are usually several messages to be evaluated in a PE. When the Time Warp mechanism is used, the bigger time-stamp a message has, the more likely the message is to be rewound. For this reason, proper message scheduling in each PE is expected to reduce rollback effectively.

In our system, a message scheduler resides in each PE. When a message is spawned, it is first registered in the scheduler in which the destination object belongs. The scheduler picks up the messages with the smallest time-stamps and sends them to destination objects at the appropriate moment.

This scheduler ensures that rollback never happens as long as an object is receiving messages from other objects in the same PE. Only messages sent from other PEs may cause rollback.

3.4 Reduction of Antimessages

In Jefferson's original Time Warp mechanism, when rollback occurs, as many antimessages must be generated as the number of messages that need to be canceled (Figure 1). However, the number of antimessages can be reduced when we assume the next condition: for any objects A and B, messages transmitted from A to B are received by B in the same order as they are sent by A (order-preserved condition)[Fukui 1989].

[1]For reference, clusters with 12 to 32 gates are generated for a circuit consisting of 12,000 gates in the simulation using 64 PEs.

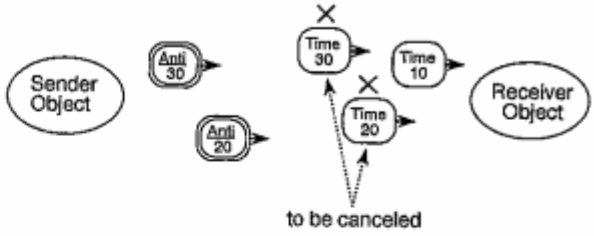

Figure 1: Cancellation with several antimessages

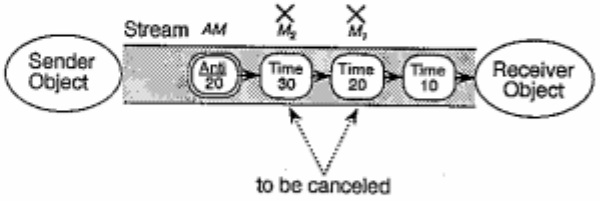

Figure 2: Cancellation with one antimessage

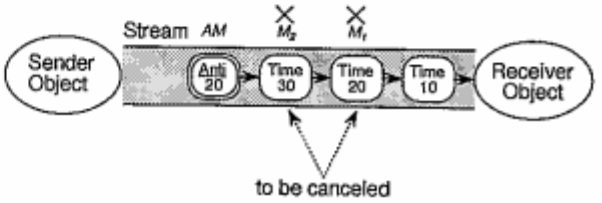

Figure 3: Cancellation with no antimessages

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

- $M_1, M_2, .., M_n$ 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-stamps 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 need be sent (Figure 2).

We advanced this idea one step further. Assume that a sender has to cancel messages $M_1, M_2, .., M_n$ which have already been sent in this order, and at the same time the sender knows that a new message M_{new} will be sent whose time-stamp is equal to or less than that of M_1.

In this case, the sender sends M_{new} but no antimessage. When a receiver receives M_{new} with a smaller time-stamp than the M_n that the destination object received just before, the receiver can easily notice that an invalid situation has occurred, and can cancel $M_1, M_2, .., M_n$ immediately (Figure 3).

In our system, the message streams of KL1 are used for communication between objects. Since KL1 keeps the order of messages in the stream, the order-preserved

condition is satisfied. So, we adopted the above optimization for reducing antimessages.

4 Hardware and Language

4.1 Hardware

This simulator is implemented on the Multi-PSI[Taki 1988], a distributed memory MIMD machine. The Multi-PSI consists of 64 processing elements (PEs) connected to each other by a 2-dimensional mesh network. A PE is a 40-bit (8 bits for tag and 32 bits for data) CISC processor controlled by horizontal micro-instruction. The cycle time is 200 nsec.

A network controller is paired with each PE, supporting message passing communication between PEs. The bandwidth of the controller is 5M bytes/sec. The network has wormhole routing functionality.

Since the Multi-PSI is a distributed memory machine, communication latency between objects in different PEs takes approximately twenty times longer than latency in the same PE. However, the distributed memory architecture can be scaled up easily.

4.2 Language and Implementation

This simulator is written in concurrent logic language KL1.

KL1 is a language without destructive value assignment to variables, that is a single assignment language. Due to its nature, data-flow synchronization is realized without significant overheads in the language implementation. Therefore, KL1 never compels programmers to describe synchronization explicitly at a primitive level.

On the other hand, a single assignment language tends to consume storage rapidly. A dynamic memory allocation mechanism and several garbage collection mechanisms are supported in the KL1 implementation. So, programmers are free from writing memory management.

The KL1 language assumes a system-wide (global) name space even on a distributed memory machine. In KL1 programming, first, a programmer writes only the logical concurrency, relations among concurrent objects or data-flow. Then, the programmer adds the "pragma" to specify object allocation to a certain processor, as below.

$$\ldots, B@processor(PE), \ldots$$

where B is a "goal" of KL1, which represents an object. Inter-PE reference pointers among objects or variables are maintained automatically by the KL1 language system at run time. Thus a programmer need not worry at all about the programming of inter-PE communication.

Since the characteristics described above eliminate the causes of many bugs, KL1 enables us to develop parallel programs much more easily than with the conventional languages (e.g. FORTRAN and C). In fact, it took one person just three months to complete the primary version of the simulator, including the circuit partitioning module! Moreover, because of KL1, several different experiments, which needed program modification, could be performed in a short period.

4.3 Avoiding Asynchronous Copying GC

As mentioned above, garbage collection (GC) is indispensable for KL1. Two kinds of garbage collection (GC) mechanisms, a copying GC[Baker 1978] and the MRB GC[Chikayama and Kimura 1987], are implemented for intra-PE memory management on the Multi-PSI.

For the Time Warp mechanism, the most important point in obtaining good performance is to keep the pace of simulation in each PE equal. However, the copying GC starts at different times in different PEs and disturbs the pace of simulation.

Fortunately, since the MRB GC collects single reference data area without stopping KL1 execution, it is expected to stabilize the pace of simulation. We took great care to keep the data reference single so that all data areas can be collected by the MRB GC. Hence we succeeded in preventing the copying GC[2].

5 Measurements and Discussions

Four sequential circuits, presented in ISCAS'89, were simulated on the Multi-PSI. The number of gates, average fan-ins and average fan-outs of the circuits are shown in Table 1. We measured system performance, speedup and overheads, such as rollback and inter-PE communication, in the experiments.

Table 2 shows the system performance for various numbers of PEs. Figure 4 indicates speedup. Table 3 shows the percentage of each process cost[3]. Table 4 shows the percentage of actual events[4] and rewound messages.

Table 5 shows the frequency of rollback f_r, the average depth of rollback d_r (i.e. the average number of rewound messages per rollback) and the frequency of inter-PE communication f_c. f_r is defined as R/E, d_r as H_r/R, and f_c as M_c/M_{all}, where R is the total number of rollback occurrences, E is the total number of actual events, H_r is the total number of rewound messages, M_c

[2]Consequently, when a certain circuit was simulated using 64 PEs, an improvement in performance of approximately 37% was attained compared to the case where the copying GC occurred(see appendix).

[3]These are the average values for 64 PEs.

[4]Actual events are the messages that are not rewound.

Table 1: Target circuits

Circuits	s1494	s5378	s9234	s13207
No. of gates	683	3,853	6,965	11,965
Avg. fan-ins	2.15	1.70	1.57	1.66
Avg. fan-outs	2.08	1.61	1.50	1.55

Table 2: Performance (events/sec)

PEs\Circuits	s1494	s5378	s9234	s13207
1	2,572	2,410	2,326	2,051
4	5,662	8,401	7,709	9,092
16	10,413	26,141	19,003	33,793
64	10,943	64,013	35,118	99,299

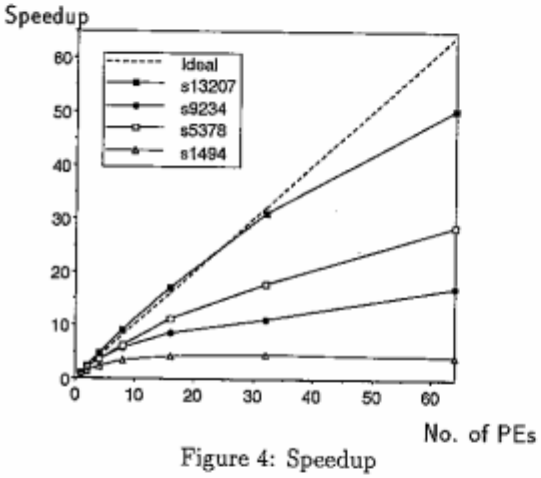

Figure 4: Speedup

is the total number of messages that are sent across PE boundaries and M_{all} is the total number of messages.

5.1 Performance and Speedup

As shown in Table 2 and Figure 4, the simulator attained approximately 99K events/sec performance and 48-fold speedup in the best case using 64 processors. This performance is fairly good for a full-software logic simulator. This good speedup shows that the Time Warp mechanism works efficiently.

In some cases, however, comparatively poor speedup was measured. In order to ascertain the cause of limited speedup, we will discuss the inter-PE communication overhead, the rollback overhead and parallelism in the following subsections.

Table 3: Percentage of time for each process (64PEs)

Process\Circuits	s1494	s5378	s9234	s13207
Evaluating and scheduling messages†, etc.	72.28	80.28	58.69	85.79
Rollback	5.32	2.50	1.61	1.53
Inter-PE communication	13.13	8.24	4.38	2.12
GVT updating	1.21	0.48	0.62	0.64
History releasing	0.43	2.41	1.57	3.86
Idling	7.63	6.09	33.13	6.06

† This process is not only for actual events but also for messages rewound.

Table 4: Percentage of actual events and rewound messages (64PEs)

Circuits	s1494	s5378	s9234	s13207
Actual events	29.34	81.84	65.23	86.54
Rewound msgs.	70.66	18.16	34.77	13.46

Table 5: Frequency and depth of rollback, Frequency of inter-PE communication(64PEs)

Circuits	s1494	s5378	s9234	s13207
f_r	0.938	0.0703	0.0510	0.0227
d_r	2.57	3.16	10.46	6.84
f_c	0.340	0.110	0.0761	0.0203

5.2 Inter-PE Communication Overhead

The cost per message for inter-PE communication was measured to be 0.503 msec[5]. However, the average cost of the essential work, that is, evaluating and scheduling a message, was 0.362 msec. So, the inter-PE communication cost was not negligible[6].

Tables 3 and 5, however, show that both the frequency and the percentage of inter-PE communication processes were low in the cases of s13207 and s9234. This means that our strategy for partitioning circuits worked effectively and that inter-PE communication had only a slight effect on performance in these cases. Conversely, in the case of s1494, both the inter-PE communication frequency and the percentage of its process were high compared to other cases. s1494 has, on average, more fan-ins and fan-outs than the others (Table 1), and it tends to

[5]A message is 25 bytes of data

[6]However, the relative cost of inter-PE communication is far lower than systems where inter-PE communication is supported by the operating system.

Table 6: Modified speedup and parallelism

Circuits	s1494	s5378	s9234	s13207
Modified speedup	3.96	23.62	12.51	35.66
Parallelism	18.88	35.52	17.95	43.24

cause the high inter-PE communication overhead.

5.3 Rollback Overhead

Rollback frequency and its cost greatly attracted our interest. Table 5 shows that the rollback frequency is not as high as we formerly suspected, except for s1494.

The average cost per rollback, even for the highest case, s9234[7], amounted to 0.578 msec by our measurement. Since the time for essential processes was 0.362 msec, the rollback procedure is not extremely time-consuming compared to the essential works, and consequently the percentage of total rollback cost is not seriously high in itself, as shown in Table 3.

5.4 Parallelism

Parallelism suggests the upper limit of speedup. In practice, however, the actual speedup is usually different from the parallelism because of several overheads.

We estimated the parallelism of each problem, as below. We made another simulator to measure parallelism. In that simulator, all PEs work according to the global synchronization. Here, we call an interval between global synchronizations a "*time slot*". A PE evaluates only one message in a time slot. All output messages, if any, are also sent and registered to their destination schedulers within the time slot. When there is no message to be evaluated in a PE at a certain time slot, the PE simply idles. Assume that the simulation finishes after N synchronization, and that M actual events, which are the messages that are not rewound, are measured in the simulation. Here, we define the parallelism of the problem as M/N. The parallelism means the speedup in such an environment where the cost for non-essential processes, such as rollback and inter-PE communication, can be ignored.

On the other hand, to make clear the effect of the inter-PE communication overhead and the rollback overhead, we measured the cost of releasing an unnecessary history area, which causes super-linear speedup[Matsumoto and Taki 1991]. Then we removed its effect from the measured speedup and recalculated the speedup. We named the recalculated speedup "*modified speedup*".

Table 6 compares the modified speedup and the parallelism of each problem using 64 PEs. The gap between the modified speedup and the parallelism is caused by the inter-PE communication overhead and the rollback overhead. For s5378, s9234 and s13207, the parallelism is close to the modified speedup. This means that the limited speedup was caused by a lack of parallelism. We conclude that our system could show good speedup as long as target problems have sufficient parallelism. With respect to the exceptional case, s1494, as Table 4 shows, a considerable percentage of the messages are rewound. It is considered that the high percentage was caused indirectly by the high inter-PE communication overhead or high rollback overhead, and resulted in further suppression of speedup.

[7]The cost is considered roughly proportional to the depth of rollback, and s9234 has the largest depth.

6 Further Experiments

Since neither the inter-PE communication cost nor the rollback cost are negligibly small, both of these costs are considered to affect performance not only directly but also indirectly. However, it is difficult to separate their influences clearly.

In this section, we report on the experiments that aim at clarifying which affects performance more, the inter-PE communication cost or the rollback cost. We assumed the model described below and made a system for the experiments.

6.1 Model

We assume that the only processes that need costs are the rollback, the inter-PE communication and an essential process. Here, an essential process consists of a message evaluation work and a scheduling work. Any other processes, such as GVT updating and releasing unnecessary history area, do not need any costs at all. It is also assumed that the essential process cost is equal for any gates.

In our model, the inter-PE communication cost C_c decomposes into three factors as follows.

$$C_c = C_{p_s} + C_l + C_{p_r} \quad (1)$$

where C_{p_s} is the time consumed in the sender PE for composing a message packet, C_l is the time from when the message leaves the sender until it arrives at the receiver (latency), and C_{p_r} is the time taken by the receiver to decompose the message packet.

As the rollback cost, t_r, is roughly proportional to the number of rewound messages, t_r is represented by the next equation.

$$t_r = k_r h_r + C_r \quad (2)$$

where h_r is the number of messages rewound in the history and C_r is a constant.

In practice, Equations 1 and 2 give a fairly precise representation of these costs. With regard to C_{p_s} and C_{p_r}, they are approximately equal[Nakajima and Ichiyoshi 1990] on the Multi-PSI, while the latency is negligible even if messages are transmitted between the most distant PEs.

6.2 Experimental System

The experimental system is based on the simulator presented in the previous sections. By adding several dummy loads to the original simulator, the actual costs for rollback and inter-PE communication become negligible. Thus this system maintains its fidelity to the model as much as is possible.

In the system, the cost for an essential process is fixed to be heavy, whereas the rollback cost and the inter-PE communication cost are changeable.

6.3 Results

We performed two kinds of comparative examination, as below.

1. The inter-PE communication cost is fixed so that its relative value to the essential process cost can be kept the same as in the actual simulation. With respect to rollback, k_r and C_r in Equation (2) are varied but C_r/k_r is kept equal to that in the actual simulation.

2. The rollback cost is fixed so that its relative value to the essential process cost can be kept the same as in the actual simulation. The inter-PE communication cost is varied but the equality between C_{p_s} and C_{p_r} in Equation (1) is kept because they were approximately equal in the actual simulation.

We simulated s9234 and s1494. They involved approximately the same parallelism, whereas both the inter-PE communication frequency and the rollback frequency were very different.

Figures 5 and 6 show the results. In Figure 5, the X axis shows the relative value of $C_{p_s}+C_{p_r}$ compared to the essential cost. In Figure 6, the X axis shows the relative value of C_r. The Y axis shows the relative performance (by solid lines) and the relative amount of rewound messages (by broken lines) compared to those when both the inter-PE communication cost and the rollback cost are set to zero. The arrows indicate the the actual proportion points between these costs.

For both circuits, the higher the inter-PE communication cost, the worse the performance. This apparently shows that the inter-PE communication cost affected performance adversely. Interestingly, the relative amount of rewound messages increased with the higher inter-PE communication cost for s1494, while the curve of the amount is approximately flat for s9234. The difference in declination of the performance curves was, therefore, caused not only by the difference in the inter-PE communication frequency but also by the difference in the amount of rewound messages.

On the contrary, neither performance nor the amount of rewound messages varied remarkably even though the

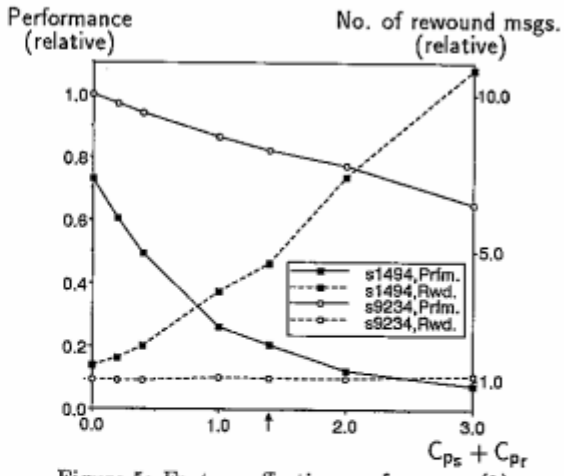

Figure 5: Factors affecting performance(1)

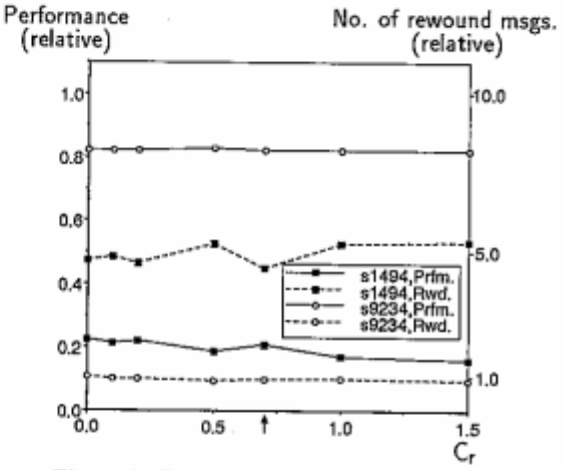

Figure 6: Factors affecting performance(2)

cost of rollback increased. This means that rollback cost did not have a dominant effect on performance in our system.

7 Summary and Conclusion

We constructed a parallel logic simulator on the Multi-PSI, a distributed memory multiprocessor. The simulator was programmed in concurrent logic language KL1. Since the causes of many bugs are essentially reduced by KL1, the simulator was able to be programmed in only three months by one person.

The Time Warp mechanism was adopted for time-keeping in the simulator. Since rollback overhead in a naive Time Warp mechanism was considered heavy, we added several devices such as a local message scheduler, an antimessage reduction mechanism and a load distribution scheme to reduce the overhead.

Several benchmark circuits were simulated on our system. Approximately 99K events/sec performance and 48-fold speedup were attained using 64 PEs. The per-

formance is fairly good for a software logic simulator. The good speedup shows that the Time Warp mechanism worked efficiently in the simulator.

We also examined the factors that are considered to affect performance adversely. The experiments revealed that the rollback overhead did not affect performance seriously in our system, while the inter-PE communication overhead decreased performance.

Acknowledgement

Valuable advice and suggestions were given by the members of PIC-WG, an ICOT working group, discussing parallel LSI-CAD. The authors gratefully thank them. Data for the evaluation of our system were recommended and given by Fujitsu Ltd. and Keio Univ. We also thank them.

References

[Agrawal 1986] P. Agrawal. Concurrency and Communication on Hardware Simulators. *IEEE Trans. on Computer-Aided Design*, Vol.CAD-5, No. 4 (1986), pp. 617–623.

[Baker 1978] H. G. Baker. List Processing in Real Time on a Serial Computer. *Communications of the ACM*, Vol. 21, No. 4 (1978), pp. 280–294.

[Briner *et al.* 1991] J. V .Briner *et al.* Parallel Mixed-level Simulation using Virtual Time. *CAD accelerators*, North-Holland, 1991. pp. 273–285.

[Chung 1989] M. J. Chung and Y. Chung. Data Parallel Simulation using Time-Warp on the Connection Machine. In *Proc. 26th ACM/IEEE Design Automation Conf.*, 1989. pp. 98–103.

[Chikayama and Kimura 1987] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proc. Fourth Int. Conf. on Logic Programming*, 1987. pp. 276–293.

[Fujimoto 1990] R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, Vol.33, No.10 (1990), pp. 30–53.

[Fukui 1989] A. Fukui. Improvement of the Virtual Time Algorithm. *Trans. of Information Processing Society of Japan*, Vol.30, No.12 (1989), pp. 1547–1554. (in Japanese)

[Jefferson 1985] D. R. Jefferson. Virtual Time. *ACM Trans. on Programming Languages and Systems*, Vol.7, No.3 (1985), pp. 404–425.

[Kudoh *et al.* 1991] T. Kudoh *et al.*. Parallel Logic Simulator for Shared Memory Multiprocessors. *IEICE Technical Report*, CPSY91-23 (1991), pp. 151–131. (in Japanese)

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

[Matsumoto and Taki 1991] Y. Matsumoto and K. Taki. Parallel Logic Simulation based on Virtual Time. In *Proc. Joint Symposium on Parallel Processing '91*, 1991. pp. 365–372. (in Japanese)

[Misra 1986] J. Misra. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, Vol.18, No.1 (1986), pp. 39–64.

[Nakajima *et al.* 1989] K. Nakajima *et al.*. Distributed Implementation of KL1 on the Multi-PSI/V2. In *Proc. 1989 Int. Conf. on Logic Programming* 1989. pp. 436–451.

[Nakajima and Ichiyoshi 1990] K. Nakajima and N. Ichiyoshi. Evaluation of Inter-processor Communication in the KL1 Implementation on the Multi-PSI. *ICOT Technical Report*, TR-531 (1990).

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

[Soulé and Blank 1988] L. Soulé and T. Blank. Parallel Logic Simulation on General Purpose Machines. In *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 166–170.

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

[Ueda and Chikayama 1990] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol.33, No.6 (1990), pp. 494–500.

Appendix

For the purpose of ascertaining the influence of the asynchronous copying GC, we made another simulator and compared it to the original. The difference between the comparative simulator and the original is as follows.

Original simulator :
Only the MRB GC works for collecting garbage.

Comparative simulator :
The copying GC happens asynchronously in each PE.

Table 7 compares the simulators when s13207 was simulated using 64 PEs. The result shows that asynchronous outbreaks of the copying GC in each PE increased both rollback frequency and rollback depth. It certainly caused the poor performance of the simulator.

Table 7: Influence of asynchronous copying GC

	Original simulator	Comparative simulator
Performance (K events/sec)	99.299	72.895
Frequency of rollback	0.0243	0.0261
Depth of rollback	7.96	11.684