タイムワープ機構の LSI-CAD への応用
Applications of the Time Warp Mechanism to LSI-CAD

松本 幸則
Yukinori Matsumoto

三洋電機 (株) 東京情報通信研究所
SANYO Electric Co., Ltd.

概要

第五世代コンピュータプロジェクトの一環として行なった、並列 LSI-CAD におけるタイムワープ機構適用の研究を総括する。タイムワープ機構は、主に並列イベントシミュレーションの時刻管理機構として用いられてきた。本研究では、論理シミュレーションのほか、全く新たな対象として LSI 配線を取り上げ、これらにタイムワープ機構を適用し、その有効性の評価を行なった。論理シミュレーションへの適用においては、a) ロールバックオーバヘッド削減技法提案とその評価、b) 他の時刻管理機構との比較、の二項目に焦点を当てた。具体的には、アンチメッセージ削減技法や適応時刻界 (AdaptiveTime-Ceiling) を提案し、その効果を評価した。また、同期的手法および保守的手法を用いた論理シミュレータとの性能比較を行ない、タイムワープ機構の優位性を示した。一方、並列 LSI 配線への適用では、タイムワープ機構を分散同期手法の一つとして位置付けた。そして、タイムワープ機構の組み込みにより、複数ネット同時配線時の領域競合が解決できること、および、既存の並列化手法と比べ、より効率的な配線処理を実現できることを実証した。

Abstract

This paper overviews the research of applying the Time Warp mechanism (TW) to LSI-CAD, which has been made during the Fifth Generation Computer Systems (FGCS) project. TW is a well-known technique for time-keeping in Parallel Discrete Event Simulation. In our research, not only logic simulation but also LSI routing — a new application of TW — were chosen as target problems. In the development of a logic simulator, an antimessage reduction mechanism and the Adaptive Time-Ceiling technique were proposed and embedded in the simulator. Evaluation showed that both techniques greatly reduced the rollback overhead. In addition, We compared TW with the synchronous and conservative mechanisms to show that TW is the most efficient one. On the other hand, by applying TW to LSI routing, we demonstrated the wide applicability of TW. TW not only solved the problem of conflicts between nets but also enhanced the routing speed more than a conventional parallel routing approach.

1 Introduction

As the progress of LSI integration, considerable amount of time is consumed in LSI design. Therefore, speeding up LSI-CAD tools is highly required. Parallelizing LSI-CAD tools is one of the most likely approach to acceleration.

In parallel processing, synchronization is the indispensable operation for guaranteeing correctness. In some problems, such as numerical analysis and image processing, sufficient parallelism can be extracted by mean of static analysis. In these applications, usually sophisticated synchronization techniques are not needed. In the other problems, however, only poor parallelism can be extracted statically. Moreover, naive and frequent synchronization tends to sacrifice efficiency. Many problems in LSI-CAD, such as logic simulation and LSI routing, belong to the latter class.

Logic simulation is a typical application of discrete event simulation. The time keeping mechanism, which is a kind of synchronization to control event execution order, is at the heart of problems in parallel discrete event simulation. The mechanisms broadly fall into three categories: synchronous, conservative and optimistic approaches[3].

Any clear conclusion about superiority among them was not given so far because each has individual shortcomings. Synchronous mechanisms have poor potential of exploiting parallelism. Only events with the same timestamp can be executed in parallel[12]. Conservative mechanisms tend to deadlock when circuits have feedback loops. A lot of computational power is needed to avoid this[9]. Optimistic mechanisms expend some computation power on rollback processes [4].

We, however, expected that the Time Warp mechanism, an optimistic technique, could be the most suitable for logic simulation by adding some new devices to reduce rollback overheads.

For efficient parallel LSI routing, high parallelism should be extracted not only by parallelizing the search phase but also by concurrent routing of multiple nets (terminals to be connected). In concurrent routing of multiple nets, the router must avoid conflicts between different nets. In other words, any two nets can not occupy the same space on a routing layer. This leads to a bottleneck and is currently the most serious problem in parallel LSI routing.

As mentioned in [4], the Time Warp mechanism can be regarded as a distributed synchronization technique. The key feature of the mechanism is that it exploits high parallelism in target problems with speculative computation, while a speculation error once occurs, it is corrected by rollback operations. We consider this feature of TW contributes to dynamic detection of the concurrently-routable nets.

Our research has been made based on the above consideration, focusing on the points shown below: 1: how to reduce the rollback overhead? 2: how is the advantage of the TW against the other time-keeping mechanisms? 3: how to apply TW to LSI routing?

This paper organized as follows: Section 2 overviews the Time Warp mechanism. Rough information on our hardware/software platform is given in Section 3. In Section 4, our experiments concerning logic simulation is shown. Also, experiments about LSI routing is reported in Section 5. Section 6 summarize this paper.

2 The Time Warp Mechanism

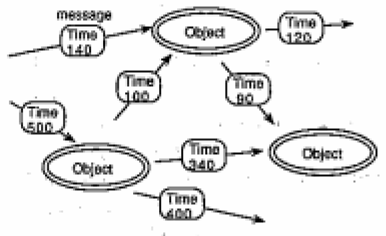

Figure 1: A problem model which the Time Warp mechanism targets

The Time Warp mechanism (TW)[4] is an optimistic technique for distributed synchronization, which preserves the cause-result relationship in a distributed manner.

Assume a model where several objects change their states by exchanging time-stamped messages (Figure 1). Also assume that there is a causality constraint that messages must be evaluated in timestamp order at each object.

In TW, an object optimistically assumes that messages will arrive in timestamp order. As long as they arrive chronologically, each object can correctly evaluate received messages while recording the history of messages and states.

In reality, however, sometimes messages may arrive at an object out of timestamp order. In such cases, the object rewinds its history (this operation is called rollback), and makes adjustments as if the messages had arrived in the correct order. If there are messages which should not have been sent but have already been sent, the object also sends anti-messages to cancel them. The rollback operation allows TW to remain within its constraint.

On the other hand, Global Virtual Time (GVT) should sometimes be updated for memory management. Details are presented in [4].

3 Machine and Programming Language

Our programs were written in concurrent logic language KL1 [16] and were implemented on the Parallel Inference Machine Multi-PSI and PIM/m[14, 10].

KL1 supports data-flow synchronization. This is a powerful feature for describing concurrent objects which act cooperatively to get a solution. In addition, a dynamic memory allocation mechanism and garbage collection mechanisms similar to LISP are supported. KL1 allows programmers to be free from troublesome memory management (e.g. the history management of TW).

The PIM/m is one of the target machines of the FGCS project, whereas the Multi-PSI is a prototype of PIM/m. Both are MIMD machines with distributed memory, where processing elements (PEs) are connected to each other by a 2-dimensional mesh network. For PIM/m, up to 256 PEs can be connected, while 64 PEs for the Multi-PSI.

4 Gate-level Logic Simulation

4.1 Specification

Our logic simulator 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 (nominal delay model). Functions are the minimum set for the experiment, but they can be expanded easily (handling more signal values, etc.).

4.2 Antimessage Reduction

In our simulator, the message streams of KL1, where the order of messages are maintained, are used. In this environment, an optimization of reducing antimessages can be applied as described below.

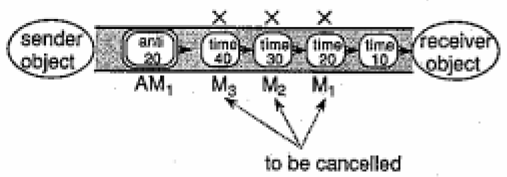

Figure 2: Reduction of antimessages

Figure 2 shows that object A sends several messages to object B. 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-stamp values greater than or equal to AM.

When B receives AM, obviously object B can know that $M_1, M_2, .., M_n$ must be cancelled but no other messages must be cancelled, by checking their senders, arriving order and timestamps.

4.3 Circuit Partitioning

For efficient parallel simulation on a distributed memory machine, a load distribution scheme is essential. We propose a new partitioning strategy called "Cascading-Oriented Partitioning" (or COP).

COP consists of two phases. In the first phase, cascade-formed clusters of gates are generated by tracing the gate connection straightforward from the primary input of the circuit. In the second phase, small clusters containing very few gates are merged into adjacent large clusters, whereas extremely large clusters are cut into several smaller clusters to avoid load imbalancing.

Clusters generated are finally assigned to PEs at random, while making each PE contain a roughly equal number of gates.

4.4 Evaluation of the Logic Simulator

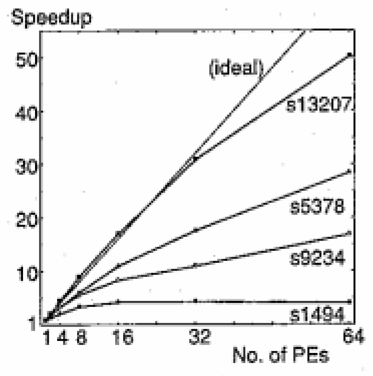

Figure 3: Speedup

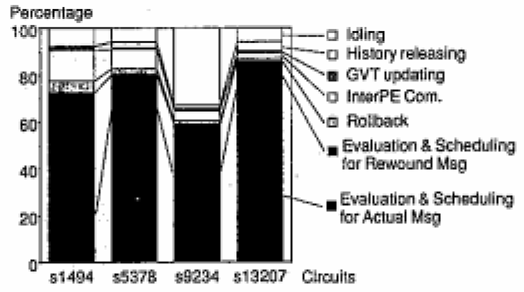

Figure 4: Percentage of time for each process (64PEs)

Four sequential circuits, presented in ISCAS'89[1], were simulated on the Multi-PSI. We measured the speedup, absolute performance and overheads, such as rollback and inter-PE communication.

Table 1: Parallelism and modified speedup (64PEs)

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

Table 2: Antimessage reduction ratio(64PEs)

Circuit	s1494	s5378	s9234	s13207
Reduction ratio	0.381	0.498	0.892	0.700

Figure 3 indicates speedup. Figure 4 shows the percentages of each process cost[1].

In the best case, using 64 PEs, our system attained approximately 100K events/sec performance and 50-fold speedup, a fairly good result. But in some cases, comparatively poor performance and speedup were measured.

4.4.1 Parallelism Analysis

In order to ascertain the cause of limited performance, we created an environment where the cost for nonessential processes, such as rollback and inter-PE communication, can be virtually ignored. We, then, measured the speedup of each problem for 64 PEs. We now call this speedup "*parallelism*". The *parallelism* suggests the upper limit of actual speedup of each problem.

Furthermore, we removed the cost of releasing the history area[2] from the actual execution time, and re-calculated the speedup. We named the re-calculated value "*modified speedup*". The parallelism and the modified speedup are compared in Table 1.

For the cases of s5378, s9234 and s13207, the parallelism is close to the modified speedup. This means that the cause of the poor performance was a lack of parallelism. We conclude that our system could show good performance as long as target problems have sufficient parallelism.

With respect to the exceptional case, s1494, as Figure 4 shows, a considerable percentage of the messages are rewound, and that causes further suppression of speedup.

4.4.2 Evaluation of Antimessage Reduction

We estimated the number of antimessages assuming that our antimessage reduction mechanism was not embedded, and compared it with the actual number of antimessages generated.

Table 2 shows the reduction ratio when 64 PEs are used. Here, the reduction ratio is defined as N_a/N_e, where N_a is the actual number of antimessage generated, N_e is the estimated number. The comparison

[1]These are the average values for 64 PEs.
[2]The cost causes super-linear speedup.

shows that our reduction mechanism worked very effectively.

4.5 Empirical Comparison with Other Mechanisms

For the purpose of comparing TW with others on the same machine, we further made two simulators; one uses the synchronous mechanism and the other uses the conservative mechanism.

4.5.1 Synchronous Mechanism

In the synchronous mechanism, a centralized time keeping mechanism, called a time-wheel, is used. In our synchronous simulator, each PE possesses a time-wheel. The time-wheel manages gates assigned to the same PE. All time-wheels synchronize globally to advance their clocks one step further.

4.5.2 Conservative Mechanism

In conservative mechanisms, deadlock is the most significant problem [9, 13]. Using null messages is one way of avoiding deadlock. Then, however, another problem arises wherein a large number of null messages are generated.

In order to suppress the rapid increase of null messages, we adopted a mechanism by which consecutive null messages at the same input of a gate can be reduced to a single null message.

4.5.3 Comparison Results

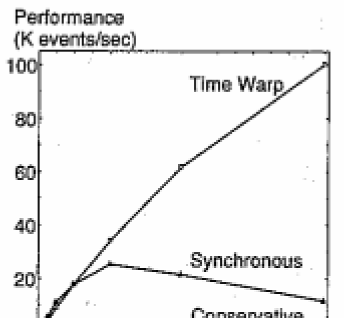

Figure 5: Performance Comparison (events/sec)

Figure 5 compares simulator performance when circuit s13207 was simulated under the same conditions (load distribution, input vectors, etc.).

The synchronous mechanism showed good performance using comparatively few PEs, however, the performance peaked at 16 PEs. Global synchronization at every tick apparently limits performance.

The conservative mechanism indicated good speedup but poor performance: using 64 PEs, only about 1.7 k events/sec performance was obtained. We measured the number of null messages generated during the simulation, and found that the number of null messages was 40 times as many as that of actual events! That definitely caused the poor performance.

This comparison substantiates that TW provides the most efficient simulation of the three mechanisms on distributed memory machine such as the Multi-PSI.

4.6 Adaptive Time-Ceiling

4.6.1 Time Window

In the original Time Warp mechanism, a message is eagerly evaluated even if its time-stamp is extremely large. Such a message, however, will probably be rewound later. Time Window[2, 11] is a technique to suppress the evaluation of messages with extremely large timestamp.

Although Section 4.4 reports that rollback overhead was not large, it may become serious problem when using more PEs. Therefore, Time Window is considered a non-trivial technique.

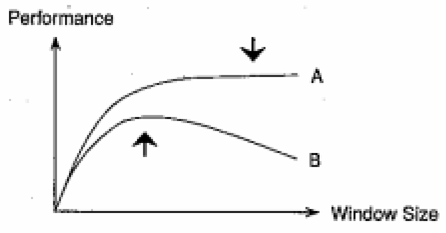

Figure 6: Time Window size vs. performance

In Time Window, how to know the optimum Time Window size is the most important but difficult question because the optimum size depends on target problems. Figure 6 is a rough sketch of the relationship between the Time Window size and simulator performance. In some applications, the relation is described by Curve A. There is no need to introduce the Time Window or, if used, the window size should be large enough, as indicated by a downward pointing arrow. In another application, however, the relation is shown by Curve B. The appropriate size should be at the point where there is an upward pointing arrow.

4.6.2 Strategy for Adaptive Decision

Here, we call the upper bound of Time Window *Time-Ceiling (TC)*. Also, we call the window size *TC height*.

Adaptive Time-Ceiling (ATC)[8] is a new mechanism which adjusts the TC height at runtime. The current performance of the simulator and the frequency of speculation error occurrence give useful information for adaptive changes of the height. A strategy for the adjustment of the TC height is shown below. Here, we assume that the TC height can vary within a set of discrete values[3]. An example of the table is shown in Figure 7. Also, we assume that the initial TC height is set to the minimum value.

[3]These values should be given in an exponential manner.

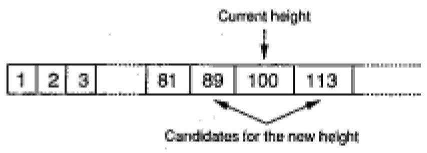

Figure 7: Table of Time-Ceiling heights (an example)

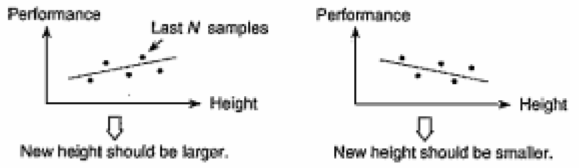

Figure 8: Adaptive decision of the Time-Ceiling height

The algorithm is:

```
IF $R_{err} > Threshold$ THEN
    The TC height is moved to the smaller side.
ELSE IF $K \geq 0$ THEN
    The TC height is moved to the larger side.
ELSE IF $K < 0$ THEN
    The TC height is moved to the smaller side.
```

where R_{err} is the current frequency of speculation error occurrence. Here, we define the ratio of rewound messages to non-rewound messages as the frequency of speculation error occurrence. K is the gradient of the least-squares approximation line for the last N samples (Figure 8).

4.6.3 Evaluation of ATC

We experimented with six sequential circuits of ISCAS'89 benchmarks on PIM/m using up to 256 PEs. Figure 9 shows the performance vs. the number of PEs.

Sometimes performance was fairly good even without ATC, and for these cases, ATC decreased performance a little. In other cases, ATC enhanced performance. We can see that the more processors that are available, the higher the effectiveness of ATC. In fact, in our experiments, ATC showed the greatest effectiveness, 37% improvement in performance on average, when using 256 processors. This is quite natural because the frequency of speculation error occurrence grows as the number of processors increases.

These results show that the ATC overhead is not so large, and ATC is, on average, effective for obtaining near-optimum performance.

5 LSI Detailed Router

5.1 Background

In order to exploit high parallelism in LSI routing, it is required to route multiple net in parallel. Conflicts between nets, however, must be avoided.

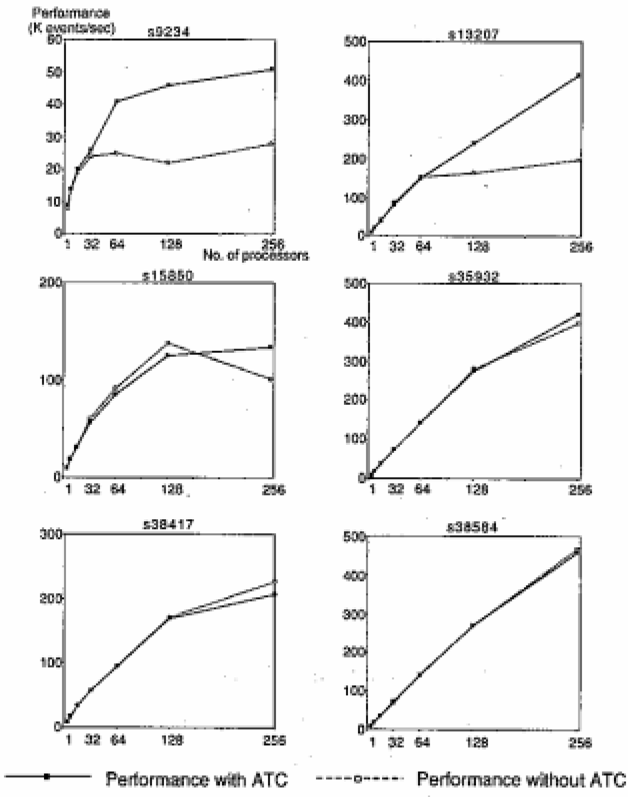

Figure 9: Effect of ATC on performance

A static analysis[18] detects some of nets which can be concurrently routed without conflicts, as shown in Figure 10. Here, each bounding box shows the global route. Detailed routing for each net is performed within the bounding box. In this example, Net A and Net B can be routed in parallel, because these do not overlap in terms of the bounding box. On the contrary, the bounding boxes of Net A and Net C overlap. Therefore, they can not routed concurrently.

However, consider the case shown in Figure 11, where the bounding boxes of Net X and Net Y overlap with each other. But after detailed routing, we can find that their connection paths do not conflict at all. This means Net X and Net Y could be routed concurrently in reality. Unfortunately, static analysis never extracts this sort of parallelism because we can not know the connection path before detailed routing.

Speculative computation, based on an optimistic assumption that Net X and Net Y could be routed concurrently, exploits this kind of parallelism *at runtime*. Of course, in some cases the assumption may be wrong and multiple nets conflict with each other, trying to occupy the same space. In this case, one of the conflicting nets (e.g., which has the highest priority) is permitted to occupy the space. Other nets are obliged to search the alternative paths again.

We expect the optimistic assumption is correct in many cases, and high parallelism can be exploited. The framework of TW makes for runtime parallelism ex-

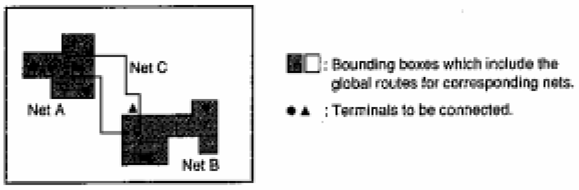
Net C
Net A
Net B
▨☐ : Bounding boxes which include the global routes for corresponding nets.
● ▲ : Terminals to be connected.

Figure 10: Examples of nets which can be routed concurrently in the static analysis approach

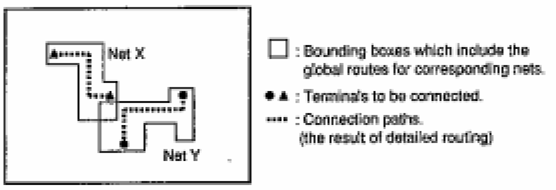
Net X
Net Y
☐ : Bounding boxes which include the global routes for corresponding nets.
● ▲ : Terminals to be connected.
···· : Connection paths. (the result of detailed routing)

Figure 11: Examples of nets which can be routed concurrently by runtime parallelism extraction

traction.

5.2 Basic Routing Algorithm

Now, we assume the following routing models:

- Manhattan wiring model on two layers
- Two-terminal net model

We adopted a rectangle-based algorithm[15] as the basic routing algorithm. In this algorithm, each layer is represented by a set of rectangular tiles as shown in Figure 12.

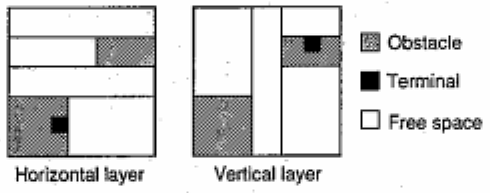

Figure 12: Free spaces represented by rectangular tiles

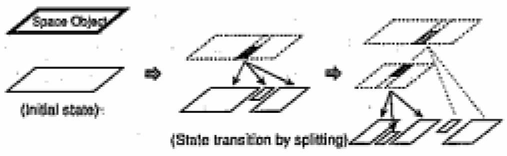

Figure 13: State transition of an object

5.3 Object-Oriented Modeling

Hereafter, we call our router *the Time Warp Router (TWR)*. For the ease of embedding TW, we modified the rectangle-based algorithm based on an object-oriented model.

In TWR, tiles are modeled as objects. Each object corresponds to either an terminal or a free space. We call the former object *a terminal object*, while the latter *a space object*. A space object has a state that represents the current free area within it. The initial state is the whole area of the object. As the routing process proceeds, some parts of the free area become occupied by nets. In this case, the state changes with splitting into several segments, regarding the occupied areas as new obstacles(Figure 13).

Objects communicate with each other to perform routing. *Search, Terminate* and *Traceback* messages are used for communication.

The routing algorithm is divided into two phases; the search phase and the state-update phase.

In the search phase, a *Search* message is sent from the start terminal object. Then *Search* messages are transmitted among objects according to A^* algorithm[5, 17]. If a connection path exists, the target terminal object receives a *Search* message. Then, the state-update phase starts. In this phase, first, a *Terminate* message is broadcast in order to stop transmitting unnecessary *Search* messages. Then, a *Traceback* message is sent from the target terminal object, tracing back the footmark of the received *Search* message. By doing so, the connection path of the net is determined in detail, and states of objects related to the path is updated.

For the purpose of performing the search phase according to the A^* algorithm, the cost function f is attached to each *Search* message. All *Search* messages must be controlled to be evaluated in the f order. In TWR, this control is realized by TW with the appropriate definition of time as described below.

5.4 Definition of Virtual Time

As mentioned in Section 2, TW controls the action of objects so that they evaluate messages in timestamp order. This means that the timestamp represents the processing order. Therefore, we define the key parameter *time* to make the routing process proceed in an appropriate order.

The desirable computation order is as follows.

- For different nets, the message evaluation related to the higher-priority net should precede the lower-priority net.
- For the same nets, *Search* messages should be evaluated in f order to follow the A^* algorithm.

From the above consideration, we let the tuple of the net number N and cost f, (N,f), be a timestamp and attach it to each *Search* message. For each *Terminate* and *Traceback* message, the timestamp of the corresponding *Search* message, just received by the target object, is copied and attached.

Let TS_i be a timestamp (N_i, f_i). Timestamps TS_l and TS_m can be compared according to the following rule.

```
IF        $N_l < N_m$                          THEN $TS_l < TS_m$
ELSE IF   $N_l = N_m$ and $f_l < f_m$          THEN $TS_l < TS_m$
ELSE IF   $N_l = N_m$ and $f_l = f_m$          THEN $TS_l = TS_m$
ELSE                                                $TS_l > TS_m$
```

In TWR, objects evaluate messages concurrently under the control of making messages be processed in the timestamp order. As a result, the obtained path is guaranteed to be the shortest. In addition, the routing results are the same as those when LSI routing is performed entirely in a sequential manner.

5.5 Evaluation of TWR

We tested TWR on the PIM/m with 64 PEs and evaluated it from two points of view; the advantage over the static analysis approach and the influence of rollback.

In our experiments, two industrial LSI data were examined. Both data assume two layers and virtual grids although TWR needs no grids. The net number was assigned statically, and a severe constraint concerning the path length was put on each net.

Table 3: Testing data

	DATA1	DATA2
Grid size	106 ×262	322×389
# nets	136	85
# blocks (vertical)	528	501
(horizontal)	528	440
# space objects	1,274	1,286

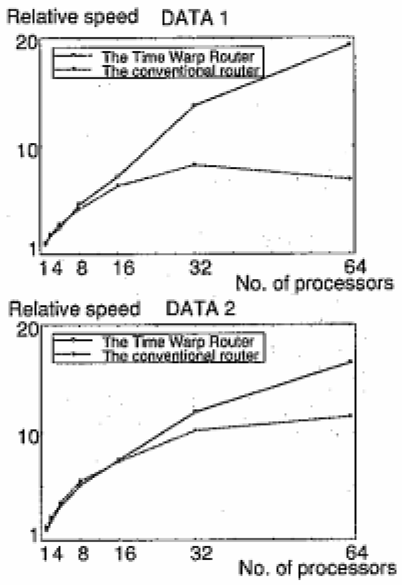

Figure 14: Comparison of relative routing speed between TWR and the conventional router

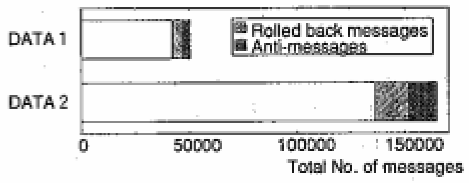

Figure 15: An analysis of the number of messages generated (using 64 PEs)

5.5.1 Advantage of TWR

In order to clarify the advantage of TWR, we made another parallel router which extracts parallelism by static analysis, and compared it with TWR. Hereafter we call this router *the conventional router*.

The conventional router is based on the same routing model and algorithm as TWR. So, the routing results is completely equal to that of TWR. The difference between these is that, in the conventional router, only the nets belonging to the same *routing unit* can be routed in parallel. Here, a routing unit is a group of nets which are permitted to be routed concurrently by static analysis. Note that termination detection is needed before the routing process of the next routing unit starts. Due to the path length constraint on each net, the search space is greatly limited. By using this information, the conventional router estimates the bounding box for each net and creates routing units, even without the global routing result.

Figure 14 compares the relative routing speed. Here, relative speed is defined as $T_{TWR}(1)/T_{Router}(N)$, where $T_{Router}(N)$ is the execution time of a *Router* (TWR or the conventional router) using N PEs. $T_{TWR}(1)$ is the execution time of TWR using one PE.

When using fewer PEs, the conventional router worked faster than TWR. This is because TWR contains overheads of recording history. Using larger number of PEs, however, TWR greatly surpassed the conventional router. For DATA1, the peak speed of TWR was 2.3 times faster than that of the conventional router. The cost of termination detection increases as more PEs are used. This causes the limited speedup of the conventional router. Thus, we conclude that TWR is more scalable, and, is very suitable for large-scale parallel machines.

5.5.2 Influence of Rollback

The influence of rollback greatly attracts our interest. We measured the number of messages related to rollback, that is, the number of anti-messages and rolled back messages. Figure 15 shows the number of messages related to rollback when using 64 PEs. This figure denotes that rollback influence was not very serious.

6 Summary

We examined the efficiency and applicability of the Time Warp mechanism to parallel LSI-CAD, especially to logic simulation and to LSI routing.

The antimessage reduction mechanism and sophisticated circuit partitioning were added to our parallel logic simulator. As the result, the simulator showed

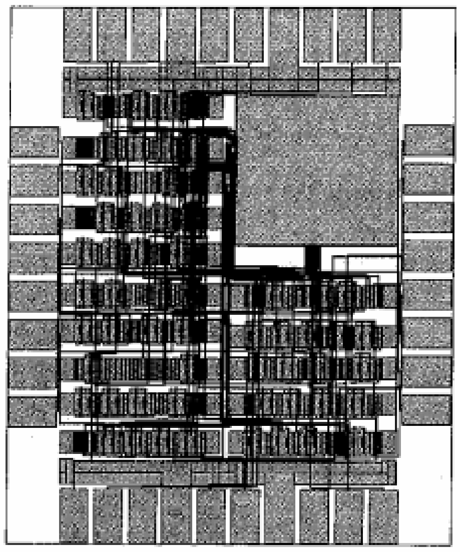

Figure 16: Routing Result (DATA2)

very good speedup of 50-fold on the Multi-PSI using 64 processors. Comparison with other time-keeping mechanism were made to reveal that the Time Warp mechanism is the most efficient.

For more processors, we proposed the Adaptive Time-Ceiling for reducing the rollback influence. In our measurement, embedding ATC led to 37% improvement in speed on average, using 256 processors.

We also showed a methodology of applying the Time Warp mechanism to LSI routing. This is the first step to showing the applicability of the Time Warp mechanism to a wide range of problems. The Time Warp mechanism not only solved the problem of conflicts between nets but also enhanced the routing speed more than a conventional parallel routing approach.

References

[1] Brglez, F. *et al* : "Combinational Profiles of Sequential Benchmark Circuits," *Int. Symp. on Circuits and Systems '89*, pp1929–1934 (1989)

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

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

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

[5] Kumar, V. *et al.* : "Parallel Best-First Search of State-Space Graphs: A Summary of Results," *Proc. AAAI'88*, pp. 122–127 (1988)

[6] Margarino, A. *et al.* : "A Tile-Expansion Router," *IEEE Trans. Computer-Aided Design*, Vol. CAD-6, No.4, pp. 507–517 (1987)

[7] Matsumoto, Y and Taki, K.: "Parallel Logic Simulation on a Distributed Memory Machine," *Proc. 1992 European Conf. on Design Automation*, pp. 76–80 (1992)

[8] Matsumoto, Y. and Taki, K.: "Adaptive Time-Ceiling for Efficient Parallel Discrete Event Simulation," *Proc. Western Multiconf. Comput. Simulation — Object-Oriented Simulation Conference—*, pp.101–106 (1993)

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

[10] Nakashima, H. *et al.* : "Architecture and Implementation of PIM/m," *Proc. Int. Conf. on Fifth Generation Computer Systems*, pp. 425–435 (1992)

[11] Sokol, L.M. *et al.* : "MTW: A strategy for scheduling discrete simulation events for concurrent execution," *SCS Multiconference on Distributed Simulation*, pp. 34–42 (1988)

[12] Soulé, L. and Blank, T. : "Parallel Logic Simulation on General Purpose Machines," *25th ACM/IEEE Design Automation Conference*, pp. 166–170 (1988)

[13] Soulé, L. and Gupta, A. : "Analysis of Parallelism and Deadlock in Distributed-Time Logic Simulation," *Stanford University Technical Report*, CSL-TR-89-378 (1989)

[14] Taki, K. : "Parallel Inference Machine PIM," *Proc. Int. Conf. on Fifth Generation Computer Systems*, pp. 50–72 (1992)

[15] Tsai, C.C., *et al.* : "An H-V Alternating Router," *IEEE Trans on CAD*, Vol.11, No.8, pp.976-991 (1992)

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

[17] Winston, P.H. : *Artificial Intelligence*, Addison Wesley (1984)

[18] Yamauchi, T. *et al.* : "PROTON: A Parallel Detailed Router on an MIMD Parallel Machine," *Proc. Int. Conf. Computer-Aided Design'91*, pp. 340–343 (1991)