ICOT Technical Report: TR-0830

TR-0830

Time Warp Router: A New Application of the Time Warp Mechanism

by
Y. Matsumoto & K. Taki

January, 1993

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

Time Warp Router:
A New Application of the Time Warp Mechanism

Yukinori MATSUMOTO and Kazuo TAKI

Abstract

This paper presents a new application of the Time Warp mechanism to LSI routing. An efficient parallel routing method for gridless wiring based on the Time Warp mechanism is proposed. The Time Warp mechanism can exploit high parallelism in target problems with speculative computation while maintaining essential sequentiality. This feature is advantageous to LSI routing: the mechanism enables not only maintainance of the routing order but also allows optimal solutions to be obtained while multiple nets are routed efficiently in parallel. Measurements were made on an MIMD computer with distributed memory and 14-fold speedup was attained using 64 processors.

1 Introduction

LSI routing is an important stage in LSI design, in which connection paths between terminals are searched. Since the routing stage is one of the most time-consuming stages, automatic routers have been developed on high performance computers. Nowadays, semiconductor process technology is progressing rapidly so that more highly integrated LSI chips can be produced. Therefore, faster routers with adequate flexibility are required to meet the progress made in LSI industry.

Using parallel computers is a promising approach to accelerate routers while retaining flexibility[1, 3, 14]. For this reason, we are aiming to build a parallel router that executes efficiently on large-scale parallel computers, most of which are MIMD machines with distributed memory.

Parallel routers should attain not only high speed routing but also high quality solutions. To attain high speed routing, high parallelism must be extracted from target problems. Concurrent routing of many nets[1] is considered capable of exploiting high parallelism. On the other hand, to obtain high quality solutions, the routing order should be maintained while routing. The reason for this is as follows. Assume that some of paths are already determined. Since these paths are obstacles to the nets still to be routed, the quality of routing solutions depends highly on the routing order. In general, the order is decided with great care in the preprocessing stage.

It seems difficult to route multiple nets efficiently in parallel while maintaining the given routing order. We, however, expected the Time Warp mechanism to overcome this difficulty. The Time Warp mechanism is a mechanism capable of exploiting high parallelism in target problems with speculative computation. When speculation errors occur, they are corrected by rollback operations. This mechanism has been applied mainly to parallel discrete event simulation and distributed database management. Its efficiency has been reported in regard to these applications[9], but the mechanism has not been applied to parallel routing yet.

In order to evaluate the effectiveness of the Time Warp mechanism in parallel routing, we developed a parallel router to which to apply the mechanism. The router was implemented on the Parallel Inference Machine PIM, which is a large-scale MIMD computer with distributed memory.

For the basic routing algorithm, our router employs a gridless routing algorithm based on rectangle partitioning[8, 10], although the Time Warp mechanism can be applied to any other routing algorithm. We chose this algorithm because the following aspects make it a promising method for the future.

- It enables smaller chips to be produced because there are no restrictions on the positions of terminals and paths.
- It treats hybrid circuits involving both digital and analogue circuits, where various widths of wires are required.

[1] A net is a terminal pair to be connected.

This paper is organized as follows. Section 2 presents the basic algorithm for gridless routing. In Section 3, a methodology for applying the Time Warp mechanism to routing is described. Implementation techniques for efficient parallel execution are overviewed in Section 4. In Section 5, we report on experimental results. Our conclusion is given in Section 6.

2 Gridless Routing Algorithm

We apply the following assumptions in our explanation:

- Only two layers are available; one for vertical paths and the other for horizontal paths. When a path bends, two layer are connected with a via hole at the bending point[2].
- All nets consist of two terminals; a terminal can be either a *start* or a *target*.

With respect to data structuring in our router, both terminals and the whole routable area are represented by rectangles. For example, on the horizontal layer, rectangles corresponding to routable regions are generated by horizontally slicing the routable space at all obstacle vertices. Figure 1 shows the rectangles generated for each layer.

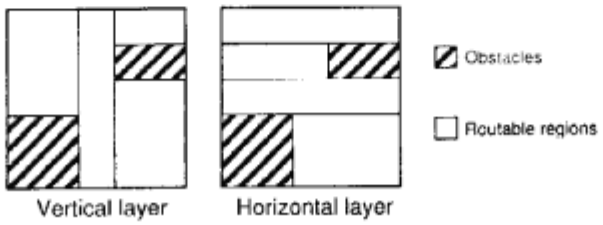

Figure 1: Routable regions represented by rectangles

Figure 2 shows the rough flow of the algorithm. The routing process for a net is roughly divided into two phases; the search phase and the path fix phase.

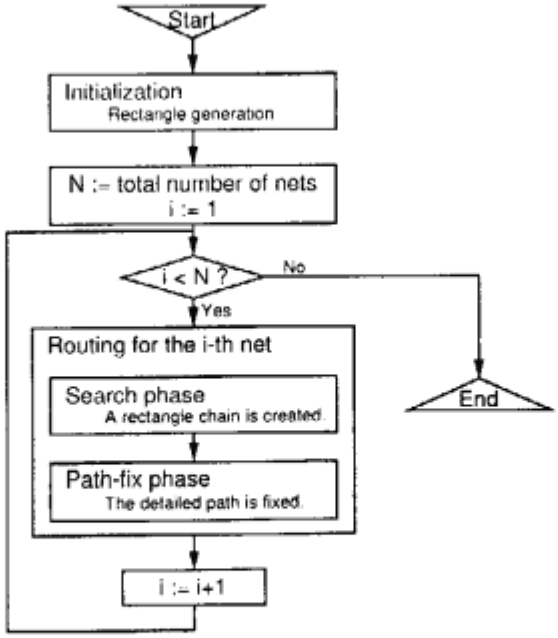

Figure 2: Routing process flow

[2]The width of each path and the size of via holes are specified in the preprocessing stage according to design rules.

2.1 Search Phase

Here, we say that a terminal and a routable region are touching if they are on the same layer and one of their edges overlaps. Also, we say that two routable regions are touching if they are on different layers and overlap (Figure 3).

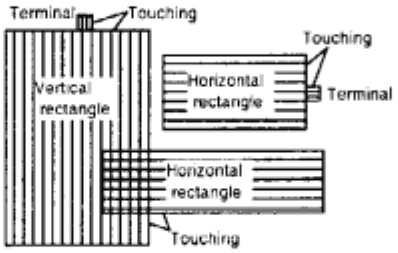

Figure 3: Touching rectangles

In the search phase, two rectangles are linked if they are touching with enough overlap to connect a path[3]. The linking operation starts at the start terminal, and continues until the target terminal is linked[4] or all routable regions are linked[5].

The linking operation proceeds in the evaluation function f order so that our routing algorithm is based on the A^* algorithm. f is given as follows:

$$f = g + h$$

where g is the cost from the start terminal S to the intermediate square M, while h is an underestimated cost from M to the target terminal T. The intermediate square, whose size should be adequate for the via hole, is the closest square from the previous intermediate square in the rectangle (Figure 4). For each start terminal, the intermediate square is itself.

In routing, g is usually represented by the sum of the total length of the already-searched route[6] and the total bending penalty[7] between S and M. For h, the sum of the Manhattan distance and the penalty for the minimum bending between M and T gives a good under-estimate.

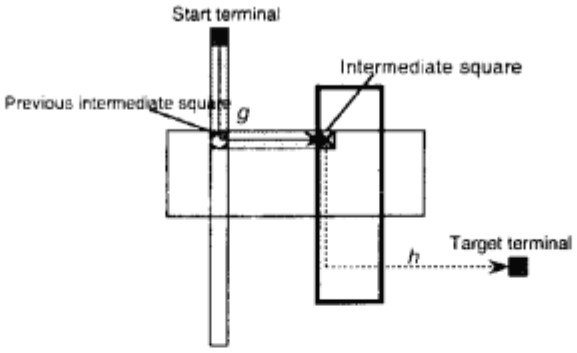

Figure 4: An intermediate point

In this phase, a chain of rectangles which includes an optimal path with the least cost is created if the path exists. Figure 5 shows snapshots in the search phase.

2.2 Path-fix Phase

The detailed path is fixed in this phase by tracing the rectangles backwards along the created chain. When a rectangle is traced back, an appropriate part of the rectangle is fixed as a wire area. To fix a path is, operationally, to partition the rectangle into smaller sub-rectangles and regard the wire area as a new obstacle

[3]Note that searches should be appropriately pruned to avoid redundant linking.
[4]An optimal path is found.
[5]No path is found.
[6]The path length for each layer may be weighted differently according to whether it is silicon or metal.
[7]The penalty must be a non-negative value.

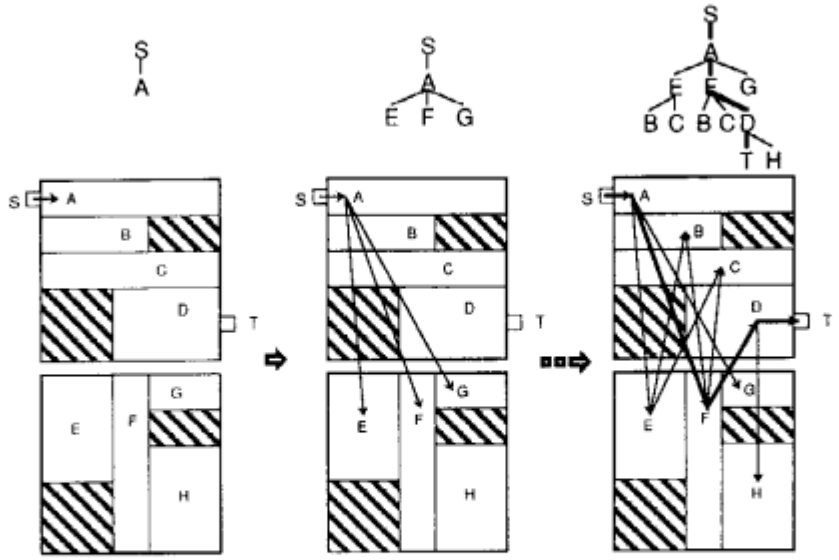

Figure 5: Snapshots in the search phase

(Figure 6). These operations continue until the traceback operation reaches the start terminal. Note that once a rectangle is partitioned, it can not be used in the following routing process, although the sub-rectangles are used.

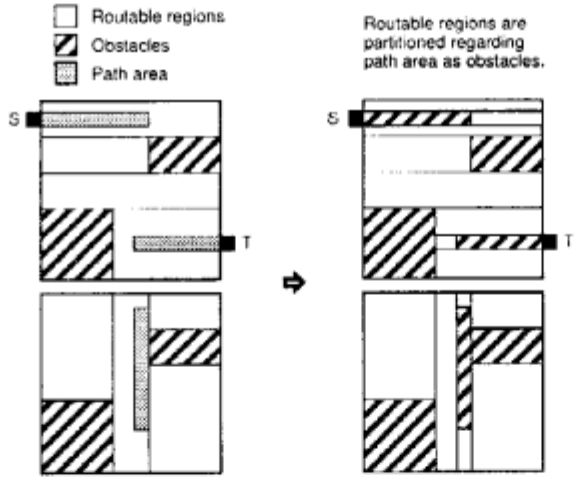

Figure 6: Rectangle partitioning

3 Parallel Routing with Time Warp

Our target is to run the router efficiently on highly parallel machines, most of which are MIMD machines with distributed memory. Therefore, high parallelism, which is practically available even on such machines, should be extracted from our routing algorithm.

Unfortunately, our method involves considerable sequentiality which leads to difficulty in exploiting parallelism. The major sources of sequentiality are as follows:

- The linking operation is performed in f order.
- Routing processes for any two nets cannot be performed in parallel.

The first source of sequentiality is needed in order to follow the A^* algorithm. Of course, several parallel A^* algorithms have been proposed[5, 7]. In these algorithms, however, global synchronization is needed which

causes large overheads on MIMD machines with distributed memory[8]. In the second source of sequentiality, the routing order must be kept rigidly in order to provide high quality solutions [9].

To overcome the difficulty, we introduced the Time Warp mechanism into our router. The Time Warp mechanism is a well-known mechanism which can extract high parallelism from the target problems at runtime with speculative computation. Sometimes speculation errors may occur, but they are collected by the rollback operation. Eventually, sequentiality (i.e., the processing order) is completely maintained.

We considered that the whole sequentiality in our router is maintained without any global synchronization by introducing the Time Warp mechanism. And we expected that the mechanism would lead to efficient execution of our router on large-scale MIMD machines with distributed memory.

In the following, the Time Warp mechanism is overviewed. Then, our modeling methodology and application of the mechanism to routing are presented.

3.1 Time Warp Mechanism

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

In the Time Warp mechanism[6], 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.

But, in reality, 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 antimessages to cancel them. The rollback operation allows the Time Warp mechanism to remain within its constraint.

On the other hand, Global Virtual Time (GVT) should sometimes be updated for memory management. Details are presented in [6].

3.2 Object-oriented Modeling of Routing Algorithm

We used an object-oriented methodology to model our routing algorithm[3].

In our router, terminals and initial routable regions are modeled as objects. An object has a state that corresponds to the current routable area within it. The initial state is the whole area of the object, while the state changes to several sub-rectangles as routing proceeds (Figure 7).

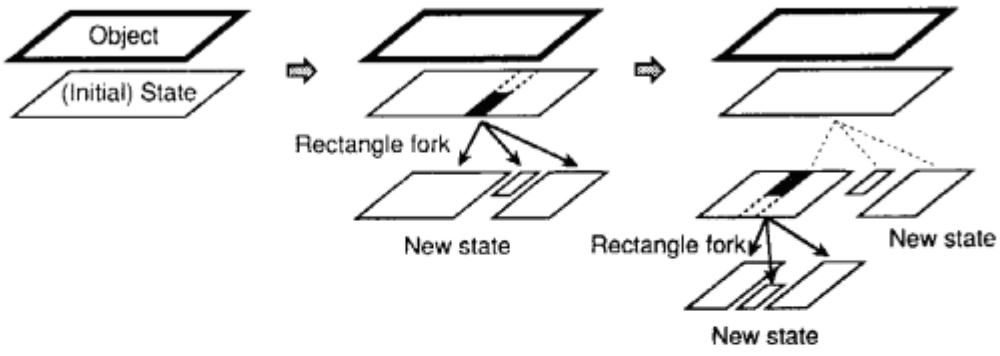

Figure 7: State transition of an object

Objects communicate with each other to perform routing. The *search*, *terminate* and *traceback* messages are used for communication. The expansion of the linking operation in the search phase corresponds to the transmission of a *search* message between objects. An f value is attached to each *search* message. Note that there is a constraint whereby the messages must be received by the destination object in f order. This means

[8]For example, high parallelism becomes available by allowing multiple linking operations with different f values to be performed in parallel. In this approach, since the chain created first does not give an optimal path sometimes, the least-cost path must be selected from all chains created after the search phase terminates. Generally, global synchronization is needed in this case.

[9]If the search area for each net is bounded in preprocessing, multiple nets can be routed in parallel as long as the bounded areas for the nets do not overlap. But this approach can not guarantee that the optimal solutions will be obtained[14].

that routing is performed according to the A^* algorithm. When the target terminal receives a *search* message which completes an optimal chain of rectangles, the search phase terminates.

To notify all objects of the termination of the search phase, *terminate* messages are broadcast. When an object receives the *terminate* message for a net, it stops all operations concerned with that net. Any received message related to the net is ignored thereafter.

After the termination of the search phase, the path-fix operation starts by sending a *traceback* message from the target terminal to the adjacent rectangle in the chain. When an object receives a *traceback* message, it computes the appropriate wire area and changes the state, regarding the area as a new obstacle. At the same time, the object sends a *traceback* message to the next rectangle in the chain. When the start terminal receives the *traceback* message, the path-fix phase terminates.

3.3 Timestamp vs Routing Order and f Value

In order to apply the Time Warp mechanism to routing, we made following optimistic assumptions.

- The *search* messages will arrive at their destinations in f order. So, the *search* message which arrives at the target terminal first will create the chain for an optimal path[10].
- Any rectangles involved in the chains will be different[11]. So, routing of multiple nets can be performed in parallel.

If these assumptions are correct, high parallelism can be exploited. Even if there are some exceptions, these can be corrected by the rollback operation. Thus, finding the optimal paths and maintaining the routing order are both guaranteed while parallel routing.

In the Time Warp mechanism, the correctness of speculative computations are judged in terms of the timestamp order. So, we let the tuple of the routing order N and cost f, (N, f), be a timestamp and add it to each message. *terminate* and *traceback* messages also have timestamps equal to the *search* message received by the terminal.

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

$$
\begin{array}{lll}
\text{IF} & N_l < N_m & \text{THEN } TS_l < TS_m \\
\text{ELSE IF} & N_l = N_m \text{ and } f_l < f_m & \text{THEN } TS_l < TS_m \\
\text{ELSE IF} & N_l = N_m \text{ and } f_l = f_m & \text{THEN } TS_l = TS_m \\
\text{ELSE} & & TS_l > TS_m
\end{array}
$$

In our router, objects evaluate messages concurrently, while the processing order represented by the timestamp is thoroughly maintained in the result by the Time Warp Mechanism. In other words, all messages can be regarded as being processed in f order, and, as a result, the obtained path is guaranteed to be the optimal path. Furthermore, with respect to the routing order, the solutions obtained are the same as those when routing sequentially.

3.4 Examples of Rollback

To further understand the behavior of our router, we give two typical examples of rollback[12].

In the following, $S_r^s(N, f)$ is a *search* message, $TM^s(N, f)$ is a *terminate* message and $TR_r^s(N, f)$ is a *traceback* message. The timestamps of these are (N, f). The superscript s shows the sender object, while subscript r shows the receiver object.

Case A: The optimal path is obtained later.

Target terminal object t received $S_t^o(10, 100)$ from object o. Then, it broadcast $TM^t(10, 100)$ and sent $TR_o^t(10, 100)$ and t received $S_t^{o'}(10, 50)$.

In this case, the rollback operation is needed; t sends antimessages to cancel both $TM^t(10, 100)$ and $TR_o^t(10, 100)$. Then, t broadcasts $TM^t(10, 50)$ and sends $TR_{o'}^t(10, 50)$ for the traceback operation (Figure 8). Note that $S_t^{o'}(10, 50)$ gives the optimal path if no *search* message arrives in the future. By this rollback operation, the obtained paths are guaranteed to be optimal.

Case B: Routing proceeds out of order.

[10] Therefore, the path-fix phase can start as soon as the target terminal object receives a *search* message, without global synchronization.

[11] In this case, the routing order is meaningless. Routing in a different order will not result in a different solution.

[12] Although rollback occurs in many other cases, we do not have the space here to show such examples.

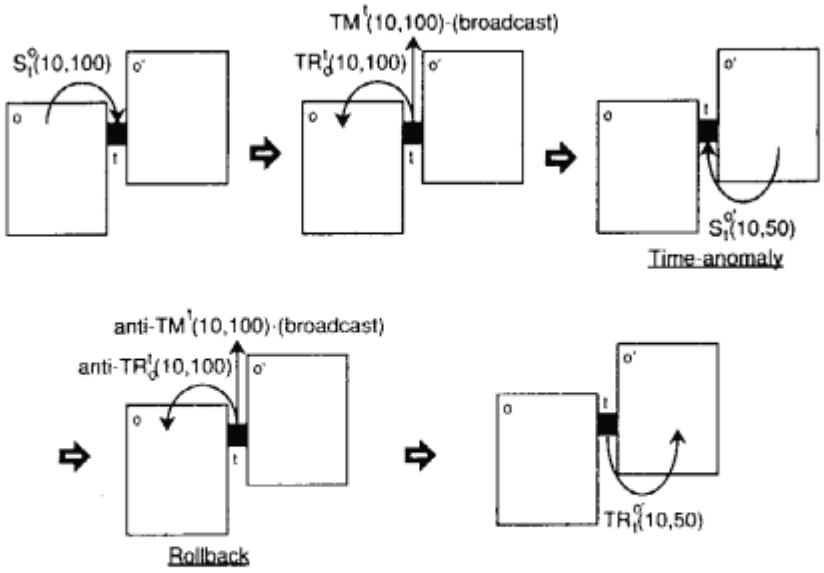

Figure 8: An example of rollback(A)

Rectangle object o received $TR_o^p(10,100)$ from object p. Then, o changed its state and sent $TR_n^o(10,100)$ to the next object n in the rectangle chain and o received $TR_o^{p'}(5,200)$.

In this case, o rewinds its history, recovers its state at (5,200) and sends antimessages to cancel $TR_o^p(10,100)$. Then o changes its state according to $TR_o^{p'}(5,200)$ and sends $TR_n^o(5,200)$[13] (Figure 9). This rollback operation maintains the routing order.

In addition to the above examples, the rollback operation is performed when an antimessage arrives but the message to be canceled was already evaluated.

4 Implementation

4.1 Machine and Language

The router is written in concurrent logic language KL1 [13] and is implemented on the PIM/m[12].

KL1 supports data-flow synchronization. This is a powerful feature for describing concurrent objects which work 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 area of the Time Warp mechanism).

The PIM/m is an MIMD machine where up to 256 processing elements can be connected to each other by a 2-dimensional mesh network. The PIM/m is a distributed memory machine, so remote access to a different processing element costs considerably more than local access. The PIM/m is, however, easy to scale up.

4.2 Message Scheduler

There are usually many messages to be evaluated in a processor. When the Time Warp mechanism is used, the bigger timestamp a message has, the more likely the message is to be canceled. For this reason, message scheduling in each processor is expected to reduce rollback frequency effectively[2].

Our system has a message scheduler for each processor. When a message is spawned, it is first registered in the scheduler which manages the destination object. The scheduler picks up the message with the smallest timestamp and sends it to the destination object at the appropriate moment.

4.3 Localization of Time-anomaly Check

In the Time Warp mechanism, when a time-anomaly is detected at an object, the rollback operation is performed to correct the anomaly. However, if the time anomaly does not cause an illegal result, the rollback operation need not be performed.

[13] In this case, if o sent messages with timestamps larger than or equal to (10,100), they must be canceled. In addition, all received messages with timestamps larger than or equal to (10,100) are re-evaluated using the new state after rollback.

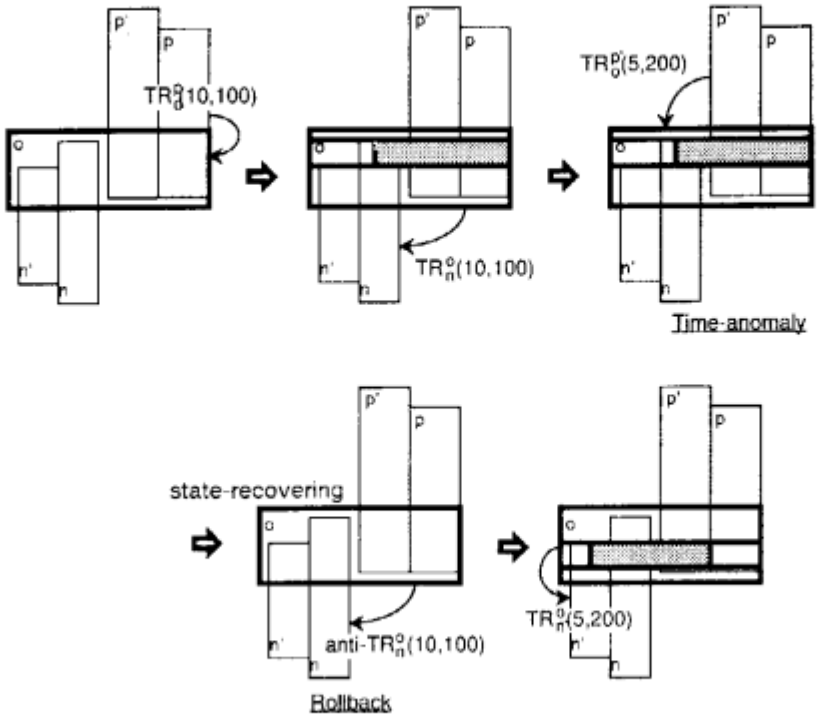

Figure 9: An example of rollback (B)

For this reason, in our router, the time-anomaly check is done at a more primitive level than the object level. More precisely, the rollback operation is done only when messages arrive out of order and affect the same sub-rectangle in the state (Figure 10). The localized time-anomaly check is expected to significantly reduce rollback frequency.

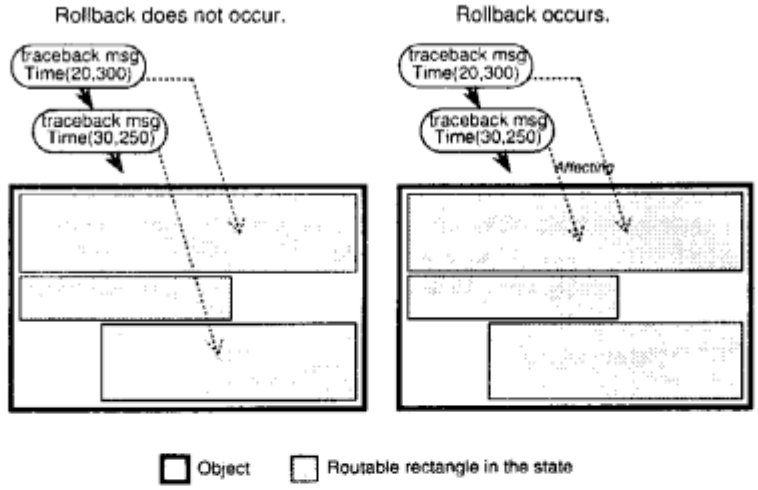

Figure 10: Localized time-anomaly check

4.4 Load Distribution

We use a static load distribution scheme based on a round-robin strategy, where objects are distributed in a cyclic manner. Note that not just any object can migrate to other processors (static load distribution).

This strategy is good from the viewpoint of load balancing and parallelism extraction, while it is not good from the viewpoint of reducing inter-processor communication.

5 Measurement Results

We tested our router on the PIM/m with 64 processors. In our experiments, we used an industrial LSI data with 136 nets and 528 obstacles on each layer. In order to clarify the advantage of employing the Time Warp mechanism, we performed two experiments and compared them.

(a) Routing with the Time Warp mechanism: multiple nets are routed in parallel while the routing order is maintained by the Time warp mechanism.

(b) Routing with a parallel A^* algorithm: the search phase for each net is parallelized while allowing redundant computation, and routing for each net is performed sequentially.

In (b), there is global synchronization whenever a search phase terminates. This is needed to guarantee that optimal paths are obtained. Note that the solutions obtained are the same in both experiments.

Figure 11-(i) shows speedup — the relative execution speed[14] using multiple processors compared to that when using one processor. The number of *search* messages generated in routing is also shown in Figure 11-(ii). The solid line corresponds to (a), while the dotted line corresponds to (b). The figure shows that 14-fold speedup was attained using 64 processors in (a), whereas speedup peaked at 32 processors in (b), at 6-fold speedup. The decline in speedup in (b) is considered to have been caused by the high cost of global synchronization on large scale distributed memory machines.

This comparison reveals that parallel routing with the Time Warp mechanism is effective for accelerating processing, especially on large-scale distributed memory machines.

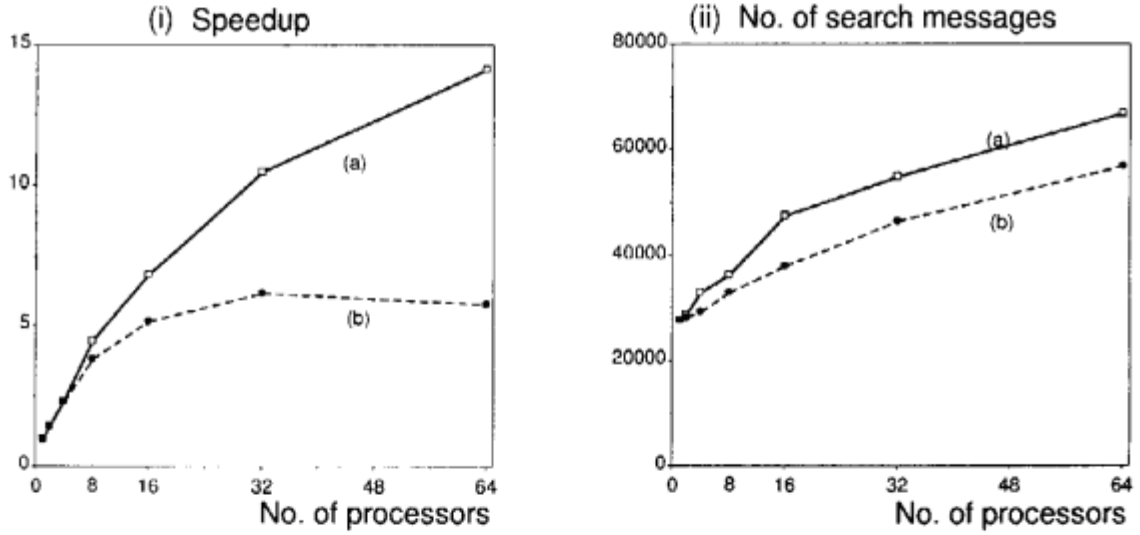

Figure 11: Speedup and No. of search messages

The number of *search* messages was 2.43 times larger than that when using one processor. However, even without the Time Warp mechanism, the number increased up to 2.05 times that when using one processor. Therefore, we conclude that the increase caused by the Time Warp mechanism was not significant. One reason for this should be that the pace of the routing process on each processor was approximately equal due to our load distribution strategy. The round-robin strategy usually attains good load balancing.

In addition, we confirmed that our router got a high quality solution in terms of both wireability and the path length; all nets were routed and an optimal path was obtained for each net.

Our experiments showed that our router maintained the routing order and guaranteed an optimal solution while running efficiently in parallel with the Time Warp mechanism.

6 Conclusion

In this paper, we presented a gridless router with the Time Warp mechanism. The basic algorithm is a gridless algorithm based on rectangle partitioning. We employed an object-oriented methodology for modeling our

[14] The inverse of the execution time.

algorithm and extracted high parallelism by applying the Time Warp mechanism to our router. The Time Warp mechanism exploits high parallelism with speculative computation at runtime, while maintaining the needed sequentiality. With the Time Warp mechanism, efficient parallel routing was realized even on a large-scale distributed memory machine, while guaranteeing not only that the routing order was kept but also that optimal paths were obtained.

We built the system on the PIM machine, which is a large-scale MIMD machine with distributed memory, and evaluated the performance. In our experiments, 14-fold speedup using 64 processors was obtained. With respect to the quality of the solution, 100% wireability was attained. We also confirmed that the path for each net was the optimal one.

Our future works are as follows:

1) We are examining our router with larger circuit data to evaluate its practical applicability.

2) For the purpose of accelerating the router, the lazy cancellation technique should be introduced into the router. This will reduce rollback frequency. An appropriate scheme for load distribution also should be developed.

3) In order to deal with cases where the given routing order is not good, a dynamic ripup-and-reroute mechanism is being embedded. Rip-up and rerouting can be done implicitly by means of the rollback operation of the Time Warp mechanism. This would become one of the most powerful features of our router.

References

[1] R. J. Brouwer and P. Banerjee, "PHIGURE : A Parallel Hierarchical Global Router," In *Proc. 27th DA Conf.*, 1990, pp. 650–653.

[2] V. Burdorf and J. Marti, "Non-Preemptive Time Warp Scheduling Algorithm," *ACM Operating System Review*, Vol.24, No.2, pp. 7–18, 1990.

[3] H. Date *et al.*, "LSI-CAD programs on Parallel Inference Machine," In *Proc. Int. Conf. on Fifth Generation Computer Systems*, 1992, pp. 237–247.

[4] W. Heyns *et al.*, "A Line-Expansion Algorithm for the General Routing Problem with a Guaranteed Solution," In *Proc. 17th DA Conf.*, 1980, pp. 243–249.

[5] Keki N. Irani and Yi-fong Shih, "Parallel A^* and AO^* Algorithms: An Optimality Criterion and Performance Evaluation," In *Proc. ICPP'86*, 1986, pp. 274–277.

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

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

[8] A. Margarino *et al.*, "A Tile-Expansion Router," *IEEE Trans. on CAD*, Vol. CAD-6, No.4, pp. 507–517, 1987.

[9] Y. Matsumoto and K. Taki, "Parallel Logic Simulation on a Distributed Memory Machine," In *Proc. EDAC'92*, 1992, pp. 76–80.

[10] T. Ohtsuki and M. Sato, "Gridless Routers for Two-Layers Interconnection," In *Proc. ICCAD*, 1984, pp. 76–78.

[11] M. Sato *et al.*, "A Hardware Implementation of Gridless Routing Base on Content Addressable Memory," In *Proc. DA Conf.*, 1990, pp. 646–649.

[12] K. Taki, "Parallel Inference Machine PIM," *Proc. Int. Symp. on Fifth Generation Comp. Sys.*, pp. 50–72, 1992.

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

[14] T. Yamauchi *et al.*, "PROTON: A Parallel Detailed Router on an MIMD Parallel Machine", *Proc. ICCAD*, pp. 340–343, 1991.