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

LSI-CAD Programs on Parallel Inference Machine

Hiroshi Date† Yukinori Matsumoto† Koichi Kimura†
Kazuo Taki† Hiroo Kato‡ Masahiro Hoshi‡

†Institute for New Generation Computer Technology
1-4-28, Mita, Minato-ku, Tokyo 108, Japan
{date, yumatumo, kokimura, taki}@icot.or.jp

‡Japan Information Processing Development Center
3-5-8, Shibakouen, Minato-ku, Tokyo 105, Japan
{j-kato, hoshi}@icot21.icot.or.jp

Abstract

This paper presents three kinds of parallel LSI-CAD systems developed in ICOT and describes their experimental results on a parallel inference machine. These systems are routing, placement and logic simulation. All of them are implemented in KL1, a concurrent logic language, and executed on the Multi-PSI, a distributed memory machine with 64 processors.

We regard our parallel inference machines as high performance general purpose machines. We show programming techniques to derive high performance on parallel inference machines. The common objectives of these systems are, firstly, to provide speedup by extracting major parallelism, and, secondly, to show the applicability of our hardware and language system to practical applications. For this reason, our systems are evaluated using real LSI chip data.

The key features are, in the routing system, *concurrent object modeling* of routing problems to realize a lot of concurrency; in the placement system, *time-homogeneous parallel simulated annealing* to optimize placement results; and in the logic simulation system, *the Time Warp mechanism* as a time-keeping mechanism for simulations.

Experimental results of these systems show that these techniques are effective for parallel execution on large-scale MIMD machines with distributed memory structure, like the parallel inference machines.

1 Introduction

A parallel computer system PIM (the Parallel Inference Machine), one of the goals of the Japanese Fifth Generation Computer Systems project, has been completed, and its evaluation is starting. PIM has been developed mainly to target high performance knowledge information processing. Since most problems in this domain are of an extremely large size, exploiting the whole power of parallel machines is important. In practice, however, it is not easy to derive their maximum power because of the non-uniformity of computation, that is dynamically changing parallel computation depending on time and space.

In order to move programs efficiently on PIM, the following are important. First is to adopt good concurrent algorithms. Second is to design programs based on programming paradigms to realize high parallelism. And last is to use effective load distribution techniques including processor mapping. We aimed at gaining experiences with these techniques through large-scale practical application experiments on PIM.

PIM is an *inference* machine, however, its applicability should not be limited to knowledge information processing. From the viewpoint that PIM is a high performance general purpose machine, we chose LSI-CAD as one of the application fields.

Nowadays, LSI-CAD is indispensable for LSI design. The integration of the LSI chip has increased exponentially in proportion to the progress of the semiconductor process technology. The quality of LSIs depend on the performance of LSI-CAD tools. Therefore, higher performance is required. Besides, the flexibility of the tools must be kept for a variety of demands. Using hardware accelerators is one possible way of obtaining faster tools, however, it usually results in a sacrifice of flexibility. A likely alternative is to parallelize software tools. This certainly satisfies the above two requirements: making the tools faster and keeping their flexibility.

We focused on three stages of LSI-CAD; *logic simulation, placement* and *routing*, which are currently the most time-consuming in LSI design. Each system has following features.

The routing system finds paths based on the lookahead line search algorithm [Kitazawa 1985]. This algorithm provides high quality solutions, however, it was originally proposed with the assumption of sequential execution. We introduced a new implementation method of parallel

router based on the concurrent objects model, and improved the basic algorithm to make it suitable for parallel execution. The concurrent objects model is expected to derive a lot of parallelism among small granular processes. We investigated the description complexity and overhead of our routing programs. Also its performance (real speed, speedup and wiring rate) was evaluated in comparison with a sequential router on general purpose computer using real LSI chip data.

The cell placement problem is a combinatorial optimization problem. Simulated annealing (SA) is a powerful algorithm to solve such problems. Cooling schedules are important for efficient execution of SA. In our placement system, the time-homogeneous parallel SA algorithm [Kimura *et al.* 1991] was adopted. This algorithm constructs appropriate cooling schedules automatically. We evaluated quality of solutions in our system using MCNC benchmark data.

Logic simulation is an application of discrete event simulation. The key to its efficient execution in parallel is keeping the time correctness without large overheads. We adopted the Time Warp mechanism (TW) as the time-keeping mechanism. TW has been considered to contain large rollback overheads, however, it has not been evaluated in detail yet. We not only improved the rollback process but also added some devices so that TW would become an efficient time-keeping mechanism. Cascading-Oriented Partitioning strategy for partitioning circuits are also proposed to attain good load distribution. We evaluated our system on speedup and real speed (events/sec) as compared with the systems that had other time-keeping mechanisms (Conservative and Time Wheel) using ISCAS89 benchmark data.

These systems were implemented in KL1 [Chikayama *et al.* 1988, Ueda *et al.* 1990], a concurrent logic language, and have been experimented with on the Multi-PSI/V2 [Nakajima *et al.*1989, Taki 1988], a prototype of PIM.

This paper is organized as follows : The routing system is described in Section 2. A routing algorithm based on the concurrent objects model and its implementation is presented in detail. Section 3 explains the placement system. The time-homogeneous SA algorithm is introduced and optimization in the implementation is explained. Section 4 overviews the logic simulator and reports on its evaluation. Our conclusion is given in Section 5.

2 Routing System

2.1 Background

There have been many trials to realize high speed and good quality router systems with parallel processing. These trials can be classified into two areas. One is the hardware engine which executes the specified routing algorithm efficiently [Kawamura *et al.* 1990, Nair *et al.* 1982, Suzuki *et al.* 1986]. The other is concurrent routing programs implemented on general purpose parallel machines [Brouwer 1990, Olukotun *et al.* 1987, Rose 1988, Watanabe *et al.* 1987, Won *et al.* 1987]. The former approach can realize very high speeds, while the latter can provide large flexibility. We took the latter approach to realize both high speed and a flexible router system, targetting very large MIMD computers.

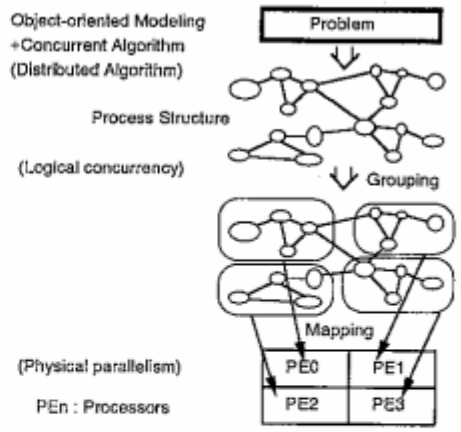

Figure 1: Program design paradigm based on the concurrent objects model

In general, a lot of parallelism is needed to feed a large MIMD computer. So, we propose a completely new parallel routing method, based on a small granual concurrent objects model. The routing method was implemented on the distributed memory machine, Multi-PSI, with a logic programming language KL1. We made preliminary evaluations of the new router, from the viewpoints of (1) data size vs. efficiency, (2) wiring rate vs. parallelism, and (3) comparison of execution speed with general purpose computers.

This section contains the following. A programming paradigm based on the concurrent objects model, a router program with an explanation of concurrent algorithms and implementation, problems in parallelization, and preliminary measurements and evaluation results.

2.2 Programming Paradigm

Formalizing a problem based on *the concurrent objects model* is one of the most prospective ways to embed parallelism in a given problem.

This section describes our methodology to design parallel programs from problem formalization to parallel execution. We also show coding samples in the KL1 language.

Figure 1 shows the flow of parallel program design. Firstly, a given problem is formalized based on the concurrent objects model. That is, many objects make a solution cooperatively, by exchanging messages. At the same time, a concurrent algorithm is designed upon the

```
Concurrent_Object ( [ Message_1 | Rest ], Interior state variables, Stream variable ) : -
    true |
        Process correspond to Message_1.          Output of Message to other objects
        Renew of Interior state variables,
        Stream variable = [Message | New Stream variable],
        Concurrent_Object ( Rest , New Interior state variables, New Stream variable ).

Concurrent_Object ( [ Message_2 | Rest ], Interior state variables, Stream variable ) : -
```

Figure 2: Implementation of a concurrent object in KL1

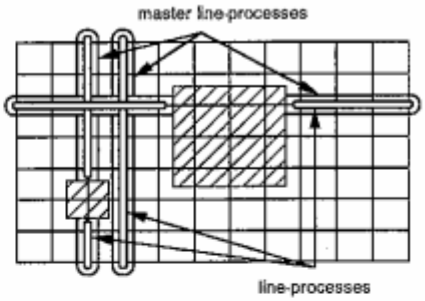

Figure 3: Master line processes and line processes

model. Sometimes, the algorithm is a distributed algorithm. Through this design phase, the activities of the objects corresponding to messages are defined.

Then, each object is implemented as a KL1 process. Process connection topology is decided based on input data. Usually, a much larger number of processes is needed than the number of processors to get good load balance. Logical concurrency (the possibility of parallel processing) is designed through this flow.

Secondly, the processes, which exchange messages frequently, are grouped to increase communication locally. When each process has a large computational amount (large granularity) and a low communication rate, this phase can be omitted.

Then, the groups are assigned to processors and executed. This is called mapping. Physical parallelism is realized in this phase. The KL1 language system allows independent descriptions of the problem solving part (logical concurrency) and the mapping part (physical parallelism) of a program. Performance tuning of parallel processing can be done only by changing the mapping part, not by changing the problem-solving algorithm.

The KL1 language is quite suitable for describing concurrent objects. Processes representing the objects are written in the self recursive call of the KL1 language. These processes can communicate with each other through the message streams. Figure 2 shows a coding sample of an object. The functions of an object are defined with a set of clauses. Each clause corresponds to a message which the object receives.

2.3 Router Program

We used the lookahead line search method [Kitazawa 1985] as a basic algorithm. Then we reconstructed the algorithm for highly parallel execution, taking the concurrent objects model as a basic design framework.

2.3.1 Basic algorithm

The lookahead line search method is one of the line search algorithms coupled with lookahead operation. It is, if you like, a sort of hill-climbing algorithm, looking for a good route. The algorithm, also, has two features. One is to escape from the local optimum point with the help of the Inhibited Expected Point (IEP) flag. The other is backtracking to retrace bad routes and to retry searching. The algorithm guarantees connection between a start point and a target point when paths exist between them.

2.3.2 Concurrent routing algorithm

In KL1 programming, an execution unit is a process corresponding to an object. Since the line search algorithm decides a route, line by line, we designed the concurrent algorithm so that objects=processes corresponds to every line segment on a routing grid. Line processes exchange messages with each other to look for a good route. Each line process maintains the corresponding line's status and, at the same time, the execution entity of the search.

As Figure 3 shows, each process corresponds to each grid line (*master line process*) and line segment (*line process*) on it. A master line process manages line processes on the same grid line and passes messages between the line processes and crossing line processes.

The routing procedure of one net is almost the same as that of the basic algorithm, except that the procedure is broken down into a sequence of messages and their operations are executed among processes. Computing the best expected point is done as follows. The expected point is the closest location to a goal on a line segment. The distance to the goal is used for a cost function in the hill climbing method.

When a line process receives a routing request message with information of a goal point, it changes its status to "under searching". Then, it sends request messages for calculation of expected points to line processes that cross it (Figure 4).

Thus computation of the expected point is executed concurrently on each line process that receives the request message.

After the computation results are returned to the searching line process, it aggregates those results and determines the best expected point.

When the best expected point is determined, the searching line is connected to the crossing line that includes the best expected point. The searching line process splits into an occupied part and a free part, and

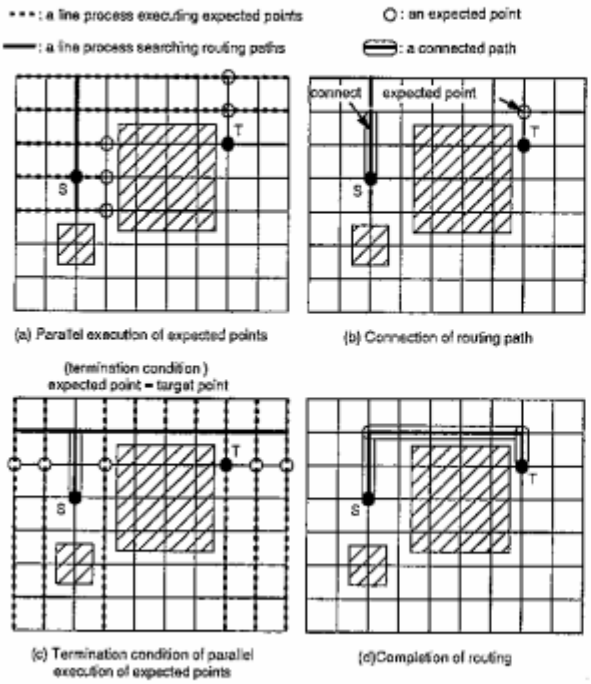

Figure 4: Parallel execution of expected points

the status is maintained. Then, the next routing request message is sent to the connected crossing line.

Messages are sequenced at the entrance of each process. Only one message can be handled at any time in a process. No problems of exclusive access to an object or locking/unlocking objects arise with this scheme.

In our algorithm, two types of parallelism are embedded. One is concurrent computation in the lookahead operation and the other is concurrent routing of different nets.

2.4 Problems in Parallel Execution

When we parallelize the lookahead line search method, three problems arise. The first is deadlock, the second is conflict among routing nets and the last is memory overflow for communicating between processors.

2.4.1 Avoidance of deadlock

When two or more nets are searched concurrently, deadlock may occur. Figure 5 shows an example. When line processes that intersect orthogonally send request messages to compute the expected point to each other at the same time, computation will not occur. This is because they cannot carry out the next messages until the execution of present messages terminates.

If it is guaranteed that execution of a message terminates within a fixed period, deadlock can be avoided. To

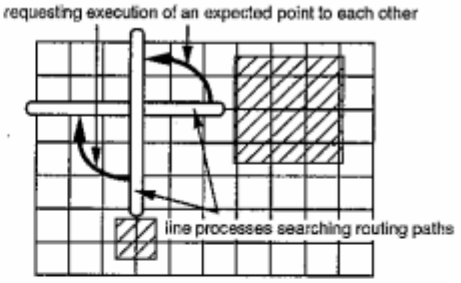

Figure 5: Example of deadlock

satisfy this condition, we made the following modification.

Firstly, messages are grouped into group A and group B. B-type messages are guaranteed to terminate execution within a fixed period. A-type messages are not guaranteed to terminate, that is, some synchronization with other processes is needed before terminating message execution.

We modify the operations of A-type messages as follows. Each process executing an A-type message observes all messages arriving successively. When an A-type message is found, it is left in a message queue, that is no operations are performed. When a B-type message is found, it is processed immediately before termination of the currently executing A-type message. For this processing of B-type messages, a temporary process status that differs from the sequential algorithm is needed. By applying this modification, deadlock can be avoided.

In our router, the routing request messages are A-type, and the request messages of computing expected points are B-type.

2.4.2 Conflict among nets

When concurrent routing of multiple nets is done, different nets may conflict on the same line segment. In this situation, message sequencing works well and the first message to arrive (corresponding to net A) occupies the segment. The second message to arrive (net B) fails to complete a route, and backtracks.

However, net A may backtrack afterwards and may release the line segment. In this case, net B does not visit the line segment anymore and the line segment may be left unused. This fact causes lower quality routes (longer paths) or a lower wiring rate (more unconnected nets).

To avoid those degradations in routing quality, the scheduling of the order in which nets start routing is important and may limit concurrency by eliminating the number of nets routed concurrently. However, parallelism can be affected by these controls. Relations between wiring rates and parallelism are studied in the experiments.

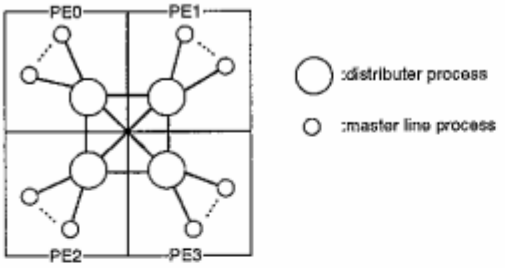

Figure 6: Improved process structure

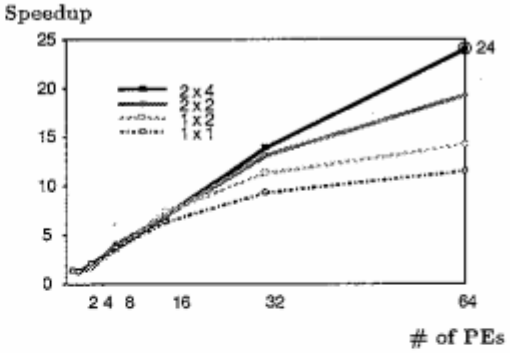

Figure 7: Data size vs. speedup

2.4.3 Overflow of memory for communication among processors

When we implement the concurrent program using KL1, two kinds of memory are necessary. One is the memory for representing processes. The other is the memory for communication paths among processors.

In our routing program, the process structure shown in Figure 3 was implemented. Each master line processes, which mediates between line processes, must communicate all orthogonal master line processes. Therefore the number of communication paths is increasing for large-scale data. Experimental results show that the maximum grid size of chip data to be treated by this routing program is about 500 × 500. This size is too small for applying practical data.

In order to solve this problem, we improved the process structure, as in Figure 6. Each distributer process controls communication among processors.

2.5 Measurements and Evaluation

We evaluate our router from the following three points of view. (1) Data size vs. Speedup, (2) Parallelism vs. Wiring Rate, and (3) Comparison with a general purpose computer. The program was executed on a MIMD machine with distributed memory and 64 processors, the Multi-PSI. Two types of real LSI data were used. The features of these data are shown in Table 1. Terminals to be connected are distributed uniformly in DATA1. Meanwhile, terminals are concentrated locally in DATA2. DATA3 is large-scale data.

Table 1: Testing data

Data	DATA1	DATA2	DATA3
Grid size	262×106	322×389	2746×3643
# of nets	136	71	556
Presented by	Hitachi Ltd.	NTT Co.	NTT Co.

2.5.1 Data size vs. speedup

Generally, when data size increases, the number of processes increase too and more parallelism can be expected. Higher parallelism can lead to greater efficiency or larger speedup with a fixed number of processors. We measured the relationship between the size of data and speedup.

In this experiment, we used data copying DATA1. Here we measured four cases (1×1, 1×2, 2×2, and 2×4). Figure 7 shows the result of measurement. This graph shows that the larger the size of data, the higher the speedup. It also shows 24-fold speedup with 64 processors for 2×4 data it does not look saturated yet. We have to investigate the limit of speedup with increasing data size.

2.5.2 Wiring rate vs. parallelism

Parallel routing of multiple nets may cause a degradation in wiring rate. We measured the relation between wiring rate and parallelism for DATA1 and DATA2, as shown in Figure 8. The two vertical axes show execution time and wiring rate. The horizontal axis shows the number of nets routed concurrently. Parallelism is proportional to this. When equal to one, parallelism only arises from parallel lookahead operations. It was observed that terminal-distributed data shows good wirability, even if parallelism is high, when the terminal-concentrated data is poor. Concentrated terminals tend to cause a lot more net confliction.

2.5.3 Comparison with a general purpose computer

The execution time of DATA2 with a single processor was measured as 111 seconds. From Figure 8, speedup caused only by lookahead operation is calculated as 4.9.

The execution time of our system was compared with a general purpose computer, the IBM 3090/400, which is a 15 MIPS machine. The sequential lookahead line search router on the IBM machine was developed by Dr. Kitazawa (NTT Co.) before our work was conducted. Table 2 shows the performance of the routers.

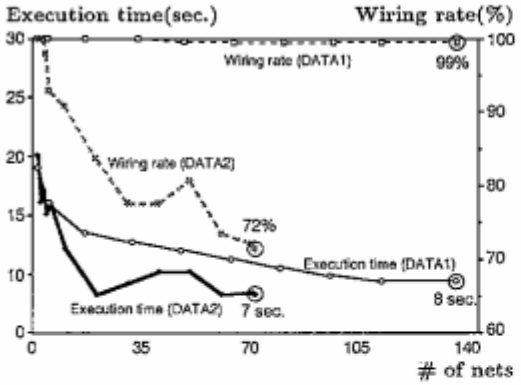

Figure 8: Wiring rate vs. parallelism

Two cases of the Multi-PSI measurements (with 64PEs) are included in the table. One routed all nets concurrently and the other routed each net one after another. The former case shows the better execution time but worse wiring rate. The latter case accomplished the perfect wiring rate but worse execution time for DATA2. We expect to realize both good execution time and good wiring rate by controlling the number of nets wired concurrently and changing the wiring order. (In fact, on DATA2, W:100 % and E:16 sec. under the number of nets wired concurrently is equal to 2.)

The evaluation for large data (DATA3) has just started. The wiring rate in the table is still insufficient but it will be improved as mentioned just above.

Table 2: Comparison of performance

Data\Machines		IBM 3090/400	Multi-PSI (64PEs) †	Multi-PSI (64PEs) ‡	Multi-PSI (1PE)
DATA2	E	7.45	7.0	20.0	111.0
	W	100	72	100	100
DATA3	E	405.0	360.0	N.A.	N.A.
	W	100	90	N.A.	N.A.

E:execution time (Sec.),W:wiring rate(%)
† concurrent wiring of all nets
‡ sequential wiring of each net

The execution time of the router on the Multi-PSI can be considered almost comparable with that on an IBM machine. When our router is ported to PIM machines, the next model to the Multi-PSI, the execution time will be reduced to 1/10 to 1/20 in execution with 256 to 512 processors.

The performance of the bare hardware of a Multi-PSI processor is 2 to 3 MIPS. And the efficiency of parallel processing (speedup/number of processors) is 25% for the case of Multi-PSI(64PEs) with concurrent wiring of all nets on DATA2. So, bare hardware performance with 64 processors is expected to be 32 to 48 MIPS (2 to 3 $\times$ 64 $\times$ 0.25). While, the actual performance is comparable with the 15MIPS machine. The degradation of actual performance must be caused by the implementation overhead of the object-oriented program and KL1 language.

2.6 Discussions

We presented a new routing method based on the concurrent objects model, which can include very large concurrency and is suitable for very large parallel computers. The program was implemented on a distributed memory machine with 64 processors. Preliminary evaluation was then done with actual LSI data.

The experimental results showed that the larger the data size, the higher the efficiency attained by a maximum of 24-fold speedup with 64 processors against single processor execution. The speedup curve did not look significantly saturated, that is, more speedup can be expected with more data.

In experiments on parallelism and the wiring rate, a good wiring rate with large parallelism was attained for data in which terminals are distributed uniformly. However, for data with concentrated terminals, the wiring rate became significantly worse, due to the increase in parallelism. We must improve the wiring rate in the latter case.

The actual performance of our router system was compared with an almost identical router on a high-end general purpose computer (IBM3090/400, 15 MIPS). Results showed that the speed of both systems was comparable. Based on a rough comparison of bare hardware speeds, the implementation overheads of the parallel object-oriented program and our language are estimated as 100 to 200% in total, against the sequential FORTRAN program on the IBM machine.

3 Placement System

3.1 Background

Cell placement is the initial stage of the LSI layout design process. After the functional and logical designs of the circuit are completed, the physical positions of the circuit components are determined so as to route all electrical connections between cells in a minimum area without violating any physical constraints. Heuristics for evaluating the quality of a placement usually promote one or more of the following: minimum estimated wire length, an even distribution of wires around the chip, minimum layout area, and regular layout shape.

The cell placement problem is well-known as a difficult combinatorial optimization problem. In other words, it is not feasible for obtaining the *optimum* placement of a circuit with practical size because it takes excessively amounts of CPU time. So efficient techniques to get nearly *optimum* placement must be employed in practice.

3.2 Simulated Annealing

Approximate methods are used to solve the combinatorial optimization problem. One such method is called *iterative improvement*. In this algorithm, the initial solution is generated, and, then, modified repeatedly to try to improve it. In each iteration, if the modified solution is better than the previous one, the modified solution becomes the new solution.

The process of altering the solution continues until we can make no more improvement, thus yielding the final solution. The problem with this algorithm is that it can be trapped at a *local optimum* in a solution space.

The Simulated Annealing(SA) algorithm [Kirkpatrick *et al.* 1983] is proposed to solve this problem. It probabilistically accepts a new solution even if the new solution may be worse temporarily. Its acceptance probability is calculated according to the change in the estimated cost value of the solution and the parameter "temperature". The cost function is often referred to as "energy". In this way, it is possible to search for the *global optimum* without being trapped by local optima.

The details of this algorithm are as follows.

It is constructed from two criteria, the *inner loop* criterion and the *stopping* criterion. At first, the initial solution and initial temperature are given. In the inner loop criterion, new solutions are generated iteratively and each solution is evaluated to decide whether it is acceptable. The units of iteration which are constructed by generating and estimating the new solution are called "step". In each stage of the inner loop criterion, the temperature parameter is fixed. In the stopping criterion, after a sufficient number of iterations are performed in the inner loop, the temperature is decreased gradually according to a given set of temperatures called the "cooling schedule". The stopping criteria are satisfied when the energy no longer changes.

One of the most difficult things in SA is finding an appropriate cooling schedule, which largely depends on the given problem. If the cooling schedules are not adequate, satisfiable solutions will never be obtained.

3.3 Parallel Simulated Annealing

A new parallel simulated algorithm(PSA) is proposed to solve the cooling schedule problem [Kimura *et al.* 1991]. The most important characteristic of this algorithm is that it constructs the cooling schedule automatically from the given set of the temperatures. The basic idea is to use parallelism in temperature, to perform SA processes concurrently at various temperatures instead of sequentially reducing the temperature. So it is scheduleless or *time-homogeneous* in the sense that there are no time-dependent control parameters.

After executing a fixed number of annealing steps, the solutions between the adjacent temperatures are probabilistically exchanged as follows. When the fixed number of annealing steps is denoted by k, $1/k$ is called the "frequency". When the energy of the solution at a higher temperature is lower than that at a lower temperature the solutions between these temperatures are exchanged unconditionally. Otherwise they are exchanged according to a probability that is determined by differences in their energies and temperatures [Kimura *et al.* 1991].

In PSA, even if a solution is trapped at a *local optimum* at a certain temperature, it is still possible to search for *global optima* because another new solution can be supplied from a higher temperature. So a *nearly optimum* solution will finally be found at the lowest temperature.

3.4 Outline of the System

Our experimental placement system employs the PSA algorithm. It is constructed on Multi-PSI, an MIMD machine, and the KL1 language is used to implement the system [Chikayama *et al.* 1988]. The intention is to provide a satisfiable solution in a feasible time. It is also applied to placement problems to examine the efficiency of the PSA algorithm.

The object of this system is the standard cell LSI without any macro blocks. The standard cells have uniform height and variant widths. These cells are arranged in multiple cell-blocks so as to minimize the chip area. Namely, it decides the location of each cell so as to minimize the total estimated wire length, which approximates the total routing length.

3.5 Implementation

3.5.1 Initial placement and new solution generation

The initial cell positions are determined randomly. In our placement system, SA processes are split into two temperature regions. The number of temperatures in the two regions should be specified by the user. Usually one or more temperatures are necessary for the lower region.

In the higher temperature region, there are two ways to generate a new solution. One way is to move a randomly selected cell to a random destination. The other way is to exchange the position between two randomly selected cells.

In the lower temperature region, generating a new solution is done by exchanging two arbitrary adjacent cells within a cell-block.

Moreover the range-limiter window is introduced [Sechen *et al.* 1985]. The range-limiter window restricts the ranges for moving and exchanging cells. The lower the temperature becomes, the smaller the size of the window becomes. It suppresses the generation of new solutions that are unlikely to be accepted.

3.5.2 Estimation of a new solution

The energy of a solution is the sum of the three values listed below [Sechen *et al.* 1985].

- estimated wire length
- the cell overlap penalty
- the block length penalty

The estimated wire length approximates the routing lengths between the cells. The estimated wire length of a single net is the half-perimeter length of the minimum bounding box which encloses all of the pins comprising the net.

The cell overlap penalty estimates the overlap between cells. In the higher temperature region, we permit overlap between cells because the cost of recalculating the estimated wire length for a new solution can be reduced. If overlap were not permitted, the overlap incurred by moving or exchanging cells would have to be removed by shifting many cells. As a result, the estimated wire length would have to be recalculated with respect to all of the nets connected to these shifted cells. In the lower temperature region, cells are never overlap, a new solution can be re-estimated only by calculating the change in total estimated wire length, because the two penalties don't change in this case.

The block length penalty estimates the difference between ideal and real block length. It is desirable to have cell blocks of a uniform length.

When the solutions are exchanged between the two temperature regions, the overlap between cells in the higher temperature region is removed as the solution is passed to the lower one.

3.5.3 Load distribution and solution exchange between adjacent temperatures

In PSA, each SA process is assigned to a separate processor, because executions at each temperature are highly independent and the amount of execution is nearly equal.

When we try to implement the exchange mechanism of the solutions, the natural way may be to exchange the solutions between the processors. But, when an MIMD machine like Multi-PSI is used, the exchange of large placement data between processors incurs a large communication overhead. So, solutions exchange between adjacent temperatures should be done by exchanging temperature values between processors.

Processors with adjacent temperatures hold a common variable and use this for communication. This is called a "stream" in KL1 [Chikayama *et al.* 1988] and is realized by an endless 'list'. These streams are also swapped between processors when the solutions between adjacent temperatures are exchanged.

3.5.4 Performance monitoring subsystem

The monitor displays the energy value of each SA process in real-time. It is useful to overview the entire state of the system performance. This energy graph is updated when adjacent temperatures are exchanged. As it displays the exchange in energy value in real time, it helps us to decide when to stop the execution. After several short time executions, we can decide the number of temperatures in the two regions and the highest temperature from the dispersion of the energy graph.

The monitoring subsystem is constructed on a Front End Process so that it does not incur an overhead in SA process execution. It is also possible to roll back the energy graph while SA processes are being executed.

Table 3: Number of temperatures .vs. quality of solution
inner-loop count = 20,000 times, frequency of exchange = 1/100

number of temp	8	16	32	63
estimated area[mm^2]	0.692	0.664	0.608	0.615
energy value	471120	436638	430320	424478

3.6 Experimental Results and Discussions

The MCNC benchmark data [MCNC 1990], consisting of 125 cells and 147 nets, was chosen for our measurements. In the initial placement, the value of energy was 911520 and a lower bound of the chip area was estimated as 1.372[mm^2].

The PSA was executed in 20,000 inner loops, with exchanges every hundred inner loops. 64 processors can be used on Multi-PSI. The number of temperatures is 63, the highest temperature is 10,000, the lowest is 20 and other temperatures are determined proportionally. 5 temperatures are assigned to the lower temperature region. The lower bound of the area of the final solution is estimated as 0.615 [mm^2], reduced by 56.0 % in comparison to the initial solution. The execution time was about 30 minutes and the final energy was 424478. Table 3 shows the system performance of the relation between the number of temperatures and quality of solutions. When the number of temperatures is 32, 16 or 8, with the other conditions the same, the lower bounds of the final chip are estimated as shown in Table 3.

When the number of temperatures is 63, the cooling schedules adopted by the final solution were as follows. The initial temperature was 3823, the highest temperature in the process was 4487, and the number of temperatures the solution passed was 53. We observed that 10 solutions out of the initial 63 had been disposed of at the lowest temperature. This indicates that the mechanism of the automatic cooling schedules actually worked as intended.

When the number of temperatures is 8, the results are even worse. If the dispersions in energy for each temperature are too far from each other, the chance of exchange gets small. So the automatic cooling schedules will not work as intended. As a result, the algorithm can not get out of the *local optimum*.

To get an effective cooling schedule, it is necessary to

find the appropriate value of the highest temperature so that it can reach the disorder state. It is also necessary to adjust the number of temperatures according to the size of the problem.

As a future work, we are planning to study the mechanism for deciding the initial temperatures assigned to each processor from the energy dispersion of the solutions.

From the viewpoint of system performance, more speed-up and improvement in the ability to treat larger amounts of benchmark data are needed as the next step.

4 Logic Simulator

4.1 Background

The logic simulator is used in order to verify not only the functions of designed circuits but also the timing of signal propagation. Parallel logic simulation is treated as a typical application of Parallel Discrete Event Simulation (PDES). PDES can be modeled so that several objects (state automata) change their states by communicating with each other. A message has information on the event whose occurrence time is stamped on the message (time-stamp). In logic simulation, an object corresponds to a gate and an event means the change of the signal value.

In PDES, the time-keeping mechanism is essential for efficient execution. The mechanisms broadly fall into three categories: synchronous mechanisms, conservative mechanisms and optimistic mechanisms. Their peculiar shortcomings are widely known; the synchronous mechanisms require global synchronization, the conservative mechanisms often deadlock and the optimistic mechanisms need rollback.

We are targeting an efficient logic simulator on PIM, which is a distributed memory MIMD machine. We adopted an optimistic mechanism, the Time Warp mechanism (TW), whose rollback process has been considered to be heavy. In practice, however, TW has neither been evaluated in detail nor compared with other mechanisms on MIMD machines.

We expected that TW would be suitable for logic simulator on large-scale MIMD machines with some devices that reduced the rollback overhead. Thus a local message scheduler, an antimessage reduction mechanism and a load distribution scheme were added to our system and evaluated. Furthermore, we made two other simulators using different time-keeping mechanisms and compared the mechanisms with TW.

4.2 Time Warp Mechanism

The Time Warp mechanism[Jefferson 1985] was proposed by D. R. Jefferson. In PDES using TW, 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 the 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.

4.3 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 simulator only treats gates, flip-flops and other functional blocks should be completely decomposed into gates.

4.4 Implementation

Since TW contains its peculiar overheads caused by the rollback processes, some devices for reducing overheads are needed for quick simulation. Furthermore, inter-PE communication overheads must be reduced because the simulator works on a distributed memory machine such as PIM.

For these purposes, a load distribution scheme, a local message scheduler and an antimessage reduction mechanism are included in our simulator. These are expected to reduce the overheads described above and might promote efficient execution of the simulator.

Each device is outlined below. Details are presented in [Matsumoto *et al.* 1992].

- Cascading-Oriented Partitioning

We propose the "Cascading-Oriented Partitioning" strategy for partitioning circuits to attain high-quality load distribution.

This scheme provides adequate partitioning solutions that satisfy these three requirements: load balancing, keeping inter-PE communication frequency low and deriving a lot of parallelism.

- Local Message Scheduler

During simulation, there are usually several messages to be evaluated in a PE. When the Time Warp mechanism is used, the bigger the time-stamp a message has, the more likely the message is to be rolled back. For this reason, appropriate message scheduling in each PE is needed for reducing rollback frequency.

- Antimessage Reduction

As long as messages are sent through the KL1 stream, messages arrive at their receiver in the same order as they are transmitted. In this environment, subsequent antimessages can be reduced. We adopted this optimization, expecting that it would reduce the rollback cost.

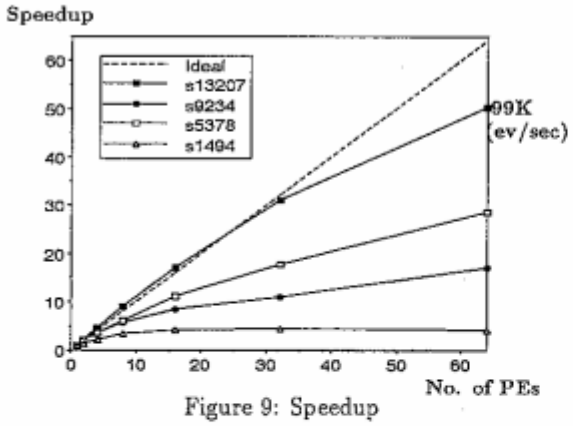

Figure 9: Speedup

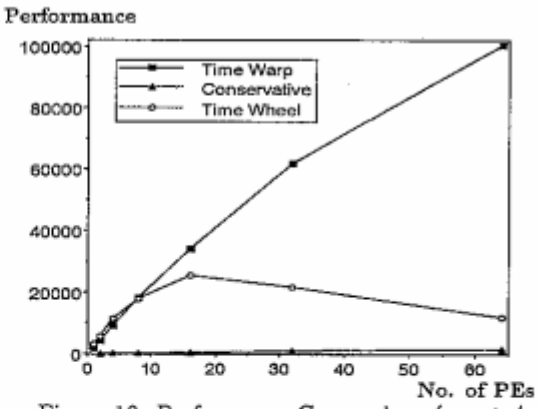

Figure 10: Performance Comparison (events/sec)

4.5 Measurements

We executed several experimental simulations on the Multi-PSI. Four sequential circuits, presented in IS-CAS'89, were simulated in our experiments.

Figure 9 shows the system performance when the circuits were simulated using various numbers of PEs. The best performance is also shown there. In the best case, very good speedup of 48-fold was attained using 64 PEs. Approximately 99K events/sec performance, fairly good for a full-software logic simulator, was also attained.

4.6 Comparison between Time-keeping Mechanisms

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

In the synchronous mechanism, only messages with the same time-stamp can be evaluated simultaneously. Therefore, a time wheel residing in each PE must synchronize globally at every tick. On the other hand, the problem of deadlock should be resolved [Misra 1986, Soulé *et al.* 1989] in conservative mechanisms. Our simulator basically uses null messages to avoid deadlock. A mechanism for reducing unnecessary null messages is also added in order to improve performance.

Figure 10 compares system 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 was the cause of the poor performance.

This comparison substantiates that the Time Warp mechanism provides the most efficient simulation of the three mechanisms on distributed memory machines such as the Multi-PSI.

5 Concluding Remarks

This paper presented ICOT-developed parallel systems for routing, placement and logic simulation, and reported on their evaluation.

In the routing system, the router program was designed based on the concurrent objects model and congruent with the KL1 description was introduced. As a result, appreciably good speedup was attained and the quality of the solutions was high especially for large-scale data.

The parallel placement system is based on time-homogeneous SA, which realizes an automatic cooling schedule. The remarkable point of this system is that parallelization was applied not for the purpose of speedup, but to obtain high quality solutions.

The parallel logic simulator simply targeted quick execution. Absolutely good speedup was attained. The experimental results for three kinds of time-keeping mechanisms revealed that the Time Warp mechanism was the most efficient time-keeping mechanism on distributed memory machines.

These three systems are positive examples which support that PIM possesses high applicability to various practical problem domains as a general purpose parallel machine. Besides them, we are currently developing a hybrid layout system in which routing and placement are performed concurrently, improving interim solutions incrementally. These experiments, including the hybrid layout system, are just the preliminary experiments in the coming epoch of parallel machines, but they must be one of the most important and fundamental experiences for the future.

Acknowledgement

Valuable advice and suggestions were given by the members of PIC-WG, a working group in ICOT, during discussion of parallel LSI-CAD. The authors gratefully thank them. Data for the evaluation of our systems were recommended and given by NTT Co., Hitachi Ltd. and Fujitsu Ltd. We also thank these companies.

References

[Brouwer 1990] R. J. Brouwer and P. Banerjee. PHIGURE : A Parallel Hierarchical Global Router. In *Proc. 27th Design Automation Conf.*, 1990. pp. 650-653.

[Chikayama *et al.* 1988] T. Chikayama, H. Sato and T. Miyazaki. Overview of the parallel inference machine operating system (PIMOS). In *Proceedings of International Conference on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988. pp. 230-251.

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

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

[Kawamura *et al.* 1990] K. Kawamura, T. Shindo, H. Miwatari and Y. Ohki. Touch and Cross Router. In *Proc. IEEE ICCAD90*, 1990. pp. 56-59.

[Kimura *et al.* 1991] K. Kimura and K. Taki. Time-homogeneous Parallel Annealing Algorithm. In *Proc. IMACS'91*, 1991. pp. 827-828.

[Kirkpatrick *et al.* 1983] S. Kirkpatrick, C. D. Gellat and M. P. Vecci. Optimization by Simulated Annealing, *Science*, Vol.220, No.4598, 1983. pp. 671-681.

[Kitazawa 1985] H. Kitazawa. A Line Search Algorithm with High Wireability For Custom VLSI Design, In *Proc. ISCAS'85*, 1985. pp. 1035-1038.

[Matsumoto *et al.* 1992] Y. Matsumoto and K. Taki. Parallel logic Simulator based on Time Warp and its Evaluation. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, 1992.

[MCNC 1990] *Proc. International Workshop Layout Synthesis '90* Research Triangle Park, North Carolina, USA, May 8-11, 1990.

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

[Nair *et al.* 1982] R. Nair, S. J. Hong, S. Liles and R. Villani. Global Wiring on a Wire Routing Machine. In *Proc. 19th Design Automation Conf.*, 1982. pp. 224-231.

[Nakajima *et al.*1989] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa and T. Chikayama. Distributed Implementation of KL1 on the Multi-PSI/V2, In *Proc. 6th Int. Conf. on Logic Programming*, 1989. pp. 436-451.

[Olukotun *et al.* 1987] O. A. Olukotun and T. N. Mudge. A Preliminary Investigation into Parallel Routing on a Hypercube Computer, In *Proc. 24th Design Automation Conf.*, 1987. pp. 814-820.

[Rose 1988] J. Rose. Locusroute : A Parallel Global Router for Standard Cells, In *Proc. 25th Design Automation Conf.*, 1988. pp. 189-195.

[Sechen *et al.* 1985] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf Placement and Routing Package, *IEEE Journal of Solid-State Circuits*, Vol.SC-20, No.2, (1985), pp. 510-522.

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

[Suzuki *et al.* 1986] K. Suzuki, Y. Matsunaga, M. Tachibana and T. Ohtsuki. A Hardware Maze Router with Application to Interactive Rip-up and Reroute. *IEEE Trans. on CAD*, Vol.CAD-5, No.4, (1986), pp. 466-476.

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

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

[Watanabe *et al.* 1987] T. Watanabe, H. Kitazawa, Y. Sugiyama. A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor. *IEEE Trans. on CAD*, Vol.CAD-6, No.2, (1987), pp. 241-250.

[Won *et al.* 1987] Y. Won, S. Sahni and Y. El-Ziq. A Hardware Accelerator for Maze Routing. In *Proc. 24th Design Automation Conf.*, 1987. pp. 800-806.