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

Experimental Parallel Inference Software

Katsumi Nitta Kazuo Taki Nobuyuki Ichiyoshi

Seventh Research Laboratory
Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
{nitta,taki,ichiyoshi}@icot.or.jp

Abstract

As tools to develop large scale intelligent systems, ICOT has developed parallel inference machines PIMs, a parallel logic programming language KL1 and an operating system PIMOS. In order to evaluate the appropriateness of these tools to the development of practical intelligent systems, we have developed four application programs in KL1 — the LSI-CAD system, the Genome Analysis System, the Legal Reasoning System and the Go Playing Game System —, and cooperating manufacturers have developed eight application programs. They cover a wide range of knowledge processing techniques such as case-based reasoning, model-based reasoning, qualitative reasoning and machine learning.

To obtain high performance from each application program, we have developed parallel programming techniques such as concurrent algorithms and load balancing. Moreover, we analyzed the performance of parallel programming technology theoretically. The result forms good guidelines for the selection of parallel programming techniques.

We introduce each application program and the results of performance analysis, and discuss our experiences of parallel programming.

1 Introduction

As tools to develop knowledge processing systems, ICOT has developed an experimental parallel inference machine Multi-PSI and five models of parallel inference machine PIMs [Uchida *et al.* 1988] [Goto *et al.* 1988]. They are MIMD machines on which user's programs written in the parallel logic programming language KL1 can run in parallel [Chikayama 1992]. As KL1 is based on the theory of first order predicate logic, it is useful to represent human knowledge naturally and to formalize inference processes naturally. Therefore, we can develop large-scale intelligent systems easier by using PIMs and KL1. However, if we develop KL1 programs naively, we cannot obtain high performance because the performance can be affected by sequential bottlenecks and various parallelization overheads; Good parallel algorithms and load distribution techniques have to be developed. Moreover, to develop efficient parallel programs, we have to understand the characteristics of the KL1 language and the architectures of Multi-PSI and PIMs (Figure 1). As these parallel programming technologies are closely related, when we develop KL1 programs, we have to choose suitable techniques carefully. Therefore, we need guidelines for selecting suitable parallel programming techniques and for estimating the relation between data size, number of processors, and performance. To get such guidelines, in addition to developing application programs, we have to conduct theoretical analysis of parallel programming techniques.

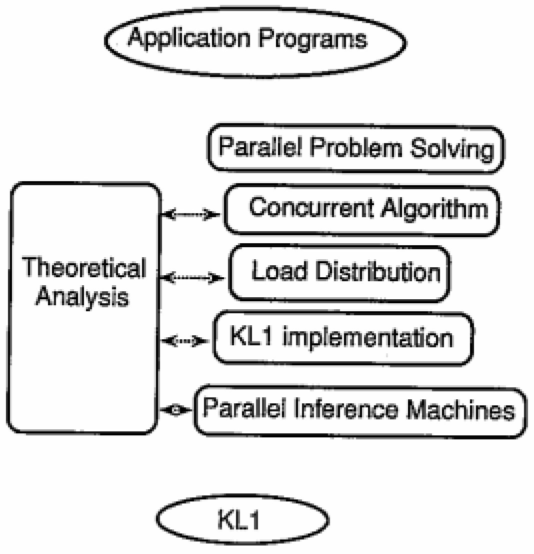


Figure 1: Parallel programming technologies

We have developed parallel application programs to achieve the following goals.

- Evaluation of applicability of PIMs to developing practical intelligent systems:

As PIMs solve problems efficiently by parallel inference, developing large scale systems using them is easier than using other computers. We wish to cultivate application fields and develop AI techniques where PIMs are effectively used.

- Development of Parallel Programming Techniques:
  By analyzing the behavior of application programs, we can extract parallel programming techniques to obtain high performance. A library of these techniques will help to develop new parallel programs.

In Section 2, we will give an overview of the research activities of the seventh research laboratory of ICOT. Section 3 presents application programs developed inside ICOT, and Section 4 presents application programs developed outside ICOT. In Section 5, the research activity in performance analysis is reported. In Section 6, we summarize the experiences of parallel program development.

2 Research Activities

As we explained in the previous section, to develop intelligent systems on PIMs, we have to cover a wide range of technologies from the knowledge of human experts to the features of hardware. To manage the various researches effectively, we organized the researchers of the seventh research laboratory into four *Application Groups* and one *Performance Analysis Group* (Figure 2). The roles of the *Application Groups* and the *Performance Analysis Group* are to develop specific application programs, and to give guidelines on parallel programming techniques by analyzing the behavior of KL1 programs theoretically.

Following are the researches of the Application Groups. To acquire knowledge from human experts effectively, these groups established four working groups: parallel IC CAD (PIC), genetic information processing (GIP), advanced design system (ADS), and knowledge architecture (KAR).

- LSI-CAD System:
  The LSI design process consists of several stages, such as architecture design, function design, logic design, micro program design, logic simulation and layout design. This group has developed the following two systems.
  - Logic Simulator
  - LSI Layout Systems
- Genome Analysis System:
  One of the most important targets of genome analysis is to interpret the meanings of protein sequences. This group has developed the following systems.
  - Protein Sequence Analysis System
  - Protein Folding Simulation Program
  - Protein Structure Analysis Program
- Legal Reasoning System:
  The difficulty of legal reasoning stems from the ambiguity of legal concepts. To deal with ambiguous concepts, this group has developed a legal reasoning system with a rule-based engine and a case-based engine.
- Go Playing Game System:
  The game of *go* is a traditional Japanese board game. This group has developed a parallel *go* playing game system.

In the next section, we will present an overview of each system.

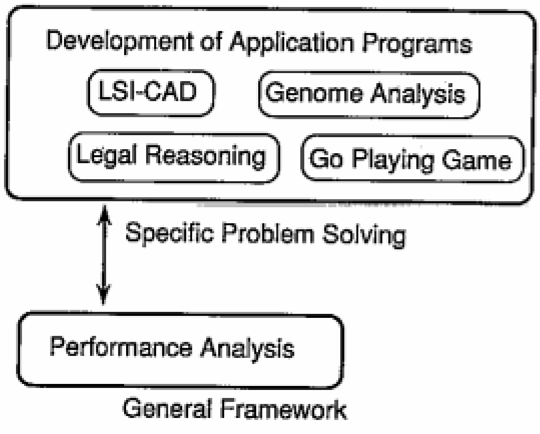


Figure 2: Research groups

Besides the above application programs, cooperating manufacturers have developed knowledge processing systems in order to evaluate the appropriateness of PIMs to these fields.

- Co-HLEX: Co-operative Recursive LSI Layout Problem Solver
  (hierarchical and cooperative problem solving)
- Cooperative Logic design Expert System on a Multi-Processor
  (assumption-based reasoning, cooperative problem solving)
- Case-based circuit design system
  (case-based reasoning)
- High Level Synthesis by Parallel Rule-based Annealing
  (rule-based annealing)
- Design Supporting System based on Deep Reasoning
  (qualitative reasoning)

- A Diagnostic and Control Expert System Based on a Plant Model
  (model-based reasoning, qualitative reasoning)

- Adaptive Model-based Diagnostic System (model-based reasoning, machine learning)

- Motif Extraction System
  (genetic algorithm, machine learning)

These systems cover various knowledge processing systems such as CAD systems, diagnosis systems, and control systems. They are related to various AI techniques such as case-based reasoning, qualitative reasoning, model based reasoning, and machine learning.

We will introduce these systems in Section 4.

3 Overview of Application Programs (1)

3.1 Logic Simulator

3.1.1 Background

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

Parallel logic simulation is 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 the information of an event whose occurrence time is stamped on the message (time-stamp). Since messages should be received and evaluated in the time-stamp order by their destination objects, the time-keeping mechanism is essential for efficient execution of PDES. Several mechanisms have been proposed for PDES time-keeping, however, each has its own peculiar shortcomings.

We are targeting an efficient logic simulator on PIM, which is a distributed memory MIMD machine. We adopted the Time Warp mechanism (TW), which has been considered to contain a heavy overhead — a rollback process. In practice, however, TW has never been evaluated in detail on MIMD machines. We expected that TW would be a suitable 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.

3.1.2 Overview of Logic Simulator

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

The Time Warp mechanism (TW) [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 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.

Since TW contains its own peculiar overheads caused by the rollback processes, some device for reducing the overheads is 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 the efficient execution of the simulator.

Each device is outlined below.

- Cascading-Oriented Partitioning

We propose the Cascading-Oriented Partitioning strategy for partitioning circuits to attain high-quality load distribution (Figure 3).

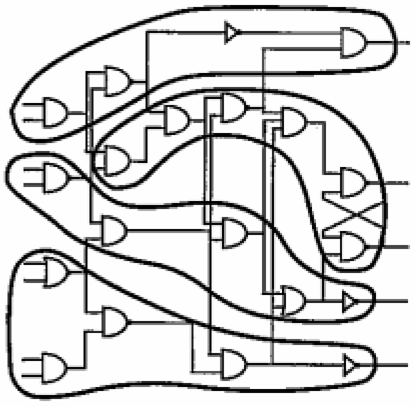


Figure 3: Cascading-Oriented Partitioning

This scheme provides adequate partitioning solutions that satisfy these three requirements: load balancing,

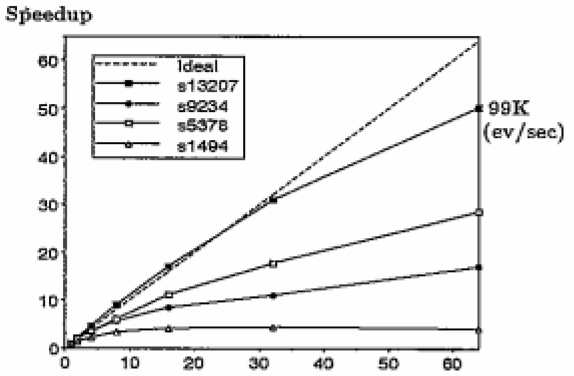


Figure 4: Speedup

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 TW is used, the bigger 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.

3.1.3 Result

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

Figure 4 shows the speedup figures 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. This experiment revealed that the Time Warp mechanism would be an efficient time-keeping mechanism.

In addition, we analyzed several factors which possibly limited speedup. Details are reported in [Matsumoto *et al.* 1992].

3.2 LSI Layout Systems

3.2.1 Background

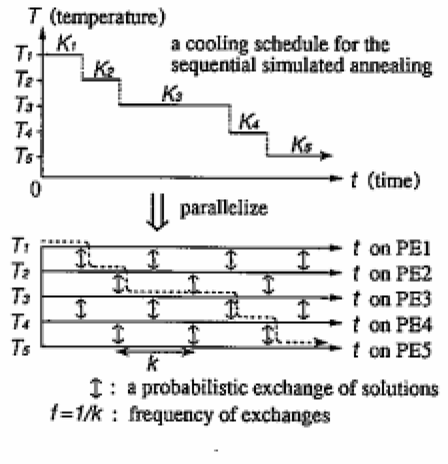
The LSI layout consists of two stages. The first is *placement*, which determines the physical position of the circuit components. The next is *routing*, which finds the paths between terminals of the circuit components. These are the most time-consuming stages in LSI design. Therefore high performance layout CAD systems lead to a shorter LSI design period.

Our aim is to study concurrent algorithms and load-balancing methodologies through design and development of parallel layout programs. Also, we are targeting the system to attain a high quality layout running on Multi-PSI and PIM.

3.2.2 Overview of LSI Layout System

(1) Placement System Our placement system is implemented for the standard cell type LSI without any macro blocks. The standard cells have uniform height and variant widths. These cells are assigned into multiple cell-blocks so as to minimize the chip area (strictly speaking, the total estimated wire length). The cell placement problem is a combinatorial optimization problem. As a powerful technique to solve such problems, simulated annealing (SA) is well-known. In order to execute SA efficiently, cooling schedules are important. In our placement system, the time-homogeneous parallel SA algorithm [Kimura *et al.* 1991], which constructs appropriate cooling schedules automatically, was adopted. Figure 5 shows an outline of this algorithm.

(2) Routing System Our routing system finds paths based on the look-ahead line search algorithm [Kitazawa 1985]. This algorithm provides high quality solutions in a short execution time, however, it was originally proposed with assumption of sequential execution. We introduced a new programming style based on a concurrent objects model in routing problems, and improved the basic algorithm to make it suitable for parallel execution. The concurrent objects model is expected to derive parallelism of small grain size. We designed the concurrent algorithm so that objects=processes corresponds to



T (temperature)
K1
K2
K3
K4
K5
T1
T2
T3
T4
T5
a cooling schedule for the sequential simulated annealing
0
t (time)
parallelize
t on PE1
t on PE2
t on PE3
t on PE4
t on PE5
k
↕ : a probabilistic exchange of solutions
f=1/k : frequency of exchanges

Figure 5: Time-homogeneous parallel simulated annealing

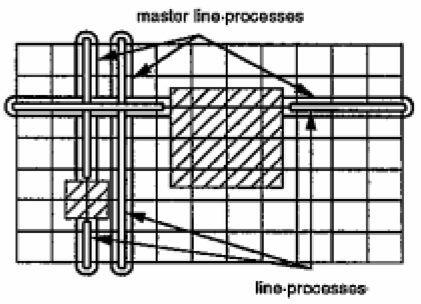


Figure 6: Master line processes and line processes

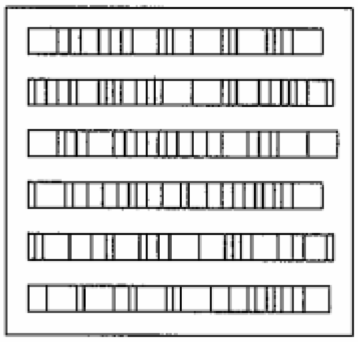


Figure 7: Placement results

every line segment on a routing grid. As in Figure 6, 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.

3.2.3 Result

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

When we executed our program for 30 minutes using 64 processors, the final energy was 424478 (the lower bound of the chip area is estimated as $0.615\ [mm^2]$).

Experimental results showed that final energy is reduced by 56.0 percent in comparison to the initial energy. Figure 7 shows the placement results.

(2) Routing We evaluate our router from the following three points of view using real LSI chip data. (1) Data size vs. Speedup, (2) Parallelism vs. Wiring Rate, (3) Comparison with a general purpose computer.

Figure 8 shows the system performance when the routing program was executed using various numbers of PEs. The size of DATA2 is larger than DATA1. In the best case, 24-fold speedup was attained using 64 PEs.

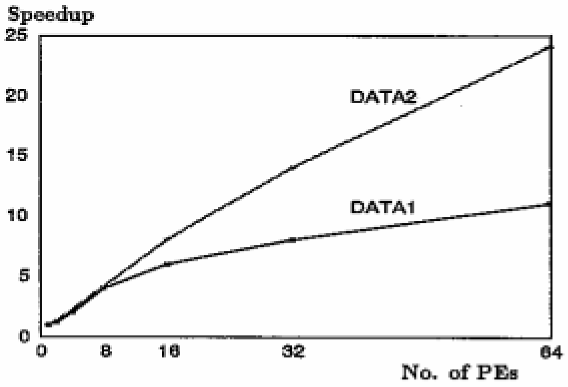


Figure 8: Speedup

Other experimental results are reported in [Date *et al.* 1992].

3.3 Protein Sequence Analysis Programs

3.3.1 Background

A primary structure of protein is a linear chain of amino acids. After a protein is created in the cell, it is folded and forms a complex structure.

The similarity analysis of protein sequences by the use of *multiple alignment* is an important technique for predicting the function and higher order structure of proteins and for drawing phylogenetic trees of creatures. An alignment is realized by lining the sequences with corresponding characters (amino acids) directly above one another as follows.

```
...YICSFADCGAAYNKNWKLQAHLC-KH...
...FPCKEEGCEKGFTSLHHLTRHFL-TH...
...FTCDSDFCDLRFTTKANMKKHFNRFH...
```

Until recently, multiple alignment was produced by hand by biologists. However, with the increasing rate of determination of protein sequences, computer assistance in multiple alignment is becoming indispensable.

It is well-known that once a similarity value between amino acids is given, the multiple alignment problem can be solved theoretically by Dynamic Programming (DP)[Needleman *et al.* 1970]. An alignment algorithm by DP method is the same as finding the shortest path in a network constructed by input sequences (Figure 9). N-way DP can align n sequences simultaneously and can derive the optimal alignment of these sequences.

One problem with DP is the incredible computational time it requires. N-way DP takes computational time in the order of the n-th power of the sequence length. To keep this expansion of computational time manageable, nearly all multiple alignment systems developed so far employ 2-way DP as a base and combine the results of 2-way DP to produce multiple alignment [Barton 1990].

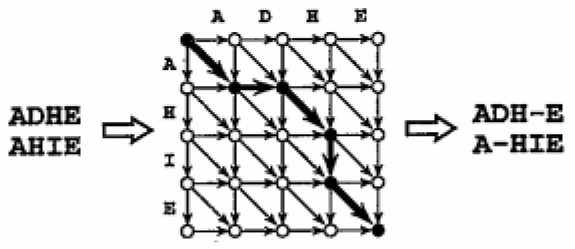


Figure 9: Alignment by 2-way DP

This class of alignment methods is good because of the small computational time required, but this is not sufficient to produce an alignment of sequences when their similarities are low.

3.3.2 Overview of Protein Sequence Analysis Programs

To produce multiple alignments of high-quality with small increases in computational time, we developed several multiple alignment systems. MASCOT (Multiple Alignment System developed by iCOT, see Figure 10) is a multiple alignment system based on DP [Hirosawa *et al.* 1991].

When protein sequences are given to MASCOT, MASCOT, firstly, classifies them into several clusters based on the similarities of sequences. Next, for each cluster, sequences are aligned from the nearest tree sequences using 3-way DP. Then, each intra-cluster alignment is refined by the simulated annealing method (Figure 5). Finally, each intra-cluster alignment is merged into a single alignment.

3.3.3 Result

Each module of MASCOT is described by the KL1 and is executed on the Multi-PSI. Though MASCOT requires more computation than conventional alignment systems due to the use of 3-way DP, parallel execution by the parallel inference machine [Ishikawa *et al.* 1991] can reduce the total time. Figure 11 shows the speedup of 3-way DP versus the number of processors used. 128 processors are about 64 times faster than a single processor.

MASCOT can produce a biologically valuable result. A resultant alignment shows clear consensus patterns in core alignments and discernible patterns in the alignment of each cluster. We think that this is a promising way to compare these kinds of pattern information with known motif information so that integrated information can be useful for attachment-alignment and intra-cluster alignment. We are now investigating how to use knowledge engineering to realize such an extension of MASCOT.

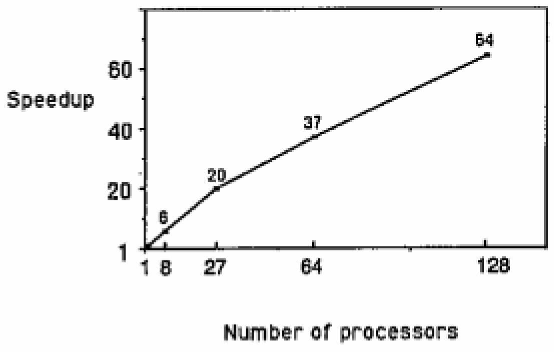


Figure 11: Speedup in 3-way dynamic programming

3.4 Folding Simulation Program

3.4.1 Background

Folding simulation simulates the process of protein formation from its stretched state to its native folded state by computer. This research topic has held the interest of biologists for a quarter of a century because while we can determine the order of amino acids in a sequence of protein extremely easily, it is very difficult to determine the structure of a protein. X-ray crystallography and NMR(Nuclear Magnetic Resonance) can be used to determine structure. However, both require plenty of time from months to a year.

One of the most frequently employed approximation methods is lattice representation [Ueda *et al.* 1978] [Skolnick and Kolinsky 1991], which restricts the position of amino acids in 3-dimensional lattice cells.

3.4.2 Overview of Folding Simulation Program

We applied *time homogeneous parallel (temperature parallel)* simulated annealing (Figure 5) to the folding simulation problem [Hirosawa *et al.* 1992]. Water-counting, which uses lattice representation (Figure 12) and employs only hydrophobic interaction, is introduced to formulate folding simulation as an optimization problem. In lattice cells, any place where protein is not present wiill be filled with water.

The energy to be minimized is expressed in the following formula.

$$E(Energy) = \sum_{m}^{sidechains}(Water\ Count_m - 1) \times Hydrophobicity_m$$

$$Water\ Count_m = \frac{Number\ of\ adjacent\ cells\ (of\ side\ chain)\ occupied\ by\ other\ amino\ acids}{The\ number\ of\ adjacent\ cells\ of\ the\ side\ chain}$$

The energy can be reduced both by increasing the amount of water around the hydrophilic amino acid and by reducing the amount of water around the hydrophobic amino acid. The minimization of energy has the effect of inviting hydrophobic amino acids toward the center

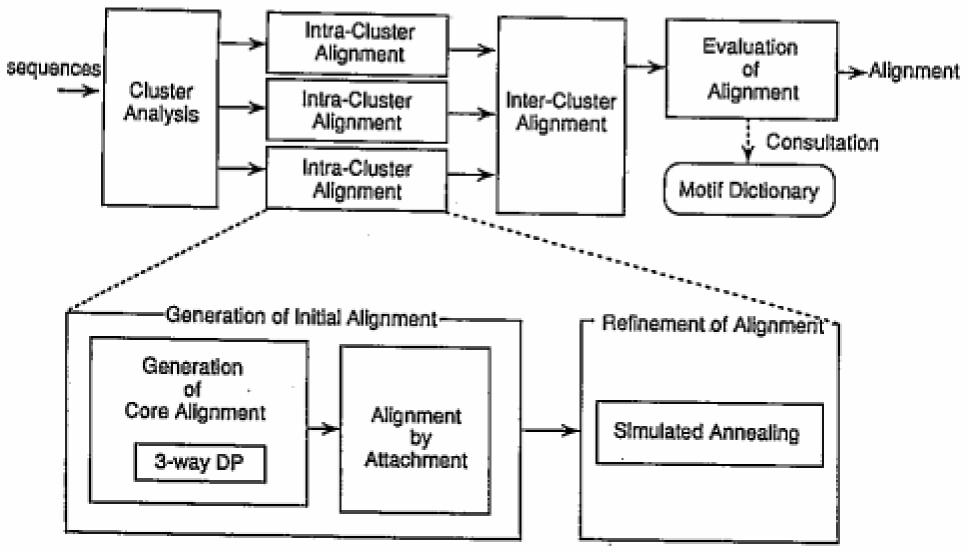


Figure 10: Multiple sequence alignment system: MASCOT

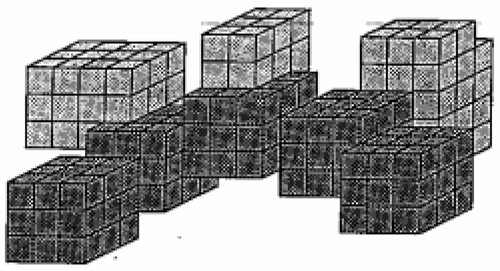


Figure 12: Representation of a section of protein: main chains(shaded) and side chains(unshaded)

of the protein where there is less water and to oust hydrophilic amino acids to the surface of the protein where water is abundant. These effects serve to produce protein that has a similar distribution of hydrophobic amino acids and hydrophilic amino acids within the protein structure.

3.4.3 Result

We selected flavodoxin, whose structure is known, as the protein to be simulated. This protein is of a medium size and has 138 amino acids. We ran the folding simulation program using temperature parallel SA on Multi-PSI using 20 processors over 10 days. This corresponds to 30,000 cycles. We also ran the folding simulation program using *simple parallel* SA in 30,000 cycles, also with 20 processors.

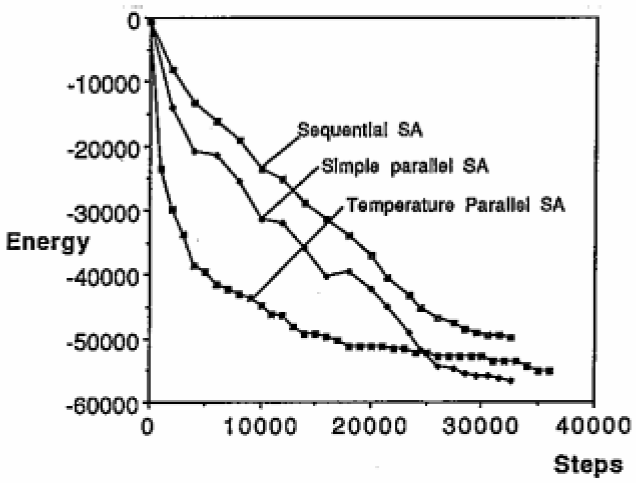


Figure 13: Energy history of folding simulation

We made the following observations from the energy history of simulation (Figure 13).

1. Two kinds of parallel SAs had better results within a fixed time than sequential SA. This is simply the effect of multiple processors.

2. Up to the middle stage of simulation, temperature parallel SA is always better than simple parallel SA. This is because temperature parallel SA can produce optimal solutions at that time.

3. Two kinds of parallel SAs have almost the same final energy value.

3.5 Protein Structure Analysis Programs

3.5.1 Background

One of the most important problems in the field of structural biology and biophysics is protein structure prediction. Structural biologists have proposed many methods to solve the structure prediction problem. Still, the accuracy of secondary structure prediction (i.e. to know the local feature of a protein structure), which seems to be the easiest part of protein structure prediction, is far below the biological demand.

3.5.2 Overview of Protein structure analysis programs

We plan to solve this difficult problem by a three-phase strategy. In the first phase, we should develop a effective method for representing the structure of protein. Secondly, we are to analyze the statistical relation between the representation and the sequence of a protein, and to obtain a statistic prediction method. Finally, we are planning to analyze which part of the prediction is statistically imprecise by logical consideration in order to know the limits of the statistical prediction method. We also plan to improve the prediction method by using logical knowledge gained from analysis. This plan should ensure that the parallel inference machine is used effectively.

At the moment, we are in the first phase, and have obtained a new way of representing the structure of protein produced by multi-variate analysis (Figure 14). The three dimensional distribution of the amino asid residues which are serial in a protein sequence is easily characterized by each standard deviation on the three main axes of the distribution. This gives us the local coordinates for analyzing the local structure.

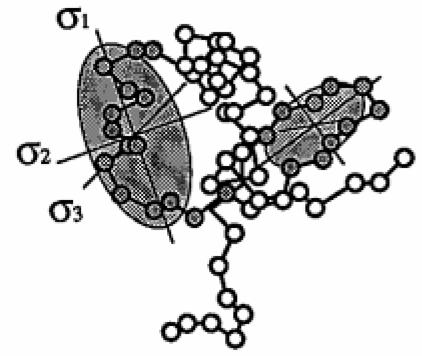


Figure 14: Main axes of the distribution of amino acid residues

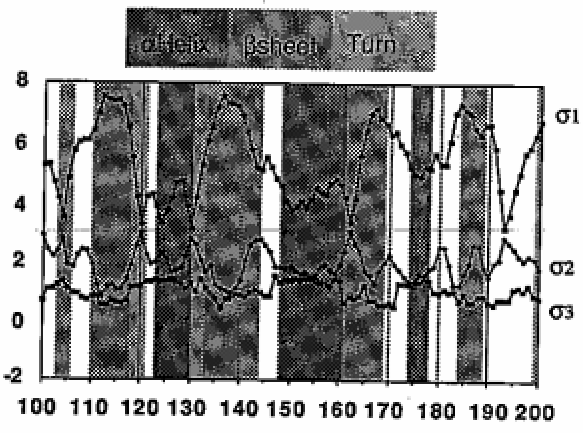


Figure 15: Spatial distribution of amino acids of protein sequence

3.5.3 Result

As the result, we found it possible to numerically represent the local structure of protein, and we can recognize its secondary structure from this new representation of protein. This numerical representation, which seems to be suitable for numerical operations such as regression analysis, may be quantized into a symbolic representation for logical or symbolic operations (Figure 15).

3.6 A Legal Reasoning System

3.6.1 Background

Legal knowledge consists of statutory laws and old cases. As a statutory law is a set of legal rules, inference by a statutory law is realized as rule-based reasoning. However, legal rules often contain legal predicates (legal concepts). Some legal concepts are ambiguous and their strict meanings are not fixed until the rules are applied to actual facts. To apply legal rules to actual facts, rule interpretation and matching between legal concepts and concrete facts are needed. To realize this, old cases are often referenced and their explanations are reused. Consequently, legal reasoning can be modeled as a mixed paradigm of rule-based reasoning and case-based reasoning.

However, there are some difficulties in developing a practical legal reasoning system. Firstly, as there are many legal rules and many old cases, it takes a long time to search for similar cases and to draw conclusions based on them. Secondly, to manage several inference engines, a complex mechanism to control inference is needed.

To solve these problems by parallel inference, we developed a legal reasoning system, HELIC-II, on the parallel inference machine.

3.6.2 Overview of the Legal Reasoning System

HELIC-II draws legal conclusions for a given case by referencing a statutory law and old cases and outputing them in the form of inference trees [Nitta *et al.* 1992].

HELIC-II consists of a rule-based engine and a case-based engine (Figure 16). The rule-based engine refers to legal rules and draws legal consequences logically. The case-based engine generates legal concepts from given facts by referring to similar old cases.

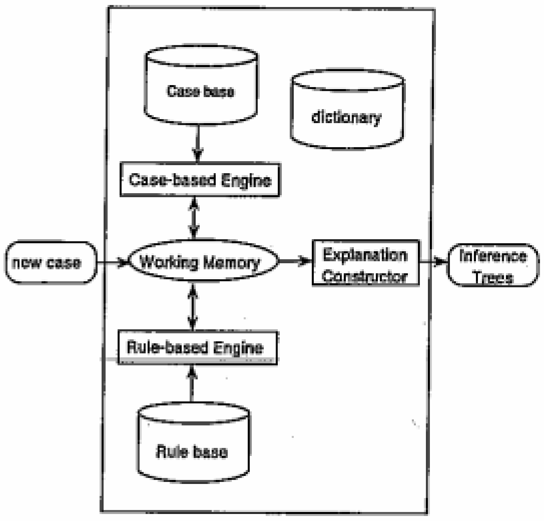


Figure 16: Architecture of HELIC-II

Rule-based inference As there are many legal rules, a fast rule-based engine is needed. Moreover, legal rules sometimes have exceptional rules, the rule-based engine has to be added some mechanism to handle nonmonotonic reasoning.

The rule-based engine of HELIC-II is based on the parallel theorem prover MGTP (Model Generation Theorem Prover) [Fujita *et al.* 1991]. Given a set of non-Horn clauses, MGTP generates models which satisfy all input clauses by parallel inference.

To use MGTP as a rule-based engine of legal rules, and to obtain high performance by pipeline effect, we added several extended functions to the original MGTP.

Case-based inference A judicial precedent (old case) consists of arguments by both sides and the opinion of judges and a final conclusion. We represent an old case as a *situation* and some *case rules.*

A *situation* contains informations on the occurrences of the case and represents a set of events/objects and their *temporal relations.* Arguments by both sides are represented as a set of *case rules.*

The function of the case-based engine is to generate legal concepts by referring to similar old cases. In the first stage, the engine searches for similar cases from the case base. Old cases are distributed to each processor(PE) of the Multi PSI, and similarities between the new case and old cases are evaluated in parallel. In the second stage, similarities between case rules of selected cases and the new case are measured using a Rete-like network (Figure 17), and new arguments are constructed.

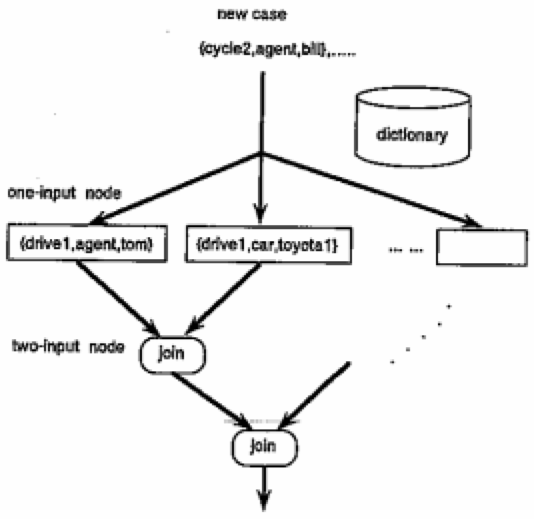


Figure 17: Rete-like network

3.6.3 Results

We observed that HELIC-II can solve several cases of the Penal Code. Figure 18 shows the speedup in the second stage of the case-based engine. We obtained more than 50-fold speedup using the 64PEs of the Multi-PSI.

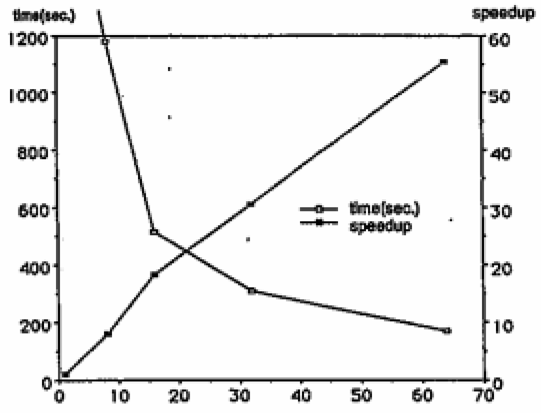


Figure 18: Performance of the case-based engine

3.7 Go Playing Game System "GOG"

3.7.1 Background

Go is a popular board game played traditionally in Japan, China, and Korea. Go is played using black and white stones and a 19 × 19 grid. The two players alternately place black and white stones on the grid intersections. The goal is to gain more secure territories than your opponent. It is a perfect information game.

Go has been a difficult game for computers to play. There have been no go-playing programs that match the ability of average human go-player. The difficulty of constructing a go-playing program comes mainly from the fact that (1) the branching factor of an average game tree is too large for brute force searches to be feasible, and (2) a simple and good board evaluation function does not exist.

As a go-playing program requires basic AI techniques such as searching, processing ambiguous patterns, exceptional processing, and cooperative problem solving, it is a suitable research subject for knowledge processing technologies.

We are trying to build a strong go program using the computing power of the parallel inference machines. We are aiming at the strength of GOG (GO Generation) with the ability of the average human player.

3.7.2 Overview of GOG

GOG has the following three features.

1. It simulates the thinking mechanism of a human player.
2. The large tasks are performed in parallel.
3. The new "flying corps" technique has been applied to improve the strength of GOG considerably while retaining its real-time response.

Simulating the Thinking Mechanism of a Human Player The process in which the GOG system determines its next moves comprises three stages (Figure 20). When the system receives the enemy's move, it first recognizes the board configuration. And then, it generates many candidate moves. It rates those moves and selects the one with the highest value as the next move.

- Board Recognition

  The raw data of the board configuration is simply the state of every board position, which is either (a) vacant, (b) occupied by a white stone, or (c) occupied by a black stone. Just like a human player, the system starts from the raw board data and successively makes higher-level data structures — stones, strings (a string is connected stones of the same color), groups (strings of the same color that are close

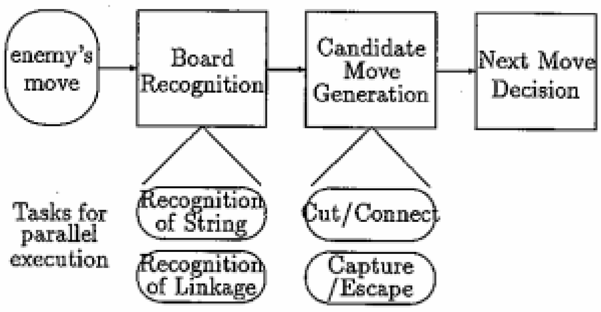


Figure 19: Outline of Process in The Parallel GOG

  to each other), families (loosely connected groups), etc. —, and then determines their attributes (potential value, area of surrounded territory, etc.) in the recognition phase.

- Candidate Move Generation

  The system has *candidate knowledge* which generates the coordinate and evaluation value of a candidate move. To decide the next move, many candidates are listed by executing tasks invoked from candidate knowledge. GOG has 12 kinds of the candidate knowledge (JOSEKI, Edge, DAME, Invasion, Spheres' Contact Point, Capture/Escape, Cut/connect, Enclose/Escape, etc.).

- Next Move Decision

  The local adjustment for candidates rearranges disharmonies between the different candidate knowledges. Then, the system sums the total proposed values of candidates at each point on the board. The system selects the one with the highest value as the next move and plays it.

Parallel Processing In GOG, one of the processors of the Multi-PSI serves as a manager processor, and the rest act as worker processors. The next move decision process is made on the manager processor, which also distributes tasks to worker processors.

When the system receives the enemy's move, it recognizes the board configuration and generates candidate moves. In those processes, it picks up large tasks such as local searches, which check whether a string to be captured or not, and dispatches the worker processors. The results are sent to the manager processor which, then, decides the next move based on those results.

Flying Corps To improve the strength of the system considerably while retaining its real-time response, we proposed the concept of *flying corps*.

This idea is to find the tasks which are important but don't have to be solved before the next move and to make flying corps processes execute these tasks. The system which incorporates the flying corps idea consists of main corps processes and flying corps processes (Figure 20). A flying corps process and a main corps process are assigned to the same processor. Main corps processes consist of a manager and workers and flying corps processes use the same manager and workers. Main corps processes execute necessary tasks to operate by go rules and tasks to maintain their strength.

Main corps processes have a higher priority than flying corps processes. Flying corps processes notify task completion to a flying corps manager process when the dispatched task is completed (which may be several moves after the initiation of the task). Whenever the main corps tasks are finished, the manager process of main corps will collect the results of finished tasks on flying corps processes. With those results and the results by main corps worker processes, the system decides on the next move. The time to decide the next move depends only on the main corps processes.

Flying corps processes execute these tasks independently from the immediate next move decision process (in main corps processes). When the opponent is thinking of the next move, the flying corps processes keep on running. When a local situation, which caused tasks for flying corps, will be changed by some later move, these tasks are aborted.

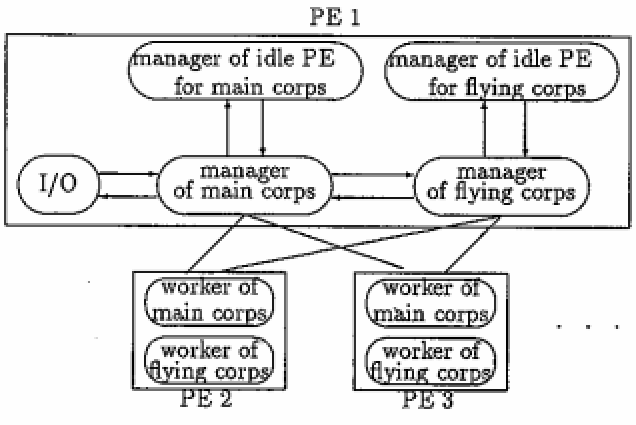


Figure 20: Configuration of System

3.7.3 Result

Table 1 shows the GOG's performance in parallel execution. From these results, the parallel execution shortens the processing time in go. The strength of GOG, including the flying corps idea, is now under evaluation.

We have been developing sequential GOG. The object is to test the new algorithm ideas of recognition, candidate knowledge, and next move decision. Last November, the sequential GOG and seven other computer go programs including last year's top five programs, participated in the tournament at the Game Playing System Workshop. The result of our sequential GOG was 2 wins and 3 loses. It shows that GOG is a top-class computer go-program. In human terms, the current system is stronger than an entry level human go player, but considerably weaker than an average player.

Table 1: Speedup in Parallel Execution

1st of final match, 13th Kisei tournament			
Stage	1 PE	4 PE	16 PE
30th move	1.0	3.3	5.1
90th move	1.0	3.4	5.3
180th move	1.0	3.7	7.5
5th of final match, 13th Meijin tournament			
Stage	1 PE	4 PE	16 PE
30th move	1.0	3.2	5.4
90th move	1.0	3.4	5.6
180th move	1.0	3.6	5.9

4 Overview of Application Programs (2)

4.1 Co-HLEX: Co-operative Recursive LSI Layout Problem Solver

LSI layout is one of the greatest problems requiring massive computation power. Also, the development and enhancement of a layout system consumes huge amounts of programmers labor. In the development of Co-HLEX, the development of a parallel algorithm as well as the possibility of more elegant program descriptions were investigated. The classical divide and conquer algorithm works well while subproblems correlate weakly. For LSI layout, this is not so. Neighboring modules should have abutting shapes and wires to avoid dead spaces. The concurrent co-operation mechanism among processes offered by FGCS paradigm might be an effective means to solve this problem.

An overview of Co-HLEX is given in Figure 21.

The problem-solving kernel is a quadtree-shaped process network called CMPN that generates a chip layout. Before layout generation, each node of CMPN contains circuit data including the module name, the module property, a list of net names connecting this module to others, and a list of sub-circuit names. After the layout is generated, layout data are added to each node: the template name (layoutframe) used to slice the node, the enveloping rectangle size, the list of adopted wiring pattern names for each net, etc.

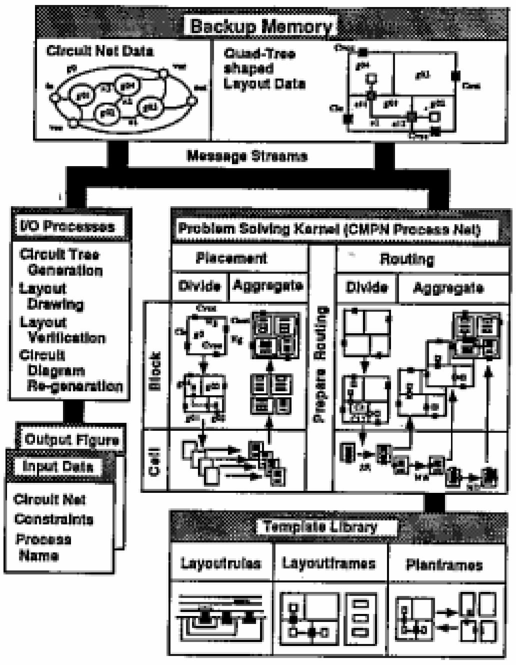


Figure 21: Overview of Co-HLEX

A recursive algorithm called HRCTL (Hierarchical Recursive Concurrent Theorem) was developed. This algorithm performs the layout by the following steps.

Placement A placement message containing a list of planned shape and planned peripheral connector placements is sent to the top node of CMPN from the co-ordination process. Then a set of recursive placement actions is performed by CMPN processes. In top-down processing, each non-terminal node is sliced by using an appropriate layoutframe picked up from the template library. Reaching the leaf node, an appropriate layoutframe defining the cell geometry is chosen. In bottom-up processing, the layouts of lower level children are aggregated to form a parent layout.

Wiring Non-terminal power supply nets Vcc and Vee are wired first, because they interfere with the wiring of signal nets. Non-terminal signal nets are then wired. Then, a set of recursive wiring actions is performed by CMPN. For each net of the non-terminal node, the existence range (CERW) of all the peripheral connectors of the net are first reduced, then an appropriate wiring pattern is selected from the wiring pattern list attached to the layoutframe chosen before. At each point where the chosen pattern crosses the sub-slice border line, an induced connector is introduced. This is used as a peripheral connector by the adjacent sub-slices in the subsequent recursion. Recursion terminates at each leaf node, with each CERW reduced to the magnitude of cell height or width. Lastly, the nets in cells are wired (SE-wiring, NW-wiring, and ND-wiring).

Layout experiments are conducted for bipolar-analog circuits with approximately 1000 modules. The resulting layout realized a compact module placement and wires free of useless bends. By runtime wire abutment cooperation, channel areas used by inter-module patch wires were avoided. This was useful for chip area reduction.

Co-HLEX has a time complexity of roughly $O(N)$, where N is the number of modules in the circuit, as contrasted to a time complexity of nearly $O(N^2)$ for traditional layout systems.

The Co-HLEX program has 1,000 lines in KL1, while traditional implementations typically have more than 100,000 lines of code. The recursive HRCTL algorithm and the modularized streamed-parallel computation model of KL1 both contributed to the size reduction.

4.2 Cooperative Logic Design Expert System on a Multiprocessor

One of the pressing problems of CAD systems is the lack of a means to iterate the cycle of evaluation and redesign until the design satisfies all constraints. Without it, it would be impossible to design a quality circuit with the desired characteristics (area and speed) by looking at the design from a global point of view.

co-LODEX is a cooperative logic design expert system on a multiprocessor, based on an evaluation-redesign mechanism using assumption-based reasoning [Maruyama 1988][Maruyama 1990]. In it, design alternatives are considered as assumptions and constraint violations are viewed as contradictions. Redesign is implemented as contradiction resolution. Justifications for constraint violations, nogood justifications (NJs), play a central role in the mechanism. co-LODEX divides the whole circuit to be designed into subcircuits in advance and designs each subcircuit on each processor to exploit parallel processing. Global evaluation-redesign takes place by processors exchanging design results or NJs. NJs received from other agents help narrow down the search space for an agent in the sense that new NJs made from received NJs enable the agent to prune the search space [Maruyama 1991]. That is the reason why we claim that co-LODEX is cooperative.

co-LODEX inputs a behavioral specification written in a hardware description language, a block diagram of the datapath, and constraints on area and speed. Constraints on area are expressed as inequalities in the gate count, and constraints on speed are expressed as inequalities in the propagation delay. co-LODEX outputs a CMOS standard call netlist that satisfies the constraints.

The resulting netlist can be input to an automatic place-and-route system for CMOS standard cells.

co-LODEX divides the whole circuit to be designed into subcircuits. Each subcircuit is designed by a design agent. Figure 22 shows the five subcircuits for a circuit that solves a second-order differential equation (DiffEQ) and the agents in charge.

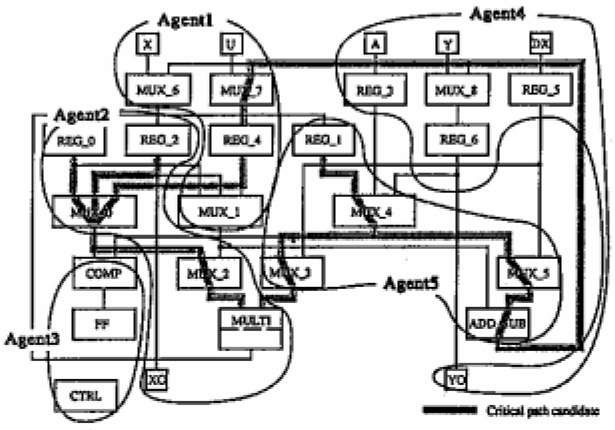


Figure 22: Sub-circuits and agents

Each design agent designs given functional blocks hierarchically using the top-down method. This method keeps splitting functional blocks and subblocks into sub-subblocks until all given blocks are implemented with CMOS standard cells.

Then it counts the number of gates and estimates delays to evaluate the implemented circuit against constraints on area and time. A design agent usually designs its subcircuit independently and in parallel with the other design agents. However, since the design results of the other agents are necessary for evaluation against global constraints, design agents exchange their results every time they design or redesign. A design agent redesigns when it detects a constraint violation for which it is responsible.

co-LODEX was implemented on Multi-PSI in KL1 [Minoda 1992]. Experimental results show that co-LODEX can efficiently carry out global optimization. Design agents correspond to processors on a one-to-one basis. We had one extra processor for distributing the functional blocks to other processors and making statistics. The relation between the number of design agents (1 to 15) and the speedup for a circuit with high uniformity is shown in Figure 23.

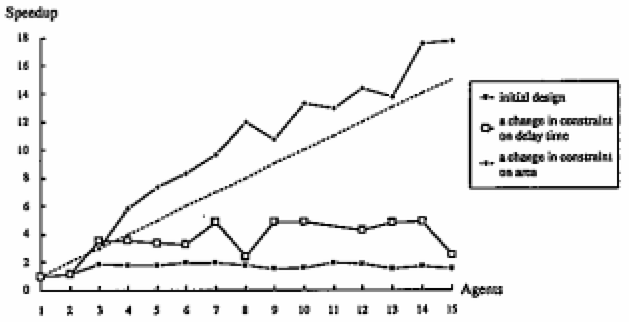


Figure 23: Relation between the number of agents and speedup

4.3 Case-based circuit design system

Recently, much attention has been paid to case-based reasoning (CBR) as a software technology for aquiring large amounts of knowledge easily and utilizing it efficiently. We have researched into a flexible and fast CBR mechanism through upper-level digital circuit design problems.

We suppose that novice designers, who have knowledge about primitive circuits but lack experiences in design, will use this system to solve application problems that are a little beyond the basic level. This system constructs block diagrams satisfying given specifications by retrieving similar precedent circuits, then modifying and combining them, based only on design cases and knowledge on primitive circuits.

This system features retrieving circuits whose functional structures are similar to the problem's and use a Structure Mapping Engine (SME) [Falkenhainer 86] as a case-retriever.

SME can extract cases structurally similar to the given problem, if higher order relations in given structures are the same between the case and the problem, even if the lower relations and entities are not same. In this system, SME evaluates the similarity of functional hierarchy trees. It, also, evaluates the descriptions of the circuit block functions that represent the hierarchical relations between the primary function and secondary functions that are necessary to realize the primary function. Then, it retrieves the circuits which have the most similar functions as a whole even though the details may be different. For example, when designing a digital clock, SME retrieves a similar circuit which counts the amount of money from the case base, even if there is no digital clock circuit.

Figure 24 shows the configuration of this system. We describe the design process briefly below.

Firstly, *Analyzer* analyzes the input specs to create functional hierarchy trees along the data flow and detailed specs for the given problem. Secondly, *Retriever* retrieves the cases which have similar functional hierarchy trees to the problems with SME. Thirdly, *Adaptor* checks whether the detail specs are the same between

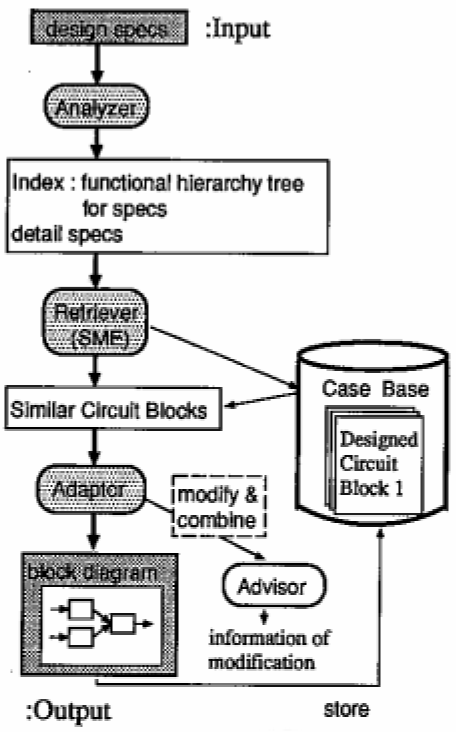


Figure 24: Configuration of case-based circuit design system

the retrieved case and the problem. When different, adaptor checks the possibility of modifying detail specs, then combines the retrieved cases which have confirmed adaptability to the given problem. In this phase, SME also predicts design failures and recovers from them, and those failure recoveries are reported via Advisor. Finally, the system outputs block diagrams corresponding to the combined cases. Users evaluate the output block diagram and, if it is suitable, the problem and the solution are stored in the case base as a new case.

We confirmed that non-stereotyped circuits are actually designed with this approach in mind; i.e. a digital clock with the additional function of sensing temperature can be an air-conditioner performance monitor.

Through experiments we also confirmed the effectiveness of the CBR method with SME. SME, however, has very high running costs because of its structural matching process which includes the combination problem. For this problem, we made SME programs parallel with the multi-level load balancer, and, with the 64 PE of Multi PSI, we obtained 10-fold speedup.

4.4 High Level Synthesis by Parallel Rule-based Annealing

Figure 25 describes the process flow of High Level Synthesis (HLS). LSI behavior descriptions written in a Pascal-like language (Paspec) are parsed and converted to a schedule table. The schedule table describes when each expression is executed and by which ALU. It corresponds to a datapath circuit. The problem finding the lowest cost configuration in the schedule table. The cost is the sum of the chip area and the execution speed. It is a typical combinatorial optimization problem (COP).

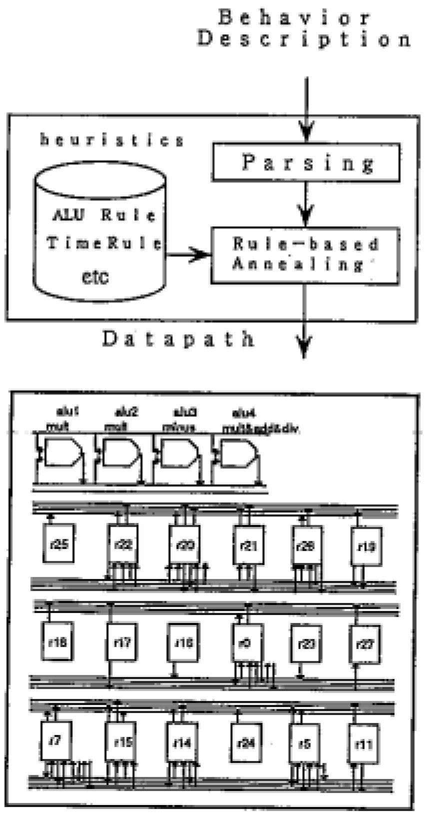


Figure 25: the process flow of HLS

Parallel Rule-Based Annealing Simulated annealing (SA) can be used to find a near global minimum in a COP, but it requires a huge number of iterations. Heuristic algorithms are faster, but the solutions are prone to capture in local minima. The *rule-based annealing (RA)* algorithm was developed, which has intermediate characteristics between the two. In each iteration step, the RA algorithm generates candidates of the next schedule table configuration by using not only random conversion but conversions using heuristic rules. The rule is selected probabilistically and the selection probability of the rule alters the temperature changes. The higher the acceptance rate of the candidate is, the higher the selection probability of the rule.

A parallel RA algorithm was then designed. The system consists of one master processor and a number of slave processors. Each processor runs the rule-based annealing independently at the same temperature, generating different sequences of configurations. At the beginning of annealing at a temperature, the master processor classifies the slave processors into a higher cost group and a lower cost group based on the cost of configuration. The annealing process continues until there is little difference in the cost distributions of the the two groups, at which time the equilibrium state is considered to have been reached. This contributes to the shortening of annealing steps at high temperatures. At low temperatures, configurations judged to be trapped in local minima are abandoned and are replaced by better configurations in other processors.

The parallel RA algorithm was implemented on a Multi-PSI with 16 processors. Figure 26 shows the experimental results. The RA algorithm was 4 times faster than the SA algorithm. The parallel RA was 8 times faster than the sequential RA. The effectiveness of the parallel RA algorithm was thus experimentally proven.

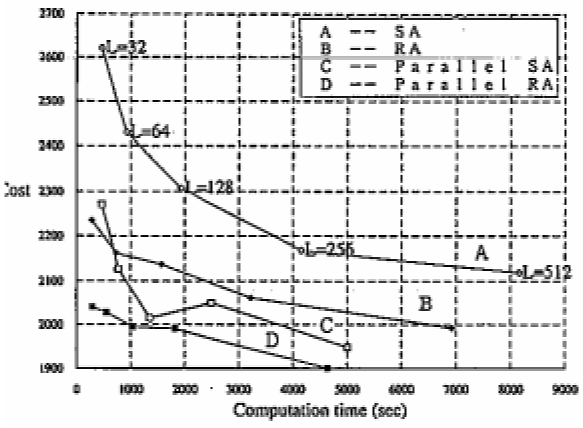


Figure 26: Experimental Result

4.5 Design Supporting System based on Deep Reasoning

In design, there are many cases in which a designer does not directly design a new device, but rather, changes or improves an old device. Sometimes a designer only changes the parameters of components in a device to satisfy the requirements. The designer, in such cases, knows the structure of the device, and needs to determine the new values of the components. This is common in electronic circuits. Desq (Design supporting system based on qualitative reasoning) determines valid ranges of the design decisions using qualitative reasoning.

Desq uses an envisioning mechanism, which, by using qualitative reasoning, determines all possible behaviors of a system. However, the qualitative reasoning of Desq is different from ordinary qualitative reasoning, because it can deal with quantities both qualitatively and quantitatively. Accordingly, Desq may be able to determine quantitative ranges, if parameters are given as quantitative values.

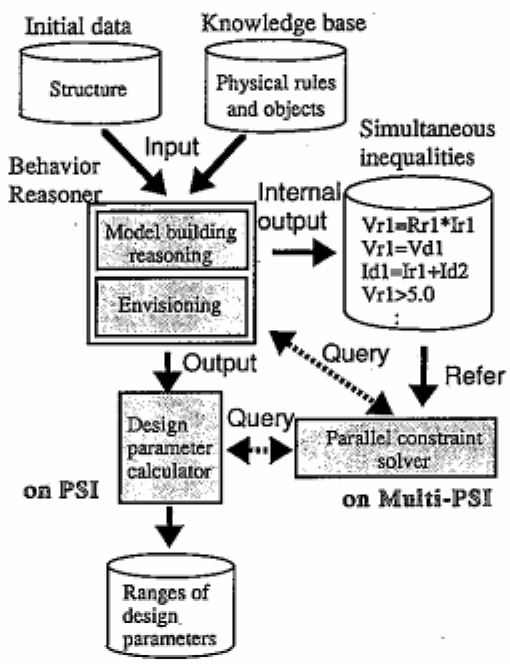


Figure 27: System organization

The system organization of Desq is shown in Figure 27. Desq consists of three subsystems:

Behavior reasoner

This subsystem is based on a qualitative reasoning system. Its model building reasoning part builds simultaneous inequalities from initial data using definitions of physical rules and objects. The simultaneous inequalities are a model of a target system. The envisioning part derives all possible behaviors.

Design parameter calculator

This subsystem calculates ranges of design parameters undefined in initial data.

Parallel constraint solver

This subsystem solves simultaneous inequalities. It is written in KL1 and is executed on a parallel inference machine.

Desq finds the valid ranges of design parameters as follows:

(1) Perform envisioning with design parameters whose values are undefined in initial data,

(2) Select preferable behaviors from possible behaviors found by envisioning,

(3) Calculate the ranges of the design parameters that give preferable behaviors.

As an experiment, Desq successfully determined the valid range of resistance Rb in the DTL circuit in Figure 28.

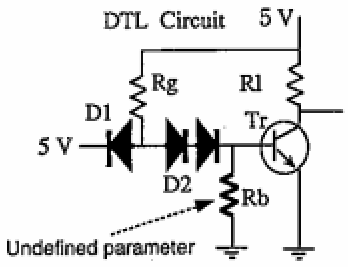


Figure 28: DTL circuit

4.6 A Diagnostic and Control Expert System Based on a Plant Model

Currently in the field of diagnosis and control of thermal power plants, the trend in systems is that the more intelligent and flexible they become, the more knowledge they need. As for knowledge, conventional diagnostic and control expert systems are based on heuristics stored a priori in knowledge bases. So, they cannot deal with unforeseen events when they occur in a plant. Unforeseen events are abnormal situations which were not expected when the plant was designed. To overcome this limitation, we have focused on model-based reasoning and developed a diagnostic and control expert system based on a plant model.

The system (Figure 29) consists of two subsystems: the *Shallow Inference Subsystem (SIS)* and the *Deep Inference Subsystem (DIS)*.

The *SIS* is a conventional plant control system based on heuristics, namely shallow knowledge for plant control. It selects and executes plant operations according to the heuristics stored in the knowledge base. The *Plant Monitor* detects occurrences of unforeseen events, and then activates the *DIS*. The *DIS* utilizes various kinds of models to realize the thought processes of a skilled human operator and to generate the knowledge for plant control to deal with unforeseen events. It consists of the following modules: the *Diagnosor*, the *Operation-Generator*, the *Precondition-Generator*, and the *Simulation-Verifier*. The *Diagnosor* utilizes the *Qualitative Causal Model* for plant process parameters to diagnose unforeseen events. The *Operation-Generator* generates the operations that deal with these unforeseen events. It utilizes the *Device Model* and the *Operation Principle Model*. The *Precondition-Generator* generates the preconditions of each operation generated by the *Operation-Generator*, and, as a result, generates rule-based knowledge for plant control. The *Simulation-Verifier* predicts the plant behavior that will be observed when the plant is operated according to the generated knowledge. It utilizes the *Dynamics Model*, verifies the knowledge using predicted plant behavior, and gives feedback to the *Operation-Generator*, if necessary.

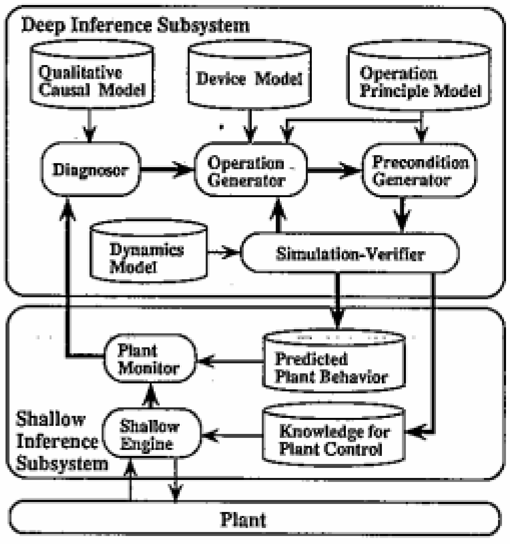


Figure 29: System Overview

The knowledge generated and verified by the *DIS* is transmitted to the *SIS*. The *SIS*, then, executes the plant operations accordingly, and, as a result, the unforeseen events should be taken care of.

We have implemented the system on Multi-PSI. To realize a rich experimental environment, we have also implemented a plant simulator on a mini-computer. Both computers are linked by a data transmission line. We have incorporated both a device and a dynamics model for each device of a thermal power plant (to a total of 78). We summarize the experimental results as follows.

- The *DIS* could generate plan control knowledge to deal with unforeseen events.
- The *SIS* executed plant operators according to the generated knowledge and could deal with unforeseen events.
- We have demonstrated a fivefold improvement in reasoning time by using Multi-PSI with 16 processor elements.

4.7 Adaptive Model-Based Diagnostic System

Though traditional rule-based diagnostic approaches that use symptom-failure association rules have been incorporated by many current diagnostic systems, they lack robustness. This is because they cannot deal with unexpected cases not covered by the rules in its knowledge base. On the other hand, model-based diagnostic systems that use the behavioral specification of a device are more robust than rule-based expert systems. However, in general, many tests are required to reach a conclusive decision because they lack the heuristic knowledge which human experts usually utilize. In order to solve this problem, a model-based diagnostic system has been developed which is adaptable because of its ability to learn from experience [Koseki *et al.* 1990].

This system consists of several modules as shown in Figure 30. The knowledge base consists of *design knowledge* and *experiential knowledge*. The design knowledge represents a correct model of the target device. It consists of a structural description which expresses component interconnections and a behavior description which expresses the behavior of each component. The experiential knowledge is expressed as a failure probability for each component. The *diagnosis module* utilizes those two kinds of knowledge.

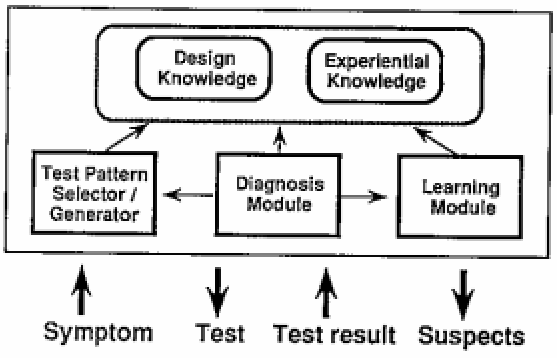


Figure 30: Structure of the System

Figure 31 shows the diagnosis flow of the system. The system keeps a set of suspected components as a suspect-list. It uses an *eliminate-not-suspected* strategy to reduce the number of suspects in the suspect-list, by repeating the test-and-eliminate cycle. It starts by getting an initial symptom. A symptom is represented as a set of target device input signals and an observed incorrect output signal. It calculates an initial suspect-list from the given initial symptoms. It performs model-based reasoning to obtain a suspect-list using a correct design model and an expected correct output signal. To obtain an expected correct output signal for the given inputs, the system carries out simulation using the correct design model.

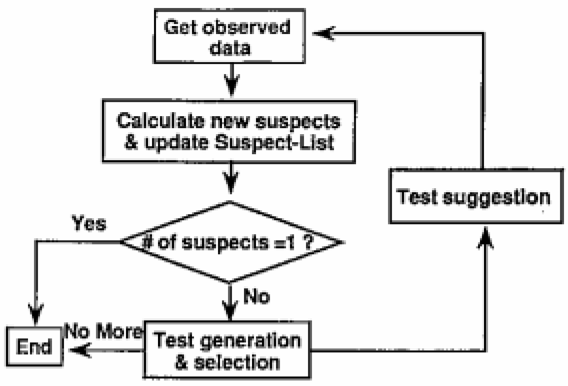

Get observed data
Calculate new suspects & update Suspect-List
Test suggestion
Yes
# of suspects =1 ?
No
End
No More
Test generation & selection

Figure 31: Diagnosis Flow

After obtaining the initial suspect-list, the system repeats a test-and-eliminate cycle, while the number of suspects is greater than one and an effective test exists. A set of tests is generated by the test pattern generator. Among the generated tests, the most cost effective is selected as the next test to be performed. The effectiveness is evaluated by using a minimum entropy technique that utilizes the fault probability distribution. The selected test is suggested and fed into the target device. By feeding the test into the target device, another set of observations are obtained as a test result and are used to eliminate the non-failure components.

Learning Mechanism The performance of the test selection mechanism relies on the preciseness of the presumed probability distribution of components. In order to estimate an appropriate probability distribution from a small amount of observation, the system acquires a presumption tree using minimum description length(MDL) criterion. A description length of a presumption tree is defined as the sum of the code length and the log-likelihood of the model. Using the constructed presumption tree, the probability distribution of future events can be presumed appropriately.

The algorithm is implemented in KL1 language on a parallel inference machine, Multi-PSI. The experimental results show that the 16 PE implementation is about 11 times as fast as the sequential one.

The performance of the adaptive diagnostic system (in terms of the required number of tests) was also examined. The target device was a packet exchange system and its model was comprised of about 70 components. The experimental results show that the number of required tests can be reduced by about 40% on average by using the learned knowledge.

4.8 Motif Extraction System

One of the important issues in genetic information processing is to find common patterns of sequences in the same category which give functional/structural attributes to proteins. The patterns are called *motifs*, in biological terms.

On Multi PSI, we have developed the motif extraction system shown in Figure 32. In this, a motif is represented by stochastic decision predicates and the optimal motif is searched for by the genetic algorithm with the minimum description length(MDL) principle.

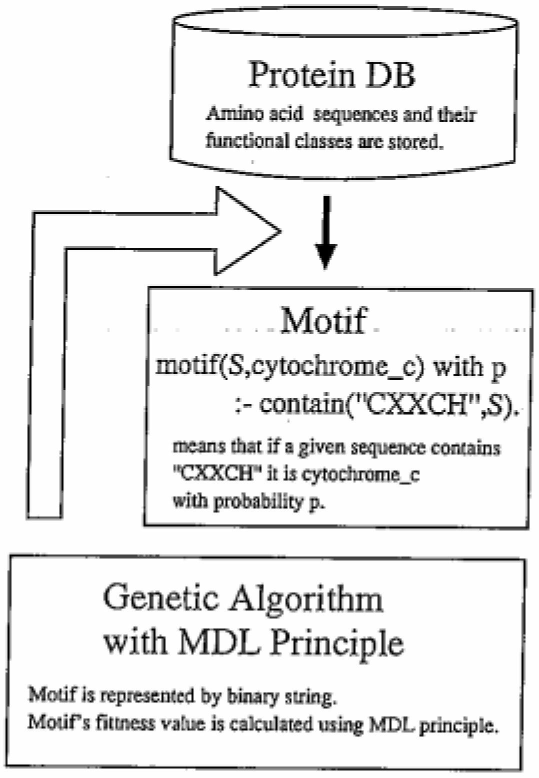

Protein DB
Amino acid sequences and their functional classes are stored.
Motif
motif(S,cytochrome_c) with p
:- contain("CXXCH",S).
means that if a given sequence contains "CXXCH" it is cytochrome_c with probability p.
Genetic Algorithm with MDL Principle
Motif is represented by binary string.
Motif's fittness value is calculated using MDL principle.

Figure 32: Motif Extraction System

Stochastic Decision Predicate It is difficult to express a motif as an exact symbolic pattern, so we employ the stochastic decision predicate as follows.

```
motif(S,cytochrome_c) with 129/225
  :- contain("CXXCH",S).
motif(S,others) with 8081/8084.
```

This example means that if S contains a subsequence matched to *"CXXCH"*, then S is cytochrome c with probability $\frac{129}{225}$, otherwise S is another protein with probability $\frac{8081}{8084}$.

Minimum Description Length Principle We employ the minimum description length(MDL) principle because it is effective in estimating a good probabilistic model for sample data, including uncertainty avoiding overfitting. The MDL principle suggests that the best stochastic decision predicate minimizes the following value.

> predicate description length + correctness description length

The value of the predicate description length indicates the predicate complexity(i.e. smaller values are better). The value of the correctness description length indicates the likelihood of the predicate(i.e. smaller values are better). Therefore, the MDL principle balances the trade-off between the complexity of motif representation and the likelihood of the predicate to sample data.

Genetic Algorithm The genetic algorithm is a probabilistic search algorithm which simulates the evolution process. We adopt it to search for the optimal stochastic motif, because there is a combinatorially explosive number of stochastic motifs and it takes enormous computation time to find the optimal stochastic motif by exhaustive searches.

The following procedures are performed in order to search for the optimal point of a given function f using the simple genetic algorithm.

1. Give a binary representation that ranges over the domain of the function f
2. Create an initial population which consists of a set of binary strings
3. Update the population repeatedly using selection, crossover, and mutation operators
4. Pick up the best binary string in the population after certain generations

We apply the simple genetic algorithm to search for the optimal motif representation. Each motif is represented by a 120-bit binary string, with each bit corresponding to one pattern (e.g. *"CXXCH"*). The 120-bit binary string represents the predicate whose condition part is the conjunction of the patterns containing the corresponding bits.

Table 2 is the result of applying the motif extraction system to Cytochrome c in the Protein Sequence Database of the National Biomedical Research Foundation. This table shows the extracted motifs and their description lengths. CL is a description length of motif complexity, PL is a description length of probabilities, and DL is a description length of motif correctness.

Table 2: cytochrome c

Motif	Compared	Matched	Correct
CXXCH	8309	225	129
others	8084	8084	8081

Description Length 286.894 (CL = 16.288, PL = 10.397, DL = 260.209)

5 Performance Analysis of Parallel Programs

5.1 Why Performance Analysis?

Along with the development of various application programs, we have been conducting a study of the performance of parallel programs in a more general framework. The main concern is the performance of parallel programs that solve large-scale knowledge information processing problems on large-scale parallel inference machines.

Parallel speedup comes from decomposing the whole problem into a number of subproblems and solving them in parallel. Ideally, a parallelized program would run p times faster on p processors than on one processor. There are, however, various overhead factors, such as load imbalance, communication overhead, and (possible) increases in the amount of computation. Knowledge processing type programs are "non-uniform" in (1) that the number and size of subproblems are rarely predictable, (2) that there can be random communication patterns between the subproblems, and (3) that the amount of total computation can depend on the execution order of subproblems. This makes load balancing, communication control, and scheduling important and nontrivial issues in designing parallel knowledge processing programs.

The overhead factors could make the effective performance obtained by actually running those programs far worse than the "peak performance" of the machine. The performance gap may not be just a constant factor loss (e.g., 30 % loss), but could widen as the number of processors increases. In fact, in poorly designed parallel programs, the effective-to-peak performance ratio can approach zero as the number of processors increases without limit.

If we could understand the behavior of the various overhead factors, we would be able to evaluate parallel programs, identify the most serious bottlenecks, and possibly, remove them. The ultimate goal is to push the horizon of the applicability of large-scale parallel inference machines into a wide variety of areas and problem instances.

5.2 Early Experiences

As the first programs to run on the experimental parallel inference machine Multi-PSI, four programs were developed to solve relatively simple problems. These were demonstrated at the FGCS'88 conference [Ichiyoshi 1989]. They are:

Packing Piece Puzzle (Pentomino)

A rectangular box and a collection of pieces with various shapes are given. The goal is to find all possible ways to pack the pieces into the box. The puzzle is often known as the Pentomino puzzle, when the pieces are all made up of 5 squares. The program does a top-down OR-parallel all solution search.

Shortest Path Problem

Given a graph, where each edge has an associated nonnegative cost, and a start node in the graph, the problem is to find the lowest cost path from the start node to every node in the graph (single-source shortest path problem). The program performs a distributed graph algorithm. We used square grid graphs with randomly generated edge costs.

Natural Language Parser

The problem is to construct all possible parse trees for an English sentence. The program is a PAX parser [Matsumoto 1987], which is essentially a bottom-up chart parsing algorithm. Processes represent chart entries, and are connected by message streams that reflect the data flow in the chart.

Tsumego Solver

A Tsumego problem is to the game of *go* what the checkmate problem is to the game of chess. The black stones surrounding the white stones try to capture the latter by suffocating them, while the white tries to survive. The problem is finding out the result assuming that black and white do their best. The result is (1) white is captured, (2) white survives, or (3) there is a tie. The program does a parallel alpha-beta search.

In the Pentomino program, the parallelism comes from concurrently searching different parts of the search tree. Since disjoint subtrees can be searched totally independently, there is no communication between search subtasks or speculative computation. Thus, load balancing is the key factor in parallel performance. In the first version, we implemented a dynamic load balancing mechanism and attained over 40-fold speedup using 64 processors. The program starts in a processor called the master, which expands the tree and generates search subtasks. Each of the worker processors requests the master processor for a subtask in a demand-driven fashion (i.e., it requests a subtask when it becomes idle). Later improvement of data structures and code tuning led to better sequential performance but lower parallel speedup. It was found that the subtask generation throughput of the master processor could not keep up with the subtask solution throughput of the worker processors. A multi-level subtask allocation scheme was introduced, resulting in 50 fold speedup on 64 processors [Furuichi *et al.* 1990].

The load balancing mechanism was separated from the program, and was released to other users as a utility. Several programs have used it. One of them is a parallel iterative deepening A* program for solving the Fifteen puzzle. Although the search tree is very unbalanced because of pruning with a heuristic function, it attained over 100 fold speedup on a 128-processor PIM/m [Wada *et al.* 1992].

The shortest path program has a lot of inter-*process* communication, but the communication is between neighboring vertices. A mapping that respects the locality of the original grid graph can keep the amount of inter-*processor* communication low. A simple mapping, in which the square graph was divided into as many subgraphs as there are processors, maximized locality. But the parallel speedup was poor, because the computation spread like a wavefront, making only some of the processors busy at any time during execution. By dividing the graph into smaller pieces and mapping a number of pieces from different parts of the graph, processor utilization was increased [Wada and Ichiyoshi 1990].

The natural language parser is a communication intensive program with a non-local communication pattern. The first static mapping of processes showed very little speedup. It was rewritten so that processes migrate to where the necessary data reside to reduce inter-processor communication. It almost halved the execution time [Susaki *et al.* 1989].

The Tsumego program did parallel alpha-beta searches up to the leaf nodes of the game tree. Sequential alpha-beta pruning can halve the effective branching factor of the game tree in the best cases. Simply searching different alternative moves in parallel loses much of this pruning effect. In other words, the parallel version might do a lot of redundant speculative computation. In the Tsumego program, the search tasks of candidate moves are given execution priorities according to the estimated value of the moves, so as to reduce the amount of speculative computation [Oki 1989].

Through the development of these programs, a number of techniques were developed for balancing the load, localizing communication, and reducing the amount of speculative computation.

5.3 Scalability Analysis

A deeper understanding of various overheads in parallel execution requires the construction of models and analysis of those models. The results form a robust core of insight into parallel performance.

The focus of the research was the scalability of parallel programs. Good parallel programs for utilizing large-scale parallel inference machines have performance that scales, i.e., the performance increases in accordance with the increase in the number of processors. For example, two-level load balancing is more scalable than single-level load balancing, because it can use more processors. But deciding how scalable a program is requires some analytical method.

As a measure of scalability, we chose the *isoefficiency function* proposed by Kumar and Rao [Kumar *et al.* 1988]. For a fixed problem instance, the efficiency of a parallel algorithm (the speedup divided by the number of processors) generally decreases as the number of processors increases. The efficiency can often be regained by increasing the problem size. The function $f(p)$ is defined as an isoefficiency function if the problem size (identified with the sequential runtime) has to increase as $f(p)$ to maintain a given constant efficiency E as the number of processors p increases. An isoefficiency function grows at least linearly as p increases (lest the subtask size allocated to each processor approaches zero). Due to various overheads, isoefficiency functions generally have strictly more than linear growth in p. A slow growth rate, such as $p \log p$, in the isoefficiency function would mean a desired efficiency can be obtained by running a problem with a relatively small problem size. On the other hand, a very rapid growth rate such as 2^p would indicate that only a very poor use of a large-scale parallel computer would be possible by running a problem with a realistic size.

On-demand load balancing was chosen first for analysis. Based on a probabilistic model and explicitly stated assumptions on the nature of the problem, the isoefficiency functions of single-level load balancing and multi-level load balancing were obtained. In a deterministic case (all subtasks have the same running time), the isoefficiency function for single-level load balancing is p^2, and that for two-level load balancing is $p^{3/2}$. The dependence of the isoefficiency functions on the variation in subtask sizes was also investigated, and it was found that if the subtask size is distributed according to an exponential distribution, a $\log p$ (respectively, $(\log p)^{3/2}$) factor is added to the isoefficiency function of single-level (respectively, two-level) load balancing. The details are found in [Kimura *et al.* 1991].

More recently, we studied the load balance of distributed hash tables. A distributed hash table is a parallelization of a sequential hash table; the table is divided into subtables of equal size, each one of which is allocated to each processor. A number of search operations for the table can be processed concurrently, resulting in increased throughput. The overhead comes mainly from load imbalance and communication overhead. By allocating an increasing number of buckets (= subtable size) to each processor, the load is expected to be improved. We set out to determine the necessary rate of increase of subtable size to maintain a good load balance. A very simple static load distribution model was defined and analyzed, and the isoefficiency function (with regard to load imbalance) was obtained [Ichiyoshi *et al.* 1992]. It was found that a relatively moderate growth in subtable size q ($q = \omega((\log p)^2)$) is sufficient for the average load to approach perfect balance. This means that the distributed hash table is a data structure that can exploit the computational power of highly parallel computers with problems of a reasonable size.

5.4 Remaining Tasks

We have experimented with a few techniques for making better use of the computational power of large-scale parallel computers. We have also conducted a scalability analysis for particular instances of both dynamic and static load balancing. The analysis of various parallelizing overheads and the determination of their asymptotic characteristics gives insight into the nature of large-scale parallel processing, and guides us in the design of programs which run on large-scale parallel computers.

However, what we have done is a modest exploration of the new world of large-scale parallel computation. The analysis technique must be expanded to include communication overheads and speculative computation. Now that PIM machines with hundreds of processors have become operational, the results of asymptotic analysis can be compared to experimental data and their applicability can be evaluated.

6 Summary of Parallel Application Programs

We have introduced overviews on parallel application programs and results of performance analysis. We will summarize knowledge processing and parallel processing using PIMs/KL1.

(1) Knowledge Processing by PIM/KL1

We have developed parallel intelligent systems such as CAD systems, diagnosis systems, control systems, a game system, and so on. Knowledge technologies used in them are the newest, and these systems are valuable from viewpoint of AI applications, too. Usually, as these technologies need much computation time, it is impossible to solve large problems using sequential machines. Therefore, these systems are appropriate to evaluate effectiveness of parallel inference.

We have already been experienced in knowledge processing by sequential logic programming languages. Therefore, we have got accustomed to developing programs in KL1 in a short time. Generally, to develop parallel programs, programmers have to consider the synchronization of each modules. This is troublesome and often causes bugs. However, as KL1 has automated mechanisms to synchronize inferences, we were able to develop parallel programs in a relatively short period of time as follows.

Program	Size	man*month
Logic Simulator	8 k	3
Placement		
(KL1)	4 k	4
(ESP†)	8 k	4
Routing	4.9 k	2
Alignment by 3-DP	7.5 k	4
Alignment by SA	3.7 k	2
Folding Simulation	13.7 k	5
Legal Reasoning		
(Rule-based engine)	2.5 k	3
(Case-based engine)	2 k	6
Go Playing Game	11 k	10

†: An extended Prolog for system programming.

In those cases where the program didn't show high performance, we had to consider another process model in regards to granularity of parallelism. Therefore, we have to design the problem solution model in more detail than when developing it on sequential machines.

(2) Two types of Process Programming

The programming style of KL1 is different from that of sequential logic programming language. A typical programming style in KL1 is *process programming*. A *process* is an object which has *internal states* and *procedures* to manipulate those internal states. Each process is connected to other processes by *streams*. Communication is through these streams. A process structure can be easily realized in KL1 and many problem solving techniques can be modeled by process structures.

We observed that two types of KL1 process structure are used in application programs.

1. **Static process structure**

   The first type of process structure is a static one. In this, a process structure for problem solving is constructed, then, information is exchanged between processes. The process structure doesn't change until the given problem is solved. Most distributed algorithms have a static process structure. The majority of application programs belong to this type.

   For example, in the Logic Simulator, an electrical circuit is divided into sub circuits and each sub circuit is represented as a process (Figure 3). In the Protein Sequence Analysis System, two protein sequences are represented as a two dimensional network of KL1 processes (Figure 9). In the Legal Reasoning System, the lefthand side of a case rule is represented as a Rete-like network of KL1 processes (Figure 17). In co-LODEX, design agents are statically mapped onto processors (Figure 22).

2. **Dynamic process structure**

   The second type of process structure is a dynamic one. The process structure changes during computation. Typically, the toplevel process forks into subprocesses, each subprocess forks into subsubprocesses, and so on (Figure 33). Usually, this process structure corresponds to a search tree. Application programs such as Pentomino, Fifteen Puzzle and Tsumego belong to this type.

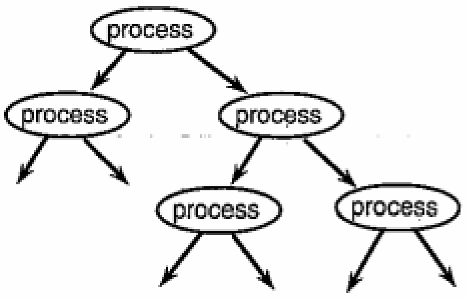


Figure 33: A search tree by a dynamic process structure

(3) New Paradigm for Parallel Algorithms

We developed new programming paradigms while designing parallel programs. Some of the parallel algorithms are not just parallelizations of sequential algorithms, but have desirable properties not present in the base algorithm.

In combinatorial optimization programs, a parallel simulated annealing (SA) algorithm (used in the LSI cell placement program and MASCOT), a parallel rule-based annealing (RA) algorithm (used in the High Level Synthesis System), and a parallel genetic algorithm (GA) (used in the Motif Extraction System) were designed.

The parallel SA algorithm is not just a parallel version of a sequential SA algorithm. By statically assigning temperatures to processors and allowing solutions to move from processor to processor, the solutions compete for lower temperature processors: a better solution has a high possibility of moving to a lower temperature. Thus, the programmer is freed from case-by-case tuning of temperature scheduling. The parallel SA algorithm is also time-homogeneous, an important consequence of which is it does not have the problem in sequential SA that the solution can be irreversibly trapped in a local minimum at a low temperature.

In the parallel RA algorithm, the distribution of the solution costs are monitored, and used to the judge whether or not the equilibrium state has been reached.

In the go-playing program, the flying corps idea suited for real-time problem solving was introduced. The task of the flying corps is to investigate the outcome of moves that could result in a potentially large gain (such as capturing a large opponent group or invasion of a large opponent territory) or loss. The investigation of a possibility may take much longer time than allowed in real-time move making and cannot be done by the *main corps*.

(4) Performance by Parallel Inference

Some application programs exhibited high performance by parallel execution, such as up to 100-fold speedup using 128 processors. Examples include the logic simulator (LS) (Figure 4), the legal reasoning system (LR) (Figure 18), and MGTP which is a theorem prover developed by the fifth research laboratory of ICOT[Fujita *et al.* 1991] [Hasegawa *et al.* 1992]. Understandably, these are the cases where there is a lot of parallelism and parallelization overheads are minimized. The logic simulator (LS), the legal reasoning system (LR), and MGTP have high parallelism coming from the data size (a large number of gates in the logic simulator and a large number of case rules in the legal reasoning system) or problem space size (MGTP). A good load balance was realized by static even data allocation (LS, LR), or by dynamic load allocation (MGTP). Either communication locality was preserved by process clustering (LS), or communication between independent subtasks is small (rule set division in LR or OR-parallel search in MGTP).

(5) Load Distribution Paradigm

In all our application programs, programs with a static process structure used a static load distribution, while programs with a dynamic process structure used semi-static or dynamic load distribution.

In a program with a static process structure, a good load balance can usually be obtained by assigning roughly the same number of processes to each processor. To reduce the communication overhead, it is desirable to respect the locality in the logical process structure. Thus, we first divide the processes into clusters of processes that are close to each other. Then, the clusters are mapped onto the processors. This direct cluster-to-processor mapping may not attain good load balance, since, at a given point in computation, only part of the process structure has a high level of computational activity. In such a case, it is better to divide the process structure into smaller clusters and map a number of clusters that are far apart from each other on one processor. This multiple mapping scheme is adopted in the shortest path program and the logic simulator. In the three dimensional DP matching program, a succession of alignment problems (sets of three protein sequences to align) are fed into the machine and the alignment is performed in a pipelined fashion, keeping most processors busy all the time.

In a program with a dynamic process structure, newly spawned processes can be allocated to processors with a light computational load to balance the load. To maintain low communication overhead, only a small number of processes are selected as candidates of load distribution. For example, in a tree search program, not all search substasks but only those at certain depths are chosen for interprocessor load allocation. The Pentomino puzzle solver, the Fifteen puzzle solver and the Tsumego solver use this on-demand dynamic load balancing scheme.

(6) Granularity of Parallelism

To obtain high performance by parallel processing, we have to consider the granularity of parallelism. If the size of each subtask is small, it is hard to obtain high performance, because parallelization overheads such as process switching and communication are serious. For example, in the first version of the Logic Simulator, the gates of the electrical circuit were represented as processes communicating with each other via streams. The performance of this version was not high because the task for each process was too small. The second version represented subcircuits as processes (Figure 3), and succeeded in improving the performance.

(7) Programming Environment

The first programs to run on the Multi-PSI were developed before the KL1 implementation on the machine had been built. The user wrote and debugged a program on the sequential PDSS (PIMOS development support system) on a standard hardware. The program was then ported to the the Multi-PSI, with the addition of load distribution pragmas. The only debugging facilities on the Multi-PSI were those developed for debugging the implementation itself, and it was not easy to debug application programs with those facilities. Gradually, the PIMOS operating system [Chikayama 1992] added debugging facilities such as an interactive tracing/spying facility, a static code checker that gives warnings on single-occurrence variables which are often simply misspelled, and a deadlock reporting facility. The deadlock reporting facility identifies perpetually suspended goals and, instead of printing out all of them (possibly very many), it displays only a goal that is most upstream in the data flow. It has been extremely helpful in locating the cause of a perpetual suspension (usually, the culprit is a producer process failing to instantiate the variable on which the reported goal is suspended).

Performance monitoring and gathering facility was later added (and is still being enhanced) [Aikawa 1992]. Post-mortem display of processor utilization along the time axis often clearly reveals that one processor is being a bottleneck at a particular phase of computation. The breakdown of processor time (into computing/communicating/idling) can give a hint on how the process structure might be changed to remove the bottleneck.

Sometimes knowledge of KL1 implementation is necessary to interpret the information provided by the facility to tune (sequential as well as parallel) performance. A similar situation exists in performance tuning of application programs on any computers, but the problem seems to be more serious in a parallel symbolic language like KL1. How to bridge the gap between the programmer's idea of KL1 and the underlying implementation remains a problem in performance debugging/tuning.

7 Conclusion

We introduced overviews of parallel application programs and research on performance analysis.

Application programs presented here contain interesting technologies from viewpoint of not only parallel processing but knowledge processing.

By developing various knowledge processing technologies in KL1 and measuring their performance, we showed that KL1 is a suitable language to realize parallel knowledge processing technologies and that they are executed quickly on PIM. Therefore, PIM and KL1 are appropriate tools to develop large scale intelligent systems.

Moreover, we have developed many parallel programming techniques to obtain high performance. We were able to observe their effects actually on the parallel inference machine. These experiences are summarized as guidelines for developing larger application systems.

In addition to developing application programs, the performance analysis group analyzed behaviors of parallel programs in a general framework. The results of performance analysis gave us useful information for selecting parallel programming techniques and for predicting their performance when the problem sizes are scaled up.

The parallel inference performances presented in this paper were measured on Multi-PSI or PIM/m. We need to compare and analyze the performances on different PIMs as future works. We would also like to develop more utility programs which will help us to develop parallel programs, such as a dynamic load balancer other than the multi-level load balancer.

Acknowledgement

The research and development of parallel application programs has been carried out by researchers of the seventh research laboratory and cooperating manufacturers with suggestions by members of the PIC, GIP, ADS and KAR working groups. We would like to acknowledge them and their efforts. We also thank Kazuhiro Fuchi, the director of ICOT, and Shunichi Uchida, the manager of the research department.

References

[Aikawa 1992] S. Aikawa, K. Mayumi, H. Kubo, F. Matsuzawa. ParaGraph: A Graphical Tuning Tool for Multiprocessor Ssytems. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992.

[Barton 1990] J. G. Barton, Protein Multiple Alignment and Flexible Pattern Matching. In *Methods in Enzymology, Vol.183* (1990), Academic Press, pp. 626-645.

[Chikayama 1992] Takashi Chikayama. KL1 and PIMOS. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992.

[Date *et al.* 1992] H. Date, Y. Matsumoto, M. Hoshi, H. Kato, K. Kimura and K. Taki. LSI-CAD Programs on Parallel Inference Machine. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992.

[de Kleer 1986] J. de Kleer. An Assumption-Based Truth Maintenance System, Artificial Intelligence 28, (1986), pp.127-162.

[Doyle 1979] J. Doyle. A Truth Maintenance System. Artificial Intelligence 24 (1986).

[Falkenhainer 86] B. Falkenhainer, K. D. Forbus, D. Gentner. The Structure-Mapping Engine. In *Proc. Fifth National Conference on Artifical Intelligence*, 1986.

[Fujita *et al.* 1991] H. Fujita, et. al. A Model Generation Therem Prover in KL1 Using a Ramified-Stack Algorithm. ICOT TR-606 1991.

[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)

[Furuichi *el al.* 1990] M. Furuichi, K. Taki, and N. Ichiyoshi. A multi-level load balancing scheme for or-parallel exhaustive search programs on the Multi-PSI. In *Proc. of PPoPP'90*, 1990, pp. 50–59.

[Goto *et al.* 1988] Atsuhiro Goto *et al.* Overview of the Parallel Inference Machine Architecture. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1988*, ICOT, Tokyo, 1988.

[Hasegawa *et al.* 1992] Hasegawa, R. *et al.* MGTP: A Parallel Theorem Prover Based on Lazy Model Generation. To appear in *Proc. CADE' (System Abstract)*, 1992.

[Hirosawa *et al.* 1991]
Hirosawa, M., Hoshida, M., Ishikawa, M. and T. Toya, T. Multiple Alignment System for Protein Sequences employing 3-dimensional Dynamic Programming. In *Proc. Genome Informatics Workshop II*, 1991 (in Japanese).

[Hirosawa *et al.* 1992] Hirosawa, H., Feldmann, R.J., Rawn, D., Ishikawa, M., Hoshida, M. and Micheals, G. Folding simulation using Temperature parallel Simulated Annealing. In *Proc. Int. Conf. on Fifth Generation Computer System 1992*, ICOT, Tokyo, 1992.

[Ichiyoshi 1989] N. Ichiyoshi. Parallel logic programming on the Multi-PSI. ICOT TR-487, 1989. (Presented at the Italian-Swedish-Japanese Workshop '90).

[Ichiyoshi *et al.* 1992] N. Ichiyoshi and K. Kimura. Asymptotic load balance of distributed hash tables. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, 1992.

[Ishikawa *et al.* 1991] Ishikawa,M., Hoshida,M., Hirosawa,M., Toya,T., Onizuka,K. and Nitta,K. (1991a) Protein Sequence Analysis by Parallel Inference Machine. *Information Processing Society of Japan, TR-FI-23-2*, (in Japanese).

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

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

[Kimura *et al.* 1991] K. Kimura and N. Ichiyoshi. Probabilistic analysis of the optimal efficiency of the multi-level dynamic load balancing scheme. In *Proc. Sixth Distributed Memory Computing Conference*, 1991, pp. 145–152.

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

[Koseki *et al.* 1990] Koseki, Y., Nakakuki, Y., and Tanaka, M., An adaptive model-Based diagnostic system, In *Proc. PRICAI'90,* Vol. 1 (1990), pp. 104-109.

[Kumar *et al.* 1988] V. Kumar, K. Ramesh, and V. N. Rao. Parallel best-first search of state space graphs: A summary of results. In *Proc. AAAI-88*, 1988, pp. 122–127.

[Maruyama 1988] F. Maruyama et al. co-LODEX: a cooperative expert system for logic design. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988, pp.1299-1306.

[Maruyama 1990] F. Maruyama et al. Logic Design System with Evaluation-Redesign Mechanism. Electronics and Communications in Japan, Part III: Fundamental Electronic Science, Vol. 73, No.5, Scripta Technica, Inc. (1990).

[Maruyama 1991] F. Maruyama et al. Solving Combinatorial Constraint Satisfaction and Optimization Problems Using Sufficient Conditions for Constraint Violation. In Proc. the Fourth Int. Symposium on Artificial Intelligence, 1991.

[Matsumoto 1987] Y. Matsumoto. A parallel parsing system for natural language analysis. In *Proc. Third International Conference on Logic Programming*, Lecturel Notes on Computer Science 225, Springer-Verlag, 1987, pp. 396–409.

[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 1992*, ICOT, Tokyo, 1992.

[Minoda 1992] Y. Minoda et al. A Cooperative Logic Design Expert System on a Multiprocessor. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*,ICOT,Tokyo, 1992.

[Nakakuki *et al.* 1990] Nakakuki, Y., Koseki, Y., and Tanaka, M., Inductive learning in probabilistic domain, In *Proc. AAAI-90,* Vol. 2 (1990), pp. 809-814.

[Needleman *et al.* 1970] Needleman,S.B. and Wunsch,C.D. A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins. *J. of Mol. Biol.*, 48 (1970), pp. 443-453.

[Nitta *et al.* 1992] K. Nitta et. al. HELIC-II: A Legal Reasoning System on the Parallel Inference Machine. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, ICOT, Tokyo, 1992.

[Oki 1989] H. Oki, K. Taki, S. Sei, and M. Furuichi. Implementation and evaluation of parallel Tsumego program on the Multi-PSI. In *Proc. the Joint Parallel Processing Symposium (JSPP'89)*, 1989, pp. 351–357. (In Japanese).

[Skolnick and Kolinsky 1991] Skolnick, J. and Kolinski,A., Dynamic Monte Carlo Simulation of a New Lattice Model of Globular Protein Folding, Structure and Dynamics, *Journal of Molecular Biology, Vol.221, No2*, pp.499-531.

[Susaki *et al.* 1989] K. Susaki, H. Sato, R. Sugimura, K. Akasaka, K. Taki, S. Yamazaki, and N. Hirota. Implementation and evaluation of parallel syntax analyzer PAX on the Multi-PSI. In *Proc. Joint Parallel Processing Symposium (JSPP'89)*, 1989, pp. 342–350. (In Japanese).

[Uchida *et al.* 1988] Shunichi Uchida et al. Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, ICOT, Tokyo, 1988.

[Ueda *et al.* 1978] Ueda, Y., Taketomi, H. and Go, N. (1978) Studies on protein folding, unfolding and fluctuations by computer simulation. A three dimensional lattice model of lysozyme. *Bilpolymers Vol.17* pp.1531-1548.

[Wada and Ichiyoshi 1990] K. Wada and N. Ichiyoshi. A study of mapping of locally message exchanging algorithms on a loosely-coupled multiprocessor. ICOT TR-587, 1990.

[Wada *et al.* 1992]
M. Wada, K. Rokusawa, and N. Ichiyoshi. Parallelization of iterative deepening A* algorithm and its implementation and performance measurement on PIM/m. To appear in Joint Symposium on Parallel Processing JSPP'92 (in Japanese).