TR-0852

# Parallel Inference System Research in the Japanese FGCS Project

by
T. Chikayama & R. Kiyohara (Mitsubishi)



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

# Parallel Inference System Research in the Japanese FGCS Project

Takashi Chikayama and Ryozo Kiyohara*

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

**Abstract.** The Fifth Generation Computer Systems (FGCS) project is a national project of Japan aiming at establishing the basic technology required for high performance knowledge information processing systems. One of its most important subprojects has been the research and development of the parallel inference system, aiming at establishing both hardware and software technologies for obtaining massive symbolic computation power through highly parallel processing.
This paper reports an overview of recent research and development on the parallel inference system, including hardware, basic software and experimental application software.

## 1 Introduction

The FGCS project is a national project of Japan, aiming at establishing the basic technology required for high performance knowledge information processing systems. The research and development principle throughout the project has been to adopt *logic* as the theoretical backbone of knowledge information processing and *parallel processing* as the key technology for obtaining high performance. Thus, one of its most important subproject has been the research and development of the parallel inference system, aiming at establishing both hardware and software technologies for massive symbolic computation power through highly parallel processing.

A concurrent logic language, KL1[UK1], was designed as the kernel language of the system to give the basis of both hardware and software technologies. As the hardware system, the parallel inference machine PIM[TK1] has been developed as a prototype to offer gigantic computation power to knowledge information processing systems. The language processor for KL1 has been developed to run efficiently on PIM. As the basic software system, an operating system PIMOS[CT1] has been developed to provide a comfortable development environment for parallel application software in KL1. PIMOS was also written in KL1. Various experimental application software systems have been developed upon PIMOS.

This paper reports an overview of recent research and development on the parallel inference system, including hardware, basic software and experimental application software.

The following sections are organized as follows. Section 2 describes the design principles of the parallel inference system. Section 3 gives an overview of the system. In Sect. 4, remarks on our experiences with the system are given. Finally, future research plans is described in Sect. 5.

## 2 Design Principles

There are two most important sets of technologies for high performance knowledge processing systems: One is technologies providing problem solving methods for knowledge information processing; the other is technologies for actually applying such methods, providing massive computational power and ease in programming. The parallel inference system subproject is aiming at establishing the latter, both in hardware and software, through logic-based parallel processing.

Several problems that did not exist with sequential processing arise with parallel processing. The most typical ones and our remedy for them are the following.

* The second author is currently at: Computer & Information Systems Laboratry, Mitsubishi Electric Corp., 5-1-1 Ofuna, Kamakura, Kanagawa 247, Japan.

*Programming language* Traditionally, parallel processing software has been written in sequential programming languages augmented with features for parallel processing. This often complicates software further. One of the most typical problems is synchronization between parallel computation activities. Synchronization failures are frequent source of bugs hard to fix.

To solve the problem, we adopted a concurrent logic programming language KL1[UK1], which was based on GHC[UK2]. KL1 is a born concurrent language, where concurrent computation is the default. Its automatic data-flow synchronization mechanism eliminates most of the synchronization problems.

*Hardware architecture* It was not clear which parallel processing architecture was most suited to knowledge information processing. It was quite difficult to evaluate many architectural ideas only through desk-top analysis. Software simulation is too time-consuming for evaluating with application software with practical complexity and size.

We thus decided to develop several (five, to be precise) models of PIM [TK1] with different processor and interprocessor connection architectures and evaluate them through experimentations with practical application systems.

*Software development environment* Tools originally designed for sequential programming do not always provide functionality required for debugging highly parallel software, even with extensions made afterwards. The same can be said about the operating system features, such as interfaces to the resource management mechanism and virtualized I/O devices. The original design relies so much on sequential processing that most of the extensions for parallel processing are only suited for small-scale parallelism.

An original operating system PIMOS (Parallel Inference Machine Operating System) [CT1] was thus developed to provide a comfortable software development environment for parallel application software.

The parallel inference system has an overall structure as shown in Fig. 1. A notable difference with conventional computer systems is that the operating system is built upon the level of the programming language processor. For efficient execution, highly parallel application software has to control parallel processing activities in the system, which usually was not needed in sequential or small scale parallel systems. The application layer and the operating system layer thus require the same primitives. We decided to provide their common basis, as the programming language KL1.

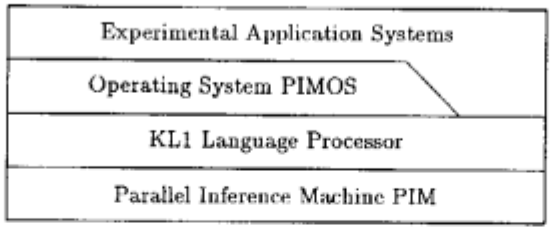


**Fig. 1.** Parallel Inference System

## 3 Overview of the System

### 3.1 The Kernel Language: KL1

This section describes the key features of KL1.

**Basic Mechanism** KL1 is a concurrent logic programming language based on GHC[UK2]. Its basic execution mechanism is common with other languages of the family, such as Concurrent Prolog[SE2], Parlog[CK1] or Janus[SA1].

KL1 programs consist of *clauses*, each of which corresponds to a logical axiom. Clauses that define a program has the following syntax.

*PredName*(*ArgList*, ...) :- *Guard* | *Body*.

Each part has the following operational meanings.

**PredName** gives the name of the predicate (or subroutine, if you like) for which this clause gives (a part of) the definition.

**ArgList** determines the correspondence of actual arguments given to the predicate and the variables written in the clause definition.

**Guard** specifies the condition needed to be satisfied to apply the clause. Any number of *goals*, i.e., invocation of predicates, can be written separated by commas and the condition is considered to be satisfied when all of them are satisfied. In the guard, only unifications and invocations of certain predicates defined in the language can be written.

**Body** specifies the action to be taken when the clause is selected. Like the guard, any number of goals can be given here and all the goals will be executed when the clause is selected. Unlike in guard, user-defined predicates can be invoked from the body, in addition to unifications and language-defined predicates.

Execution of KL1 programs proceeds roughly as follows.

1. First, the initial goal is the only member of a multi-set of goals called the *goal pool*.
2. Some of the goals in the goal pool are picked up.
3. Goals picked up are matched against *clauses* of the program.
4. If there is some clause with its *head* matching a goal and its *guard* is satisfied, the original goal will be reduced to goals in the *body* of the clause and the resultant new goals will be put back to the goal pool.
5. The steps 2 through 4 are repeated until the goal pool becomes empty.

The steps 2 through 4 can be done in parallel for many goals at a time. This is the source of concurrency in this language.

The most notable features of the concurrent logic programming languages are their side-effect free semantics and implicit data-flow synchronization mechanism. As no notion of *assignment* is in the language, value of a variable, once defined, will never change as the computation progresses. The data-flow synchronization mechanism assures that, whenever a decision is to be made for conditional execution, it is suspended automatically until the all the data required for the decision, such as operands to a comparison, get ready.

The combination of these features assures that there would never be synchronization problems such as follows.

- Overwriting a variable before its value is read.
- Reading a variable's value before it is set.

Programs in KL1 are usually organized using the object-oriented programming style [SE1]. Almost the whole PIMOS operating system and many of the application systems running on PIM are written in this way.

**Computation Mapping** KL1 provides only low level process distribution and priority-based scheduling features for controlling computation mapping. It seems that, at least with the status quo technology, no automatic load distribution schemes are universally effective to all kinds of algorithms. Our decision thus was to provide lower level primitives in the programming language level and make the software written in it responsible for computation mapping.

The primitives provided in KL1 are as follows. Note that, they are no more than pragmas that only suggest the language processor for better performance; they will not change the meaning of the programs.[2]

[2] To be precise, pragmas won't change the partial correctness of programs but certain diverging programs may be assured to stop through pragma specifications.

**Processor specification** Each body goal may have a processor specification which designates the processor on which to execute the goal.

**Priority specification** Each body goal may have a priority specification. Each goal has an integer priority associated with it.

Although process distribution is not, data distribution is made automatic. Data referenced by distributed processes are fetched from remote memory automatically on demand. The side-effect free semantics of KL1 allows copying of any data except for undefined variables without affecting the semantics. Executable codes are also distributed on demand, i.e., when a certain piece of code is needed on some processor and the code is not in the memory of that processor, it will be fetched from some other processor automatically. Memory areas occupied by executable code, as well as data structures, no longer needed are reclaimed with the garbage collection mechanism.

Several automatic mapping strategies have been developed for diverse problems using the above straightforward mechanism. Relatively universal ones are provided as libraries and used in many application software systems [FM2].

**Metalevel Control** With the basic semantics of the concurrent logic programming languages, all the goals in the system form one logical conjunction. This means, a failure or an exception in one of the goals makes the whole system fail. Also, there is no way to control execution of such goals. With this semantics, it is almost impossible to build a system that requires efficient metalevel control on computation activities, such as an operating system.

KL1 thus provides a metalevel execution control feature called "shoen".[3] A shoen is a group of goals. This group is used as the unit of metalevel control, namely initiation, interruption, resumption and abortion of execution. The shoen construct also provides exception handling and resource consumption control mechanisms.

A shoen has two communication streams as its interface: one, called the "control stream", directs inwards from outside of shoen for sending messages to control the execution; the other, called the "report stream", directs the reverse way for reporting events internal to the shoen, such as exceptions (Fig. 2).

PIMOS uses this shoen structure to construct a higher level notion of "task", which is the operating system level unit of resource management. Note that tasks are *not* a unit for parallel execution. There are usually many parallel activities within one task.

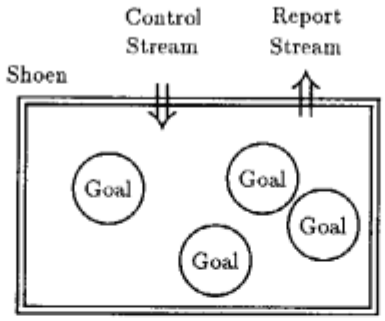


**Fig. 2.** Shoen and Related Streams

### 3.2 Parallel Inference Machine PIM

Five models of PIM have been built as listed in the Table 1. They differ in both their processor architecture and their interprocessor connection architecture. Evaluation of their architectures is one of the most important research topics currently going on.

[3] The Japanese word "shoen" roughly corresponds to the English word "manor".

**Table 1.** Five Models of PIM

| model name | PIM/p | PIM/c | PIM/m | PIM/i | PIM/k |
|---|---|---|---|---|---|
| processor | RISC+α | horiz. micro. | horiz. micro. | RISC | RISC |
| device | std. cell | gate array | cell base | std. cell | custom |
| # PE/cluster | 8 | 8 | 1 | 8 | 4 |
| inter-cluster | hypercube | crossbar | mesh | SCSI | bus |
| total # PE | 512 | 256 | 256 | 16 | 16 |

The PIM model M, which was completed first, performs 610 KLIPS[4] with one processor, providing the total peak performance of 150 MLIPS. Its average performance when application software is run upon PIMOS is usually somewhere between one third to one fifth of the peak performance. The PIM model P, which has 512 processors, marks similar total performance.

Some of the KL1 programs on PIM/m with 256 processors or PIM/p with 512 processors run about ten times faster than programs with basically the same (but sequential) algorithm on high-end workstations.

### 3.3 PIMOS

PIMOS is an operating and programming system for all models of parallel inference machines. Its overall structure is as shown in Fig. 3. Some of the characteristic submodules of PIMOS are as follows.

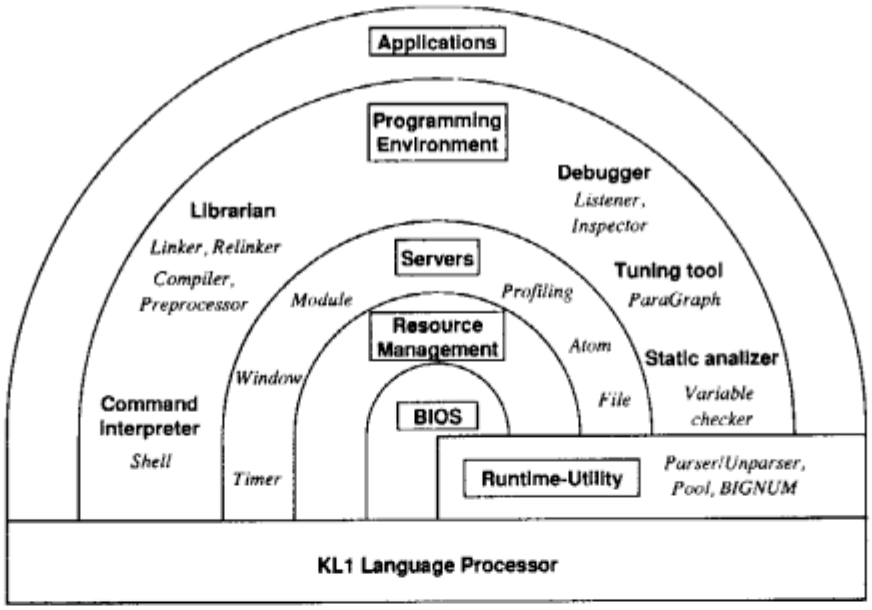


**Fig. 3.** The Structure of PIMOS

**BIOS** provides the most basic I/O through SCSI interface. KL1 provides a process model of the SCSI interface through built-in predicates.

[4] LIPS is an execution speed measure unit for logic programming languages, meaning Logical Inferences Per Second. Logical inference corresponds to a subroutine or a function call in procedural languages. It is customary to measure it with a list reversal program.

**Resource management** provides a layer for communication management. As all the I/O devices and tasks have message stream interface, management of such streams is resource management means on PIMOS[YH1].

Tasks, implemented using the shoen mechanism, can be nested with arbitrarily many levels. Thus, processes to control tasks form a tree structure, called the "resource tree". Processes to control communication with "servers" (described below) are also in this tree. By distributing such processes to the processor where user programs made request, the overhead of the operating system can be distributed without increasing the amount of communication.

**Servers** are processes to implement virtual devices upon physical devices[YH1]. Services provided by software systems, such as the database management system[KM1] or the librarian of the PIMOS described below, also have this server interface.

**Debugging Tools** provide features tailored for debugging parallel programs. It includes the "listener" with tracing and spying functions and the "inspector" to inspect data structures either statically or during their dynamic generation.

**ParaGraph** is a program tuning tool providing graphic display of execution profile information[AS1]. Such a visualization tool is quite powerful in tuning the performance of programs through changing mapping pragmas stated above.

Note that, as stated above, changing pragmas will not change the meaning of the programs. It only affects the performance. Data and executable code required are automatically distributed. This makes tuning of load distribution much easier.

**Librarian** is responsible for maintaining the correspondence with executable code modules and their names. It is implemented as a server.

Note that, the side-effect free nature of the KL1 doesn't allow even executable code to be overwritten. Updating some program module means generating a new one and changing the name correspondence. The older version may still be running somewhere in the system (possibly on the same processor). This scheme may seem inefficient but actually not, as updating executable code is not so frequent. The clean semantics, on the other hand, made users' understanding much easier.

### 3.4 Application Systems

Many experimental application systems have been developed and are running on PIMOS and PIM (currently, PIM/m and PIM/p systems are used mainly). In parallel with the development of independent application systems, performance analysis study from more general standpoint is also going on (see [KK1], for example).

The following is a list of some of such application systems. They all available as free software from ICOT.[5]

#### Symbolic Processing

**PIMOS** is the operating system for PIM and Multi-PSI. The entire source code is available as free software.

**Strategy management shell** which is an experimental multi-task operating system that automatically balances the load of multiple tasks dynamically.

#### Constraint Programming

**GDCC** is a parallel constraint logic programming system. It provides highly declarative, flexible and efficient constraint logic programming languages, dealing with various kinds of constraints including non-linear polynomial equations [TS1] .

**Consort** is a constraint solver for nonlinear inequalities.

**Dynamical programming (DP) system** is a logic programming system that controls symbolic inferences according to general heuristics based on dynamics.

[5] For details, please contact ifs@icot.or.jp.

### Knowledge Base

**Quixote** is a language system providing fundamental facilities for knowledge information processing, such as very high level knowledge representation and inferences [YH2].

**Kappa-P** is a parallel database management system based on a nested relational model [KM1].

### Theorem Proving

**MGTP** is a massively parallel, high performance bottom-up theorem prover on first order problems [FM1].

The prover tries to generate models satisfying a given axiom set. The system can be used in two ways. As a theorem prover, it will show that no models can satisfy a given axiom set augmented with the negation of the given theorem. As a constraint satisfaction system, models found by the system satisfying a given axiom set are answers to a constraint satisfaction problem. The system showed almost linear speed-up up to 512 processors.

### Problem Solving

**Andor-II** is a language system that allows simple description in a high level language which provides high reasoning power by parallel processing [TA1].

**Group problem solving system** is a multi-agent system to help task allocation to a group of people cooperatively solving a problem.

### Meta Reasoning

**Argus/V** is a system to analyze properties of Horn logic programs using first-order inference and induction.

**SME** is a system that extracts the correspondences and similarities of two systems described in tree forms, based on the structure-mapping theory.

### Natural Language Processing

**PAX** is a parallel bottom up parsing system for natural languages.

**Dulcinea** is an experimental system that generates natural language argument text to justify the given assertion [IT1].

**Laputa** is a natural language processing system based on concurrent cooperation of morphological, syntactic and semantic analysis [YS1].

### Genetic Information Processing

**Protein sequence alignment systems** have several variations, based on parallel dynamic programming [IM1], parallel simulated annealing [HM1] and through knowledge-based approach using Quixote mentioned above [HM2].

**Motif extraction system** extracts common motif patterns from amino acid sequences in the same protein category.

**Protein conformation prediction system** is for predicting 3-D structure of proteins from amino acid sequences based on geometrical stochastic reasoning [OK].

### Game Players

**GOG** is a "go" game (an oriental board game) player system [NK2].

### VLSI-CAD

**Parallel logic simulator** is a system to simulate VLSI circuits to verify their logical and timing specifications.[MY1]
This system adopts a virtual time algorithm, in which simulation done proceeds in a speculative way assuming input signal to be notified from other processors won't change. If such a change is notified later, the simulation will be rolled back. This strategy could extract much higher parallelism than non-speculative algorithm, obtaining 166-fold speed-up on 256 processors of PIM/m (534k events/sec). Although the parallel algorithm used is rather complex, it took only several man-month to build its first version.

**Cell placement system** automatically designs an optimal placement on standard cell LSI [DH1].

**LSI router** is a system to route paths between terminals of circuit modules on an LSI [DH1] [DH2].

**Case-based circuit design support system** helps designing new circuits using knowledge on similar precedential circuits.

**Co-Hlex** generates a circuit layout given a circuit, chip space and a set of module proximity conditions [WT1].

**Rodin** is a system for synthesizing LSI circuits given behavior specifications.

**Cooperative logic design expert system** generates netlists for CMOS standard cell LSI satisfying given area and speed constraints [MY2].

### Legal Reasoning

**Helic-II** is a parallel legal reasoning system referencing to both laws and precedents [NK1].
Reasoning from legal viewpoints is not simple inference process based only laws and regulations, as many of the words and phrases appearing in them are left undefined. Their intepretation is based on precedents. The system keeps two kinds of databases, one on laws and regulations and another on precedents. Given new cases are matched against precedent cases using higher level interpretaion (such as matching a taxi driver and a flight pilot as a traffic operator). The laws and regulations are applied after that, using abovementioned MGTP as the inference engine.

### Others

**Mendels zone** is a software synthesis system for concurrent programs that allow very high level specification.

**Desq** is a tool to determine ranges of design parameters using qualitative reasoning.

**Plant diagnosis and control expert system** generates if-then type rules for plant control from high level model description of the plant.

**Adaptive model-based diagnostic system** generates a probability ordered fault suspect list for given electric circuit with given symptom, estimating fault probability of each component from case records.

## 4 Through Our Experiences

This section summarize our experiences with building software systems in KL1 on the parallel inference machines.

### 4.1 Programming Ease

The automatic synchronization mechanism and fine grain concurrency of KL1 made programming much easier. The software productivity became far better than in sequential programming languages with baroque parallel processing extensions.

When we started developing the first version of PIMOS in 1987, there was no parallel KL1 language implementations available. Thus, the operating system was debugged on a sequential (pseudoparallel) implementation, which had only fixed scheduling strategy. When the system was ported to a prototype parallel machine, Multi-PSI, in 1988, we were ready for annoying synchronization bugs that will not reproduce themselves, although the automatic synchronization mechanism

of the language should avoid such problems in theory. The theory turned out to be the reality. We found almost no synchronization problems except for a small number of design problems in very high level, although the scheduling on the real parallel machine is quite different from the emulator. We knew this in theory, but actually experiencing this made us much more confident of the great merits of writing a system in a language with automatic data-flow synchronization. When PIMOS was ported to several other models of PIM systems later, each with its own scheduling strategy, we almost never encountered synchronization problems.

Most of the experimental application systems written for PIM were coded by programmers with no experience in parallel programming. Nevertheless, they did not seem to have much problem on synchronization.[6]

### 4.2 Development Environment

Software development tools, including debugging tools and performance tuning tools, tailored for parallel processing, have been found indispensable for high software productivity.

Not all the PIM systems got ready with the final scale described above. Most of the application software systems were first developed on Multi-PSI system with up to 64 processors, then ported to PIM/m with 128 processors when it got ready, then to its 256 processor version when its production completed, and then to PIM/p with 512 processors. Many programs that showed almost linear speed-up with 16 processors would not with 512 processors. Using the tuning tool ParaGraph was very effective in finding the bottlenecks and it usually took only a few weeks before a new version with improved load distribution algorithm showed good parallelism on larger scale systems.

### 4.3 High Performance Hardware

Hardware systems with high performance have been very useful. It was helpful to be able to run application systems in their earlier development phases and tune them gradually, rather than having to develop an optimized version from scratch. It is especially true with considerably complicated knowledge processing systems.

## 5 Future Work

A serious problem with the current parallel inference system is that it runs only on specially devised hardware. Although the system is efficient and self-contained, requiring special hardware is a great obstacle in sharing the environment with researchers world-wide.

Research in subsetting the language to allow more concise and efficient implementation has been conducted with promising preliminary results [UK3]. A separate effort of implementing KL1 by translating into C [CT2] shows reasonable performance with very high portability. These results indicate the future direction of implementing the language and the system on stock hardware to be shared among wider range of researchers in parallel software area.

The Fifth Generaion Computer Systems project ended in March 1993. The Japanese Ministry of International Trade and Industry, considering abovementioned recent research on implementation, launched a new project of two years beginning from April 1993, aiming at disseminating the technologies established in the FGCS project by amalgamating it with conventional computer technologies, such as UNIX and RISC processors. In this project, the following results are expected.

- KL1 implementation with reasonable software development environment on commercially available hardware. An implementation with excellent portability by compilation into C (nicknamed KLIC, for KL1 in C) for UNIX workstations, parallel UNIX systems and network-connected computer systems is planned.
- Knowledge processing software and experimental application software further refined and ported to KLIC. It is also planned to include software already available on UNIX systems as components such software, by utilizing the foreign language interface provided by KLIC.

All the resultant software is planned to be freely available worldwide to be utilized as the basis of further research in the area of knowledge information processing systems.

[6] Although those who had been too much accustomed to Prolog found difficulty in realizing large semantic difference of the two languages with similar syntax.

## References


[AS1] Aikawa, S. *et al.*: ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems. Proc. *FGCS '92* (1992) 286–293

[CK1] Clark, K. *et al.*: PARLOG: Parallel Programming in Logic. ACM Trans. Prog. Lang. Syst. **8-1** (1986).

[CT1] Chikayama, T.: Operating System PIMOS and Kernel Language KL1. Proc. *FGCS '92* (1992) 73–88

[CT2] Chikayama, T.: A Portable and Reasonably Efficient Implementation of KL1. Technical Report **747** ICOT (1992)

[DH1] Date, H. *et al.*: LSI-CAD Programs on Parallel Inference Machine. Proc. *FGCS '92* (1992) 237–247

[DH2] Date, H. and Taki, K.: A Parallel Lookahead line Search Router with Automatic Ripup-and-reroute. Proc. *EDAC-EUROASIC 93* (1993)

[FM1] Fujita, M. *et al.*: Model Generation Theorem Provers on a Parallel Inference Machine. Proc. *FGCS '92* (1992) 357–375

[FM2] Furuichi, M. *et al.*: A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI. Proc. *Second ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming* (1990) 50–59

[HM1] Hirosawa, M. *et al.*: Folding Simulation using Temperature Parallel Simulated Annealing. Proc. *FGCS '92* (1992) 300–306

[HM2] Hirosawa, M. *et al.*: Protein Multiple Sequence Alignment using Knowledge. Proc. *26th Annual Hawaii Int. Conf. on System Sci.* **1** (1993) 803-812

[IM1] Ishikawa, M. *et al.*: Protein Sequence Analysis by Parallel Inference Machine. Proc. *FGCS '92* (1992) 294-299

[IT1] Ikeda, T. *et al.*: Argument Text Generation System (Dulcinea). Proc. *FGCS '92* (1992) 385-394

[KK1] Kimura, K. and Ichiyoshi, N.: Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme. Proc. Sixth Distributed Memory Computing Conf. (1991)

[KM1] Kawamura, M. *et al.*: Parallel Database Management System: Kappa-P. Proc. *FGCS '92* (1992) 248–256

[MY1] Matsumoto, Y. and Taki, K.: Adaptive Time-Ceiling for Efficient Parallel Discrete Event Simulation. *Western Multiconf. on Computer Simulation* (1993) 101–106

[MY2] Minoda, Y. *et al.*: A Cooperative Logic Design Expert System on a Multiprocessor. Proc. *FGCS '92* (1992) 1181–1189

[NK1] Nitta, K. *et al.*: HELIC-II: A Legal Reasoning System on the Parallel Inference Machine. Proc. *FGCS '92* (1992) 1115–1124

[NK2] Nitta, K. *et al.*: Experimental Parallel Inference Software. Proc. *FGCS '92* (1992) 166–190

[OK] Onizuka, K. *et al.*: A Scheme for Protein Tertiary Structure Prediction Based on Stochastic Reasoning. *submitted to Workshop "Artificial Intelligence and Genome" at IJCAI 93* (1993)

[SA1] Saraswat, V. A. *et al.*: Janus: A Step Towards Distributed Constraint Programming. Proc. *North American Conf. on Logic Programming* (1990)

[SE1] Shapiro, E. *et al.*: Object-oriented Programming in Concurrent Prolog. New Generation Computing **1-1** (1983)

[SE2] Shapiro, E.: Systems Programming in Concurrent Prolog. In Logic Programming and its Applications, M. van Canegham and D. H. D. Warren(eds.), Albex Publishing Co. (1986) 50–74

[TA1] Takeuchi, A. *et al.*: An Operational Semantics of ANDOR-II, A Parallel Logic Programming Language with AND- and OR-Parallelism, LNCS-491, Concurrency: Theory, Language, and Architecture. Springer-Verlag (1989) 173–209

[TK1] Taki, K.: Parallel Inference Machine PIM. Proc. *FGCS '92* (1992) 50–72

[TS1] Terasaki, S. *et al.*: Parallel Constraint Logic Programming Language GDCC and its Parallel Constraint Solvers. Proc. *FGCS '92* (1992) 330–346

[UK1] Ueda, K. *et al.*: Design of the Kernel Language for the Parallel Inference Machine. The Computer Journal **33-6** (1990) 494–500

[UK2] Ueda, K.: Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. Technical Report **208** ICOT (1986)

[UK3] Ueda, K. *et al.*: A New Implementation Technique for flat GHC. In Proc. *Seventh Int. Conf. on Logic Programming*, MIT Press (1990) 3–17

[WT1] Watanabe, T. *et al.*: Co-HLEX: Co-operative Recursive LSI Layout Problem Solver on Japan's Fifth Generation Parallel Inference Machine. Proc. *FGCS '92* (1992) 1173–1180

[YH1] Yashiro, H. *et al.*: Resource Management of PIMOS. Proc. *FGCS '92* (1992) 269–277

[YH2] Yasukawa, H. *et al.*: Object, Properties, and Modules in QUIXOTE. Proc. *FGCS '92* (1992) 257–268

[YS1] Yamasaki, S. *et al.*: A Parallel Cooperation Model for Natural Language Processing. Proc. *FGCS '92* (1992) 405–413