PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

RESEARCH AND DEVELOPMENT OF THE PARALLEL INFERENCE SYSTEM IN THE INTERMEDIATE STAGE OF THE FGCS PROJECT

Shunichi Uchida Kazuo Taki Katsuto Nakajima
Atsuhiro Goto Takashi Chikayama

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

ABSTRACT

This paper introduces the research and development of the inference system in the FGCS project. Research on the parallel inference system in the intermediate stage included the parallel hardware system (PIM) and the parallel software system. It started with the adoption of a new language, GHC, as the base of the parallel kernel language KL1. At the same time, it was determined to develop the multi-PSI system in order to encourage parallel software research. A distributed language processor for KL1 was developed on the multi-PSI system.

The development of the KL1 language processor and also the development of a parallel operating system for the PIM (PIMOS) was considered to include many unknown problems. However, the goal of the research and development for the intermediate stage was set to include the experimental building of the PIMOS on the multi-PSI system and to run some small scale application software systems on it.

The development of a parallel hardware system aimed at the experimental building of a PIM having about 100 processing elements (PEs), making the best use of experiences gained in the development of the smaller version of the PSI CPU.

Another aim was to develop a cross programming environment of KL1 on the smaller version of the PSI, PSI-II, so as to prepare for the wider and larger scale parallel software development to be done in the final stage.

This paper contains not only the research and development results but also the background in which important technical decisions were made.

1 INTRODUCTION

The research of the parallel inference system in the intermediate stage included a parallel hardware system and software system. It started with planning aiming at the experimental building of a parallel inference machine (PIM) having about 100 processing elements (PEs) and also its parallel software system, the PIM operating system (PIMOS). This planning made full use of the development in the initial stage such as the parallel architectures and models proposed in the research on the dataflow machine and the reduction machine, and also the PSI hardware system and its operating system, SIMPOS.

A unique feature of the intermediate stage research and development is the coupling of the research on the parallel hardware with one type of parallel software, the parallel operating system, PIMOS, newly adopted as an important research target.

The research and development of parallel software has never been conducted in as a large project anywhere in the world. This meant that very little pragmatic software had been developed. The main reason was the lack of suitable parallel hardware.

On the other hand, parallel hardware which was worth building parallel operating systems for had never been built, although special purpose parallel hardware systems such as image processors had been built. This was because the design of general purpose parallel hardware needed the characteristics of the behavior of the parallel software running on it. This implies that the parallel software must exist before parallel hardware can exist. Thus, the relation of parallel hardware and software is something like the chicken and egg problem.

To solve this problem, a stepwise development strategy was introduced which used two successive versions of the multi-PSI systems, the multi-PSI-V1 which contained six PSI-I machines as its PEs and the multi-PSI-V2 which contained up to 64 PSI-II CPUs.

These multi-PSI systems enabled software researchers to confirm their ideas on parallel programs by writing and running them in a real parallel hardware environment. Then, their ideas reflected in the design of the next version of the hardware.

Using the multi-PSI-V1, the experimental distributed language processor of the parallel logic programming

language, Guarded Horn Clause (GHC), was built to study the communication mechanism between PEs. This work thoroughly analyzed how to extend the GHC language to design the parallel kernel language, KL1, and the kind of functions required for the PIMOS. The experimental distributed language processor of GHC had its debugging functions augmented and extended to a pseudo parallel programming environment of GHC on PSI-I and PSI-II. It was used for parallel algorithm research running many small scale benchmark programs.

This made it possible to start the development of the multi-PSI-V2, followed by the development of the practical KL1 distributed language processor and PIMOS. These research results were reflected in the design of the PIM hardware.

The stepwise development strategy in which the software and hardware development grew little by little worked more effectively than had been expected.

Thus, most of the technical aims included in the intermediate stage goal were achieved by the development of the multi-PSI-V2 with the practical KL1 distributed language processor and the kernel part of the PIMOS running on it.

The contribution of the research of the multi-PSI-V2 and PIMOS to the design of the PIM hardware enabled us to set the research goal of the experimental implementation of the PIM hardware a higher level in terms of its processing speed and the size of its PEs. Thus, the research and development of the PIM in the latter half of the intermediate stage considered not only the intermediate stage goals but also the goal of the final stage which aims at a parallel hardware system having about 1000 PEs.

The flow of the important research activities is shown in Figure 1

This paper describes how we set up the goals and how we made many important technical decisions in the research and development of the PIM and PIMOS in the intermediate stage. Technical details of KL1 language processors, multi-PSI systems, PIMOS and PIM will be described in other papers also presented at the FGCS'88 conference [5] [3].

2 INTERMEDIATE STAGE PLAN

2.1 Evaluation of The Inference Machine Research in The Initial Stage

Research on the PIM and KL1:
The research and development of the inference machine in the initial stage included research on PIM architectures and the development of the sequential inference machines.

Research on the PIM architectures was conducted to find hardware mechanisms that could efficiently execute logic programming languages in parallel using such computational models as dataflow, reduction and Kabuwake.

The languages used for this research were Pure Prolog and Concurrent Prolog, (CP) [14]. Several software and hardware simulators for these languages were built to analyze the algorithms and implementation methods of their interpreters [6].

This research showed us such problems as the insufficiency of OR-parallel Prolog in describing communicating processes and the difficulty of implementing the hardware for the dataflow model. This research made us realized that the scarce accumulation of parallel software would cause a problem in benchmarking the architectures.

Research on the parallel logic programming languages investigated and evaluated such parallel logic programming languages as CP and PARLOG [4] and also, an abstract machine language for Prolog, WAM [21]. Consideration of implementing these languages in hardware motivated us to design a simpler language and GHC was born at the end of the initial stage [20].

Development of sequential inference machines:
In the development of PSI-I [18]and CHI-I [11], the architectures to execute sequential logic programming languages efficiently and their hardware implementation techniques were developed with the KL0 firmware interpreter and the operating system, SIMPOS. This made us decide on the development of the multi-PSI system and start the design of a smaller version of the PSI, PSI-II [19].

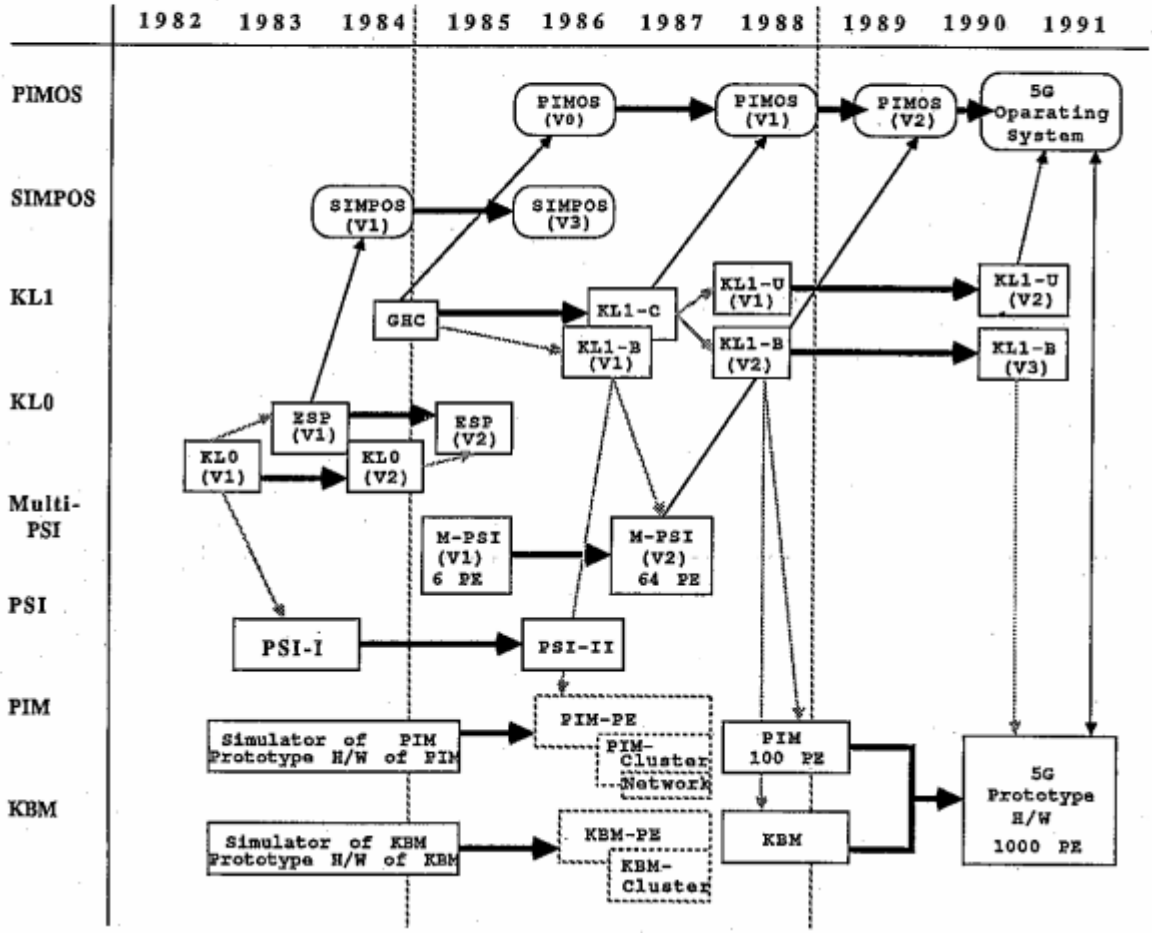
The development of PSI-II and also CHI-II [7] established such important techniques as the architecture design based on the WAM instruction set and the code optimization technique in compilers. The development of a new logic programming language, ESP [1] which combined logic programming and object oriented programming features to describe SIMPOS impressed us by its advantages in terms of software productivity, program readability and maintainability. This led us to describe the PIMOS with a single high-level language, KL1, and to make the PIMOS a single language system.

Summary of the research in initial stage:
Considering the goal of the intermediate stage, PIM having about 100 PEs, the research achievements of the initial stage are summarized as follows:

1. Logic programming is appropriate for parallel processing and also for description of operating systems.

2. Current VLSI technology, including CAD tools is not sufficient to confine all the functions required by logic programming in one chip. Thus, the functions implemented in the PE must be restricted. The code optimization technique in compilers is impor-

R & D plan of H/W and S/W



tant to reduce the complexity of the PE hardware and attain the high performance and reliability of the PIM.

3. The lack of practical parallel programs makes the architecture design difficult. It is an urgent matter to encourage parallel software research providing software researchers with practical parallel hardware environments.

2.2 The Intermediate Stage Plan and Its Implementation

Before the details of the intermediate stage goal were fixed, the following policies were confirmed:

1. The research of the parallel software systems, specifically, the KL1 language processor and PIMOS, must be closely linked to research on parallel hardware systems, that is, the multi-PSI systems and PIM. They must proceed concurrently and stimulate each other.

2. The parallel hardware systems must be designed to support software research and development. This means that hardware research must not be isolated from software research.

3. Information on the behavior of parallel software systems is essential for practical hardware design although little work had been done it. Then, the tools and the environments to encourage parallel software research must have the highest priority in the investment of human and financial resources.

In line with the above policies, the following goals were defined at the beginning of the intermediate stage, in April 1985. However, the level of these goals had to be set higher in the middle of the intermediate stage.

1. Experimental building of a PIM having about 100 PEs which efficiently supports KL1.

2. Target processing speed of 2M to 5M LIPS.

3. Experimental building of the PIMOS which is described in KL1.

As the PIMOS was considered to be the most difficult of these items, the development of the multi-PSI system which played a role of the PIMOS research tool was begun quickly to stimulate the development of the KL1 language processor. It was decided to develop the multi-PSI system in two consecutive steps: the development of the multi-PSI-V1 using PSI-I and the development of the multi-PSI-V2, making smaller version of the PSI CPUs be its PE.

In the spring of 1986, the detailed design of the PE of the multi-PSI-V2, that is the CPU of PSI-II, was completed and the execution speed of GHC by the multi-PSI-V2 was estimated by small sample programs. The estimated performance of the firmware interpreter of GHC on the PE was very impressive. This also indicated that the performance goal defined at the beginning of the intermediate stage would be attained by the multi-PSI-V2 having 64 PEs although the overhead caused by the PIMOS was uncertain.

Then, the level of the intermediate stage goals were set higher and defined in more detail, taking the final stage goals aiming at a PIM of 1000 PEs into account.

1. The performance goal is 10M to 20M LIPS for a PIM of 100 PEs and 200K to 500K LIPS for one PE.

2. To reduce the communication overhead between PEs, a cluster should be introduced to connect about eight PEs with a shared memory. Then, a PIM with 100 PEs can use several clusters connected by a hierarchical network.

3. The PIMOS should be developed on the multi-PSI-V2. A KL1 distributed language processor should be implemented in firmware. After the PIM hardware is completed, the PIMOS is moved from the multi-PSI-V2 to the PIM.

4. A KL1 cross programming environment should be developed on PSI-II using a KL1 pseudo parallel language processor.

With these goals, the development of the PIM and PIMOS could proceed independently. This enabled us to avoid the problem where one step of development has to wait for the completion of another.

From late 1987, the study of larger scale network mechanisms was started to determine the important technical problems in connecting around 1000 PEs or around 100 clusters. The technical problems were divided among the PIM research groups of the cooperating manufacturers so that they could be further studied through software simulation and by building experimental hardware.

This division of the jobs related to the design and implementation of the PIM among cooperating manufacturers was considered to be essential in the final stage to build a larger scale experimental system. Thus, the preparation for the job division was begun from the latter half of the intermediate stage.

The research and development items described above are summarized in Figure 2.

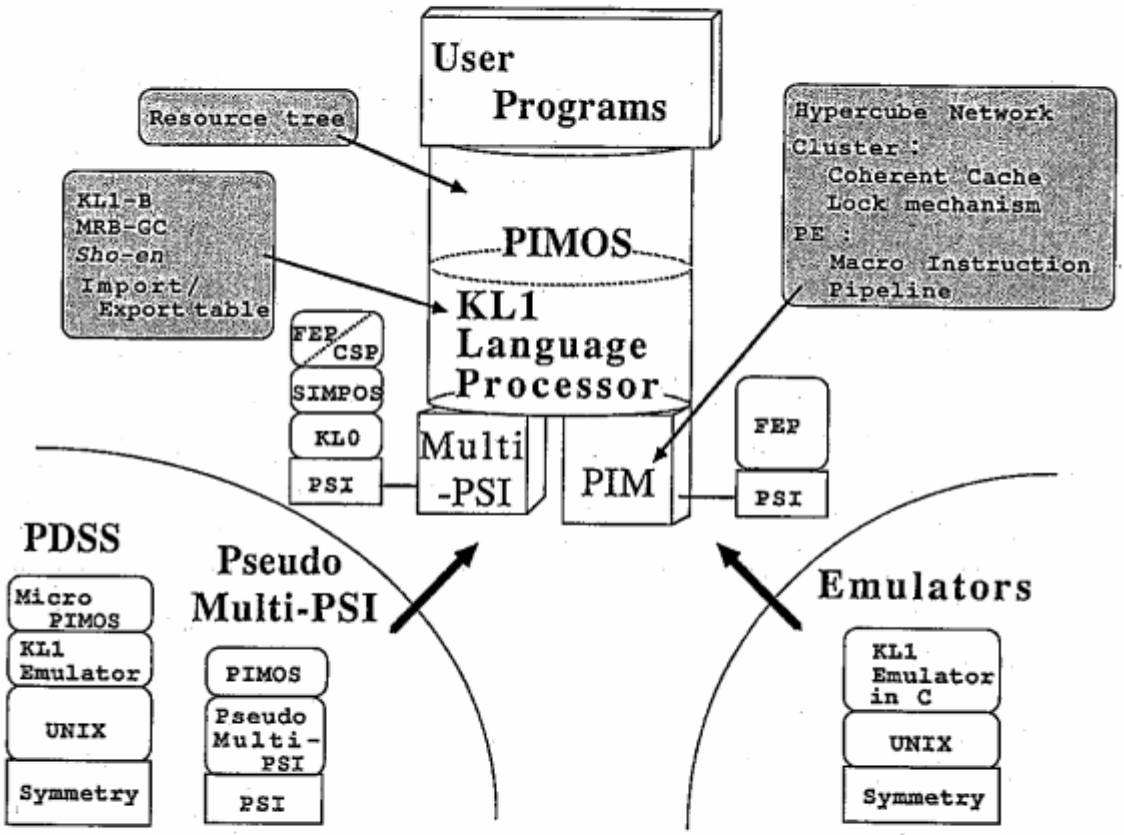


Figure 2: Main Research Items

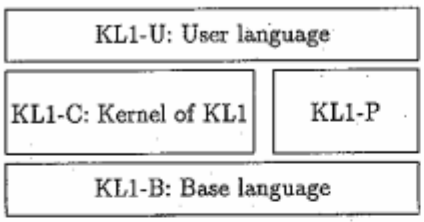


Figure 3: The KL1 Language System

3 KL1 LANGUAGE PROCESSORS

3.1 KL1 Language System

In the design of the language for the PIM, there were two choices: an OR-parallel logic programming language or a stream-based AND-parallel logic programming language. Considerations of the description of the PIMOS which required the description of message passing among many processes and also the simplicity of their hardware support resulted in the adoption of the latter, GHC was decided on.

We believe that this choice was appropriate because we had to make a great effort to develop its distributed language processor although we designed KL1 not fully based on GHC but on a subset of GHC, called Flat GHC (FGHC).

GHC is not a practical language but defines the computational model. A system description language and a machine language has to be designed based on this model. The language system of KL1 was defined at the beginning of the intermediate stage. It had the language layers shown in Figure 3.

KL1-C is a system description language which is the kernel of this language system. It has such functions as modularization and macro-expansion, and many practical built-in predicates in addition to the FGHC functions. KL1-C is used to describe the PIMOS and application systems. KL1-C is compiled to KL1-B.

KL1-B is an abstract machine instruction set used in the same way as WAM is for Prolog. It is used as the common machine language for both the multi-PSI-V2 and PIM.

KL1-P is a notation used with KL1-C to specify how to divide jobs into sub-jobs that can be processed in parallel or how to distribute jobs.

KL1-U is a user defined language which will be designed to fulfill the requirements in a variety of application systems. Some examples are an object-oriented parallel language $A'UM$ [22] and constraint logic programming languages.

3.2 Distributed Language Processors of KL1

The design of PIM primarily needs information on the specifications of KL1-C and KL1-B and the control mechanism and behavior of the PIMOS. This means that the characteristics of the total system with layers spanning from application software to hardware must be estimated.

As the estimation of the characteristics of application software was almost impossible, the study was begun from the possible configurations of the PIM hardware system, considering a PIM of 1000 PEs. To connect many PEs, the connection mechanism had to use a loosely coupled mechanism such as a packet switching network.

On the other hand, it was expected that the size of parallel processes, that is, the granularity, would be small in most application software written in KL1. For small granularity, a tightly coupled mechanism such as a common bus with a parallel cache system and a shared memory was adequate to reduce the communication delay.

Both of these are important mechanisms for large scale parallel systems. The PIM of the final stage was expected to use a hierarchical network combining both mechanisms. However, a loosely coupled network could have different structures such as two-dimensional mesh, hypercube, or cross bar, depending on the characteristics of application software in terms of structures of programs and algorithms.

Then, the KL1 language processor had to be designed independently from the details of the hardware structures. Before the design started, we decided to build three experimental KL1 language processors.

1. An intra-PE language processor.
2. A tightly distributed language processor for the PIM cluster.
3. A loosely distributed language processor for the multi-PSI-V2.

The development of the language processors (LPs) began with the design of an intra-PE language processor (LP) of which the key issues were the design of the KL1-B instruction set, optimization techniques in the compiler, the implementation method of the process management, and the garbage collection (GC) technique.

The tightly distributed LP uses a shared memory, and thus, a single address space. The key issues were the communication method between PEs, including the lock mechanism used for PEs to share the common data structure and the cache protocol, and the implementation of GC.

The loosely distributed LP needs communications among multiple address spaces. It contained many difficult research problems not confined to the LP but related to the PIMOS functions. For example, a communication between two different address spaces usually takes much

more time than the tightly distributed LP. It requires optimization in its implementation such as the caching of the transferred data. The resource management of both local and global address spaces needs complex mechanisms including global GC. The observation of the behavior of the LP needs also special mechanisms as well as the mechanism for the resource management.

The experimental intra-PE LPs for FGHC had been written in Prolog and the language C in the initial stage. Based on these, the loosely distributed LP was implemented for FGHC on the multi-PSI-V1 to determine the problems in the design of KL1-C and KL1-B.

In the latter half of the intermediate stage, the loosely distributed LP for KL1-B was designed and implemented in firmware on the multi-PSI-V2. KL1-C was designed concurrently followed by the design of the PIMOS. The design of the tightly distributed LP for PIM was started in the middle of 1987 [13].

4 MULTI-PSI SYSTEM

4.1 Outline of A Research and Development

The purpose of the multi-PSI system was to provide software researchers with the parallel hardware environment very quickly; however, its development contained many research problems. The reasons for this development are summarized as follows.

1. The development of KL1 language processors and PIMOS needs many new ideas. Effectiveness of the ideas must be quickly evaluated by making experiments. The research and development of PSI has developed many skillful researchers and engineers and has improved software and firmware tools. It is the most suitable environment in which to make many experiments in a short period.

2. It is not easy to make an experimental parallel hardware system reliable and maintainable enough for use as a software development tool. Using the PSI as its PE greatly reduces this problem.

Before the intermediate stage, discussions were held on the specifications of the multi-PSI system, especially on its network mechanism and the architecture of the smaller version of the PSI.

Their design and implementation started just after the intermediate stage began. The multi-PSI-V1 was completed in the middle of 1986. The CPU of the smaller version of the PSI was completed in early 1987 and built up as the front end machine of the multi-PSI-V2. This front end machine was also used as a stand-alone workstation, PSI-II.

PSI-II employed the instruction set based on WAM and made full use of the code optimization technique with its compiler. For compactness, its hardware used nine newly developed 8k-gate CMOS gate-array LSIs. PSI-II attained a threefold to fivefold improvement in ESP execution speed.

Table 1: Main Features of PSI-I and PSI-II

	PSI-I	PSI-II
Device	TTL (Fast)	CMOS-G.A., TTL
Cycle time	200 ns	200 ns
Word width	40 bits	40 bits
WCS	64b x 16KW	53b x 16KW
Cache memory	4KW x 2	4KW x 1
Main memory	16MW (Max)	64MW (Max)
Memory chip	256 Kbit	1 Mbit
Max. No. of Process	64	S/W defined
Machine code	Table type	WAM type
Structure data	Sharing	Copying
Exe. speed(Average)	30 KLIPS	150 KLIPS
Exe. speed(Append)	35 KLIPS	333 KLIPS

The network of the multi-PSI system has a two-dimensional mesh structure. Each node of the network has a function to relay the packets from one node to another. The routing control mechanism in the node was extended and implemented in two 20k-gate LSIs for the multi-PSI-V2. Development was completed in the spring of 1988, but the inspection of the hardware consisting of 64 PEs took a long time because of the connection of the front end machine, and preparation of test programs and observation programs. The preparation of the inspection was much more complicated than had been anticipated. The total hardware system began operation in the summer of 1988.

The design of the loosely distributed language processor of KL1-B was started in 1986. Its specification was so complex that its verification had to be made by writing it in the language C. The firmware implementation of the language processor was begun in late 1987 and partially completed in the summer of 1988. It was hurriedly provided to develop the PIMOS on PSI-II.

On PSI-II, the KL1-B firmware resides with the KL0 firmware which runs the SIMPOS. Programs written in KL1 are run as processes under the SIMPOS as well as the programs written in KL0 (or ESP). The SIMPOS

switches the firmware depending on whether the process is written in KL0 or KL1-B.

In 1988, the hardware of the multi-PSI-V2 is being reproduced to distribute it to many software research groups so that they can start parallel software research from the beginning of the final stage.

The development of the multi-PSI system made full use of experience and item that had already been developed, such as microprogramming tools, evaluation tools and the SIMPOS.

Without these, we could not cope with the scale and complexity of each part of the system, for example, the large and complex hardware, complex firmware for KL1-B, and observation and diagnosis functions of the front end machine.

4.2 Multi-PSI-V1

4.2.1 Functions and Organization of The System

The multi-PSI-V1 consists of six PSI-Is as its PEs which are connected by a two-dimensional mesh network. The reason why a two-dimensional mesh network was adopted was that it was considered to be appropriate as the first step in studying the load balancing of application software systems.

To solve the load balancing or load distribution problem, we first adopted the policy that programmers must explicitly specify how to divide their jobs into sub-jobs that can be processed in parallel at the system programming level.

This was different from most past proposals on this problem made after past parallel architecture research which tried to provide automatic mechanisms to exploit parallelism in users' programs. We concluded that this method was ideal but too difficult to attain, especially for the PIM and PIMOS.

Our proposal is that one job written in KL1-C be divided into many sub-jobs specified by the programmers using KL1-P. Furthermore, the programmer should be able to specify explicitly the amount of computational resources and the priority given to each sub-job. This specification, however, may not be accurate, something like a first order approximation. The actual loads of sub-jobs on PEs will accordingly be unbalanced among PEs. Thus, the PIMOS tries to compensate dynamically for this imbalance as much as possible.

We examined a model in which computational resources were uniformly distributed on a two-dimensional plane. When a programmer tried to specify the how jobs are divided and the amount of resources of each sub-job, the position and the amount of space were used as the parameters. An overhead caused by a communication between PEs was proportional to the distance between two points on the plane. Thus, the locality of communications is represented. This model was named the Processor Power Plane model [2].

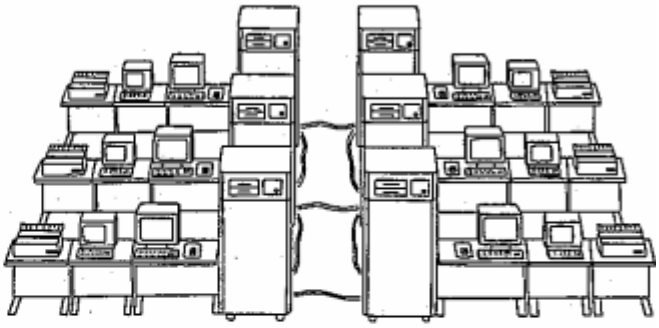


Figure 4: Multi-PSI-V1

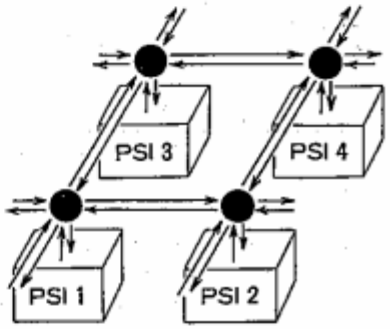


Figure 5: Network of Multi-PSI System

A dynamic job reallocation method based on this model was created. As this method was considered to cause too much overhead if implemented only by the PIMOS, its hardware support mechanism was designed and implemented in the network node of the multi-PSI-V2 [16].

The network of the multi-PSI system was designed based on the above. The appearance of the multi-PSI-V1 is shown in Figure 4. The structure of the network is shown in Figure 5.

Each node of the network has five channels. One of them is connected to the PE of its node and the other four are connected to four neighbors. Each channel has independent input and output circuits, each of which contains 10bit data and signal lines and 4k-byte FIFO buffers, so that multiple communication paths can be

opened. Each node also has a simple routing mechanism controlled by a table in which control instructions are kept. The data transfer rate of each channel is 500 Kbytes/sec.

Packet transfer is controlled by the PSI microprogram and the SIMPOS device handler. The FGHC distributed language processor implements inter-PE communications using this handler.

4.2.2 Research on The KL1 Language Processor

The purpose of the multi-PSI-V1 was to develop a loosely distributed LP for FGHC to study the implementation of distributed unification between PEs, the control method of the load distribution, the observation mechanism to watch the behavior of the PEs, and the debugging support mechanism.

The development of this LP started with the development of a pseudo parallel LP for FGHC on a single PSI-I. This pseudo parallel LP simulated the parallel execution of FGHC programs under the control of the SIMPOS. It had a simple tracer and debugger and also some measurement functions for computation time, number of reductions, and communication delay.

After the multi-PSI-V1 hardware was completed, this pseudo parallel LP was moved on to it and extended to execute the programs in parallel among six PEs. As this LP was built to evaluate the implementation methods of the mechanisms described above, the execution speed was about 1K LIPS. However, it ran several small scale programs such as the eight queen problem and the best path problem and contributed to the study of job distribution and the parallel algorithms [17].

Among the achievements of this experiment, the most valuable was the method to implement the observation and maintenance mechanisms to control and measure the distributed execution of programs. This experience was reflected in the design of the multi-PSI-V2.

4.3 Multi-PSI-V2

4.3.1 Functions and Organization of The System

The multi-PSI-V2 contains up to 64 PEs. Its appearance is shown in Figure 6. The organization of the system is as follows.

1. Each PE consists of three CPU boards, one network board and four memory boards containing 80M bytes.
2. Its network has the same structure as the multi-PSI-V1. The functions of each node are augmented using two 20k-gate LSIs to attain the data transfer rate of 5 Mbytes/sec and to contain the circuit to support the load balancing mechanism described in 4.2.1.
3. The main part of the system consists of eight cabinets, each of which contains eight PEs. The minimum configuration of the system is one cabinet.
4. The front end machine, PSI-II, has such functions as input and output, observation and maintenance for the main part. Up to four front end machines can be connected to the 64 PE system.
5. The front end machine has two logically independent functions which are called the front end processor (FEP) and the console processor (CSP). The FEP performs input and output operations for the PIMOS, the CSP performs the functions for observation and maintenance.
6. The front end machine is connected to each PE of the main part with a 10-bit-wide common bus. Using this bus, the front end machine loads firmware and software and monitors and diagnoses all the PEs.

The scale of the hardware of the main part is very large. The hardware contains 512 printed circuit boards and a 5G byte memory. Its testing and debugging required many firmware and software tools. Many of them were prepared as CSP functions. Most of the test programs were written in ESP. In its hardware debugging, the KL0 firmware and the kernel part of the SIMPOS were loaded in each PE to run the test programs written in ESP. This made the debugging very efficient and helped us to keep to the development schedule.

4.3.2 Loosely Distributed KL1 Language Processor

The development of the loosely distributed KL1 LP began with the design of the KL1-B specification. The design of the KL1-C specification was also started by extending the specification of FGHC.

At the beginning of the design, the level of KL1-B was roughly set to the level of WAM and then extended to cover distributed implementation. The architecture of the PE based on this level can be seen in Figure 7. [9]

The execution of KL1-B is performed using several queues such as a ready-queue, suspension records and a goal queue. The PE takes one goal (process) from the ready-queue and prepares the environment in registers to execute the goal. The execution produces new ready goals. They are put in the ready queue again. If some variable is not instantiated, the new goal is added to the suspension record. This is a very rough sketch of basic KL1-B execution.

In addition to the basic execution functions based on FGHC, many important functions to make KL1-B practical were added, as described below.

Incremental garbage collection (GC):
One of the major problems in executing KL1-B in a PE

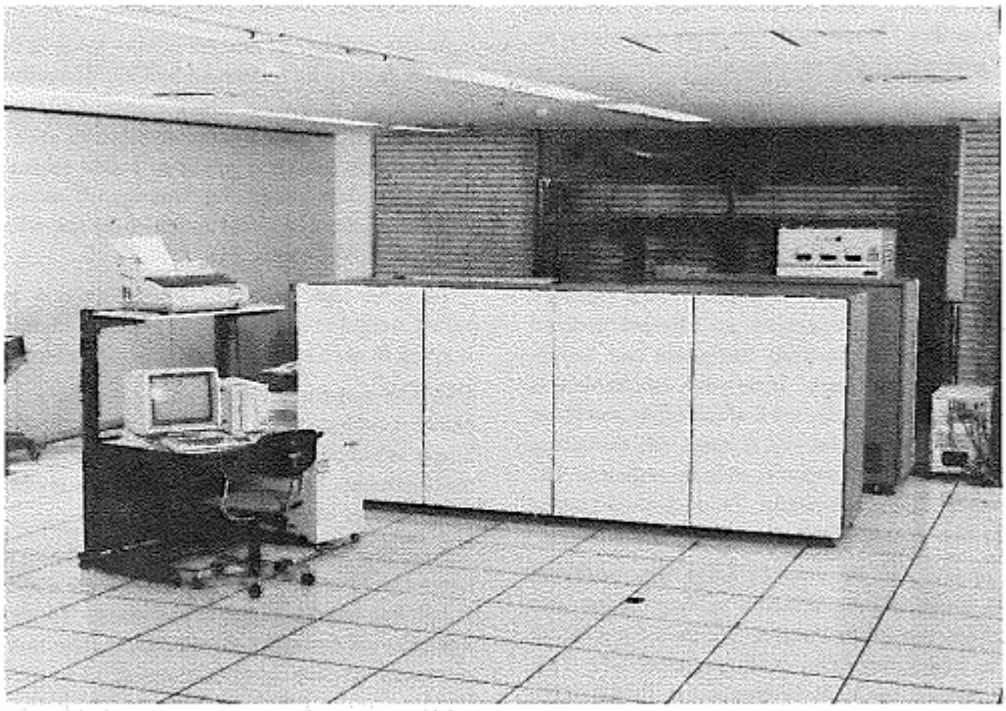


Figure 6: Multi-PSI-V2

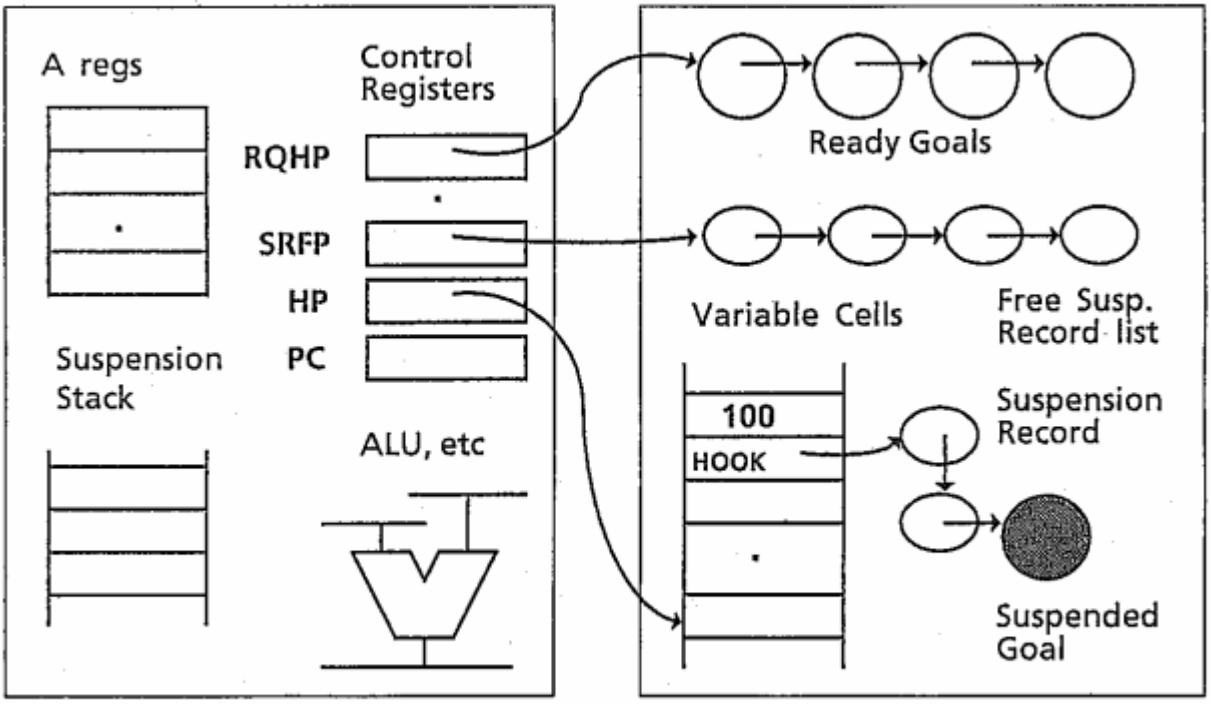


Figure 7: Abstract Architecture of PE

is GC. In executing KL1-B, data areas are made for variables, lists and so on. These areas are made in the heap area in the memory. Some parts of these areas are referred to by several parallel processes.

Languages like KL1 need a method to reclaim the data areas after they become unnecessary. In Prolog, two different methods are used to reclaim unnecessary used areas. One is a stack mechanism which reclaims them on the way of program execution. Another is GC, which usually stops the execution during the GC process.

KL1-B also needs a method to reclaim used memory areas. However, it cannot employ an efficient method like a stack mechanism because the areas can be referred to by several processes. If no such mechanism is employed, reclamation of the used areas relies solely on GC. If usual non-incremental GC is used, the execution will stop very often. Furthermore, non-incremental GC accesses memory almost randomly. Thus, it causes cache mis-hit often and degrades the execution performance.

Incremental GC can be used to solve this problem. However, its naive implementation is very heavy and greatly degrades the performance. We found a method to reduce the heavy overhead of incremental GC. This method was named multiple reference bit (MRB)-GC. MRB-GC uses one additional bit in the tag field of each memory word to indicate whether there are one or more than one references to the word. This made it possible to use the tag handling hardware so as to attain reasonable execution speed.

In the execution of KL1 programs, one memory word is referred to by only one process in many cases. Thus, MRB-GC works very effectively and reduces cache mis-hit ratio and the occurrence of non-incremental GC.

Inter-process communication for distributed unification:
Control of communication among parallel processes is hidden under the KL1-B level. This is useful to make the KL1 language processors common to the multi-PSI-V2 and the PIM.

Unifications among the PEs on the multi-PSI-V2 and the clusters of the PIM are implemented using global packet communication. Implementation of this global communication needs some memory areas in the PE or the cluster. These memory areas should also be reclaimed. A naive solution for this is global GC which must stop all the PEs or the clusters at one time.

To perform GC separately in each PE or cluster, different address spaces have to be provided for each PE or cluster to make a distributed environment. A packet transferred among the different address spaces needs its identifier. A memory address cannot be used for this purpose because the data address is changed by GC. Then, a table is used to convert local addresses to global addresses or vice versa. This table is called an import/export table. This method is often used for high-level language parallel processors.

As the number of entries of this table is limited, used entries have to be reclaimed by GC. The naive solution of this is again global GC. Incremental GC can be applied; however, its ordinary implementation in a distributed environment causes a problem called racing. Racing is caused by delay in packet transfer. If racing occurs, an entry is reclaimed eventhough a packet which refers to that entry still exists.

To solve this problem, we designed a new method called weighted export count (WEC), in which a weight is given to each entry of the table and referenced data [8].

PIMOS support functions:
Many functions were added for supporting the PIMOS. Some examples are functions for program execution management, computational resource management, debugging and maintenance.

Including the functions described above, a KL1-B interpreter which is the kernel of a loosely distributed KL1-B LP was implemented in firmware. To avoid low productivity of firmware development, the interpreter was primarily written in the language C to verify its specification and algorithm of implementation. This interpreter was next used as a specification of the firmware interpreter.

The size of the firmware interpreter is shown below.

Basic instructions	3.8 K
Basic built-in predicates	1.6 K
Memory management, network control, etc.	4.4 K
Hardware control	4.2 K
Total microprogram steps	14 K

Observing this table, the size of the memory management and network control part is larger than that of the basic instruction part which controls the execution of KL1-B in one PE. The basic instructions have been studied in detail in the past. However, the memory management and network control has not. Its instruction design, especially its assignment of required functions to each of the instructions is not optimum. This makes the structure of its firmware implementation complex. This part is most closely related to the control of distributed unification, thus , requiring further study in the future.

A PSI-II which is used as the front end machine of the multi-PSI-V2 or a KL1 cross programming environment must have both KL0 and KL1-B firmware systems. As the total capacity of both firmware systems exceeds the capacity of PSI-II microprogram memory, a firmware overlay function was added. The execution speed of the KL1-B interpreter was 50K to 100K LIPS.

In the development of this interpreter, firmware debugging tools which had been developed for the KL0 firmware were used very effectively. The kernel part of the PIMOS which was being built concurrently was used as a test program for this interpreter.

Expérience in the design and implementation of this interpreter is now used as the solid base for the development of the tightly distributed KL1 LP used in a PIM cluster and also the loosely distributed KL1 LP used for communication among the clusters.

5 PARALLEL INFERENCE MACHINE: PIM

5.1 Outline of Research and Development

5.1.1 Design of The Basic PIM Structure

The PIM intermediate stage research began substantially from 1986 after its intermediate stage goals were defined based on the evaluation of the research results attained in the initial stage.

Discussions on the basic design policy of the PIM covered many aspects such as target performance, basic architecture, available device technology, CAD design tools, development period and development cost.

One important discussion was held on circuit density of the device and development period. It proposed two alternative policies on PIM hardware building.

1. To reduce risks, the design of PIM hardware should limit its scale up to 100 PEs. Low density devices which have well prepared CAD tools should be used so as to make the development period short and certain.

2. Although the risk is high, high density devices should be used to confine one PE to one printed board. The PE should have continuity to a PIM in the final stage, which should have about 1000 PEs. The PE should be superiority in performance to the technical standard in the final stage.

Being influenced by the estimated performance of the multi-PSI-V2, the natural conclusion of the discussion was to choose 2. The research goals based on this policy were considered to be very difficult to attain. However, this choice was thought to be more suited to the philosophy of this project.

The basic architecture of the intermediate stage PIM was determined in order to employ a tag architecture which could maintain continuity from the multi-PSI-V2. The performance goal was defined to be 200K to 500K LIPS for one PE, including the overhead caused by incremental GC. This performance goal implies that if this PE is used for Prolog, it will attain more than 1M LIPS.

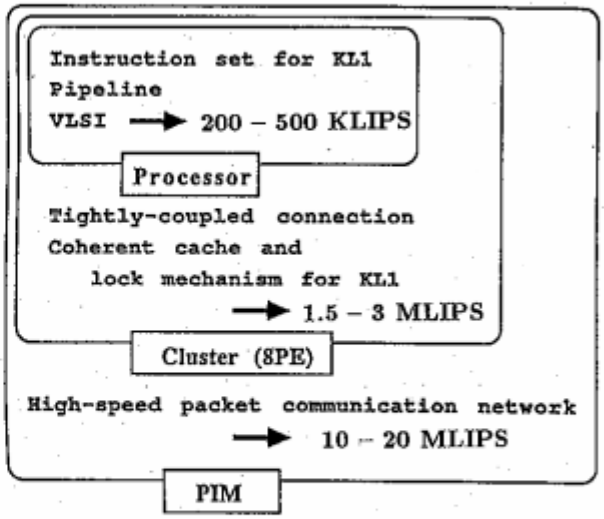


Figure 8: Target execution speed of PIM

To implement one PE on a single board and attain 200K to 500K LIPS, the PE needed not only high density chips and a highly optimized architecture to make its cycle time short but also a sophisticated code optimization technique with its compiler.

The connection mechanism of the PE needed to realize a short response delay for communication between PEs. This delay was considered to determine substantially the lower limit of process's granularity. If the delay is made shorter, the size of processes can be smaller. Then, the number of processes that can be processed in parallel will increase.

The connection mechanism which made this delay shortest was considered to be a common bus with a parallel cache and a shared memory. Then, a cluster was introduced in the PIM. The performance goals of the PE, the cluster and the total system are shown in Figure 8.

5.1.2 Simulators for PIM Design

The design of the PE began with an analysis of the KL1 program's behavior. The primary intention was design of a cache mechanism including its protocol design. This was made through software simulations.

The first simulation was made on VAX 11/785 using a KL1-B interpreter and a parallel cache simulator, both of which were written in the language C. The KL1-B interpreter generated address patterns, by executing small benchmark programs such as the N-queen program. The cache simulator analyzed the address patterns and evaluated the cache mis-hit ratio and traffic on the common bus.

In late 1986, Sequent's Symmetry system was introduced to extend the scale of simulation and to make the simulations more precise. The Symmetry system is a multi processor using conventional microprocessors. On this system, the KL1-B simulator ran at 2K to 5 K LIPS.

Through the simulations, the characteristics of KL1 programs gradually became clear. For example, execution of KL1 programs consumes a heap area very quickly. The locality of its memory accesses was worse than that of usual Prolog. Many memory cells are written and read only once. These facts motivated us to create the MRB-GC method.

For communication among PEs, the communication rate between PEs for KL1 programs is higher than that for conventional language programs. PEs lock memory words before they write them. These write operations, in most cases, do not cause access contention for KL1 programs.

With these results, the number of PEs in the cluster was determined to be eight. The KL1-B simulator was extended to have MRB-GC and then modified to run in parallel using several processors of the Symmetry system. It is now used as a base for designing tightly distributed language processors [13], [10].

5.1.3 Design of The PIM hardware System

PE Design:
The design of PE hardware began with the design of an instruction set. This instruction set may be called a concrete machine instruction set, by contrast with an abstract instruction set like KL1-B. It can approximately be one of two different types as follows:

1. A high-level instruction set which is similar to KL1-B is employed and is implemented in firmware. This instruction set has characteristics like CISC.
2. A low-level instruction set which is directly implemented in hardware. This instruction set has characteristics like RISC. In this case, KL1-B instructions are interpreted by run-time routines written in these low-level instructions.

Each of these choices had both advantages and disadvantages. As described in 4.3.2, the basic instructions of KL1-B had been studied in detail and well optimized. However, instructions for memory management and network control needed further study and were possibly to be changed for better optimization in their implementation.

The advantage of the high-level instruction set with firmware implementation is its flexibility of changing the instruction design and thus, suitable for experimental machines. Its disadvantage is obvious in chip design. It tends to need a long cycle time and large chip area to implement its microprogram memory.

The advantage of low-level instruction set, however, is its simplicity of instructions which results in a short cycle time and small amount of hardware. This feature enables us to improve the execution time if code optimization by compilers works effectively. If the code optimization does not work well, result is drastic, long chains of instructions and slow execution time.

Roughly speaking, the high-level instruction set is suitable for a flexible control oriented design, and the low-level instruction set is for a fast execution oriented design. Both of these are worth further study and evaluation totally in the framework of the PIMOS implementation.

Inter-cluster network design:
The network hardware design has many problems. Most of them are derived from the fact that its performance requirements are very vague. This is because no large scale parallel software has ever run anywhere in the world. Thus, no one can imagine how it will behave and what performance requirements are.

We are trying to design the network hardware bottom up and are holding discussions on the following issues.

1. Control methods and their hardware support of the inter-cluster network
2. Structures (or topology) of the inter-cluster network

On the first item, the target of the design is to attain fast response time or shortest delay for message transfer between two clusters. One idea is hardware support for management of the import/export table and caching of transferred data.

On the second item, several important network structures have been proposed. Some examples are the two-dimensional mesh of the multi-PSI-V2, hypercube and cross-bar.

However, no detailed discussion has been held on the structures in connection with a module structure of parallel programs, distribution of parallel jobs and locality of communication among parallel processes, because the characteristics of large scale parallel programs are not known yet. At this level, software simulation is not as reliable as the simulation of a KL1 program's behavior inside the cluster. No appropriate benchmark program is available. This is something like outer space in the research of parallel processing.

We expect that the PIMOS and some parallel application software running on the multi-PSI-V2 will give us some new knowledge on the above issues.

Current plan of the PIM implementation:
After the design of the PIM cluster was completed in the spring of 1988, the design of the PIM total system which included 16 clusters began in parallel the chip design and production.

The design of the PIM total system made us realize that it would contain many difficult problems and several technical alternatives to be further studied, as described above.

We have decided to deal with these problems including many alternative choices by dividing them among research groups at ICOT and cooperating manufacturers.

First of all, the PIM for the intermediate stage goals adopted the low-level instruction set for its PE and a hypercube network for its inter-cluster network. It was designed to be extensible up to about 500 PEs. An attempt was made to construct a hardware system having 128 PEs around the end of the intermediate stage. As this development will be continued in the final stage, we named it PIM/p to distinguish it from other PIM models to be developed also in the final stage.

We now plan to develop several experimental hardware systems based on different models such as the PIM/c which will adopt the high-level instruction set for its PE and a cross-bar network, and the PIM/m which will be an improved and extended version of the multi-PSI-V2. All of these will use the PIMOS as their common operating system and many software experiments will be made on them.

5.2 Functions and Organization of The PIM/p

5.2.1 Configuration of The Hardware System

The PIM/p, whose hardware system will be constructed around the end of the intermediate stage, consists of a main part which contains 16 clusters, a front end machine, PSI-II, and an SVP which performs maintenance of the main part as shown in Figure 9.

Eight PEs are connected in a cluster. Eight clusters are contained in one cabinet. Then, a PIM/p having 128 PEs consists of two cabinets.

5.2.2 PE and Cluster Architecture

The design of the PE started with the employment of a low-level instruction set and a four stage pipeline hardware for instruction execution. However, it was realized that the complexity of some KL1-B instructions needed too many low-level instructions to be described. Then, some macro instructions were added to the PE's instruction set. A macro instruction is interpreted by dedicated low-level instructions stored in a special internal memory which contains about 8K instructions. The instructions stored in this memory are called internal instructions. Ordinary instructions are read from an instruction cache and called external instructions. Internal instructions can be regarded as a kind of microprogram [15].

The width of the external instruction is either four, six or eight bytes. The function of each of these instructions is much more sophisticated than the usual RISC instruction of a conventional microprocessor. KL1-B instructions are too complicated to be interpreted only by the external instructions. Thus, macro instructions have to be added.

In execution of KL1-B instructions, a conditional branch operation depending on data types appears very often. To perform this operation quickly, delayed branch instructions are provided to reduce useless execution cycles by augmenting the instruction execution pipeline.

A CPU of the PE has thirty-two 40-bit general registers and other dedicated registers for tag checking, floating point numbers and so on. The contents of these registers must be saved for process switching. In execution of KL1-B, process switching happens very frequently. To perform process switching quickly, a special instruction called slit-check is introduced using the characteristics of KL1-B instructions. The slit-check instruction is a kind of optimized interrupt checking instruction. It can be executed in one cycle.

A PE cache is a coherent cache with a write-back mechanism. It has two independent buffers for data and instructions. The size of each buffer is 64K bytes. The block size of the data buffer is four words.

In addition to the functions described above, the PE has functions for connecting it to the front end machine and the inter-cluster network.

The PE is implemented on a single printed board using five 80k-gate LSIs. The cycle time of the CPU is 50 ns. The execution speed of an append operation which is written in KL1-B as high as about 600K LIPS. The structure of the PE is shown in Figure 10.

In a cluster, eight PEs and a shared memory are connected via a 64-bit wide common bus. Its address space is 4G bytes. The current implementation of the cluster includes 256M bytes for the shared memory.

One unit of the PIM network system is a four dimensional hypercube network. Each node of the network has four channels. Each channel has a one-byte data line. Its throughput is 20 Mbyte/sec.

The current implementation uses two units of the network systems to increase the throughput. Each cluster is connected to four other clusters using two channels per cluster.

5.2.3 Program Execution in The Cluster

The tightly distributed KL1-B language processor uses the cluster described above. This language processor implements inter-PE communication using the shared memory. Thus, it can make the communication delay much smaller than that of the multi-PSI-V2. It is expected that the time for one transfer of message from one PE to another can be reduced to a few microseconds or less.

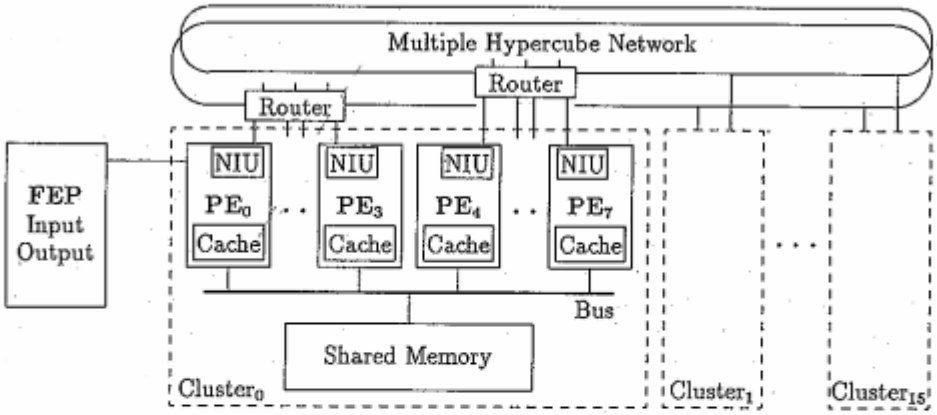


Figure 9: Configuration of PIM/p

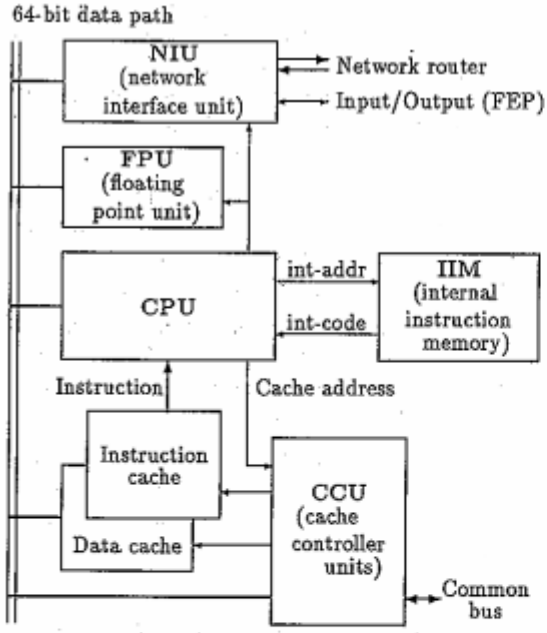


Figure 10: Structure of PE

KL1-B programs are executed in the environment described in 4.3.2, using such data structures as queues and trees. Ready queues and goal records are important data structures. The goal records which are a tree structure contain the history of program execution and are shared by all PEs in a cluster.

If one PE throws one goal to another and this goal is ready to be executed, it is put in a ready queue attached to the PE. Each PE has its own ready queue. As each goal has its own priority for its execution, each ready queue is divided into many subqueues according to the priority. There are several other data structures used for execution.

Some of these data structures are shared among PEs in a cluster. If one data structure is accessed by many PEs very often, it needs to be duplicated and allocated in separate memory areas to avoid access contentions.

Allocation of these data structures is very important to reduce bus traffic and cache mis-hit ratio. An important design criterion is to raise the locality of accesses by the optimized allocation. This enables us to make full use of the cache mechanism.

The language processor for a cluster currently has the following data structure allocation:

1. A total memory area in the shared memory is divided into several local memory areas and a common memory area. Each local memory area is assigned to each PE. The common memory area is shared by all the PEs.

2. A separate ready queue is attached to each PE and put in its local memory area because ready queues are most frequently accessed.

3. The goal records are connected by pointers and from a tree structure. This tree structure extends its branches (subtrees) according to program execution. If one goal is dispatched form one PE to another, a

new subroot is made and a new subtree grows up. In the current design, the goal record subtree is attached to each PE as shown in Figure 11.

Goals are distributed when a busy PE throws a goal when it receives a request from a non-busy PE.

The tightly distributed KL1-B language processor is under detailed design. The design is evaluated using the simulators described in 5.1.2.

6 PIM OPERATING SYSTEM: PIMOS

6.1 Outline of Research and Development

From the intermediate stage, the PIMOS was included in the plan as an important research target of this project. Before that, its role and position in the project were not clear.

Generally speaking, many researchers of parallel processing had realized that management of computational resources in connection with job distribution and load balancing was an indispensable function of parallel machines.

However, it was difficult to tell which layer of parallel machines should mainly perform this function: machine architectures, language processors, operating systems, or application programs. In research of dataflow machines for scientific computing, this function was mainly treated as a problem of machine architectures. Some hardware mechanisms were proposed for this.

In research of parallel inference machines for knowledge information processing, it seemed that machine architectures could play only a subsidiary role for this problem. Furthermore, considering the difficulty in the use of parallelism from application programs by compilers, language processors did not seem to be appropriate to embed this function in themselves. This made us decide to develop a parallel operating system for the PIM, namely, the PIMOS, although we did not have any concrete idea of embedding this function in the PIMOS. Even now, this situation has not yet changed greatly.

To start the research and development of the PIMOS, the following design policies were made:

1. A practical operating system used for large scale software experiments.

2. A stand-alone self-contained operating system.

3. A single operating system showing a parallel machine as one system.

4. To be described in KL1 and be independent from architectural details.

The PIMOS research goal in the intermediate stage was defined to have two basic management functions. One was a function for program execution management, which introduced a layered structure in program execution using meta-programming. This function is called "Sho-en". Another was a function for resource management, which manages such computational resources as CPUs, memories and input/output devices.

Before the PIMOS starts to run, it is loaded from the front end machine using the CSP functions. After it starts, it controls the entire hardware system. The front end machine is regarded as a special PE which controls input/output devices (FEP functions). The PIMOS may imagine that FEP is also running as a part of PIMOS.

To make the scope of its development as small as possible, it was decided to buid its programming environment on a PSI-II. A pseudo parallel KL1 language processor and PIMOS were built on PSI-II; however, most input/output operations including man-machine interface were performed by the SIMPOS.

This programming environment including the PIMOS built on the SIMPOS is called the PIMOS-S (PIMOS on a single processor). It is not easy for us to make many copies of the multi-PSI-V2 to distribute them to software researchers. The PIMOS-S on PSI-II will be the most popular parallel programming tool in the beginning of the final stage.

The PIMOS-S has a debugging tool that can change the order of process scheduling by random numbers so that programmers can detect bugs caused by a different execution order. The development of the programming environment described above was also included in the research goals.

Research on the PIMOS began with the design of its functional specification. As the functions of the PIMOS were closely related to that of KL1-C, the design of KL1-C was carried out concurrently. In the spring of 1987, their conceptual design was completed.

The functional design of the PIMOS and KL1-C proceeded with experimental software building so that the design could be verified. This software was built on Sequent's Symmetry system using the language C and organized into another KL1-C programming environment called the PIMOS development support system (PDSS). The PDSS has a KL1-C language processor including the Sho-en function and micro-PIMOS, both written in the language C. The functional design was completed in the spring of 1988.

In the summer of 1988, the kernel part of a KL1-B firmware interpreter began operation on a PSI-II as described in 4.3.2. The PIMOS-S also started running on the PSI-II for debugging and also for development of demonstration programs for the coming FGCS'88 conference. The PIMOS-M (The PIMOS on the Multi-

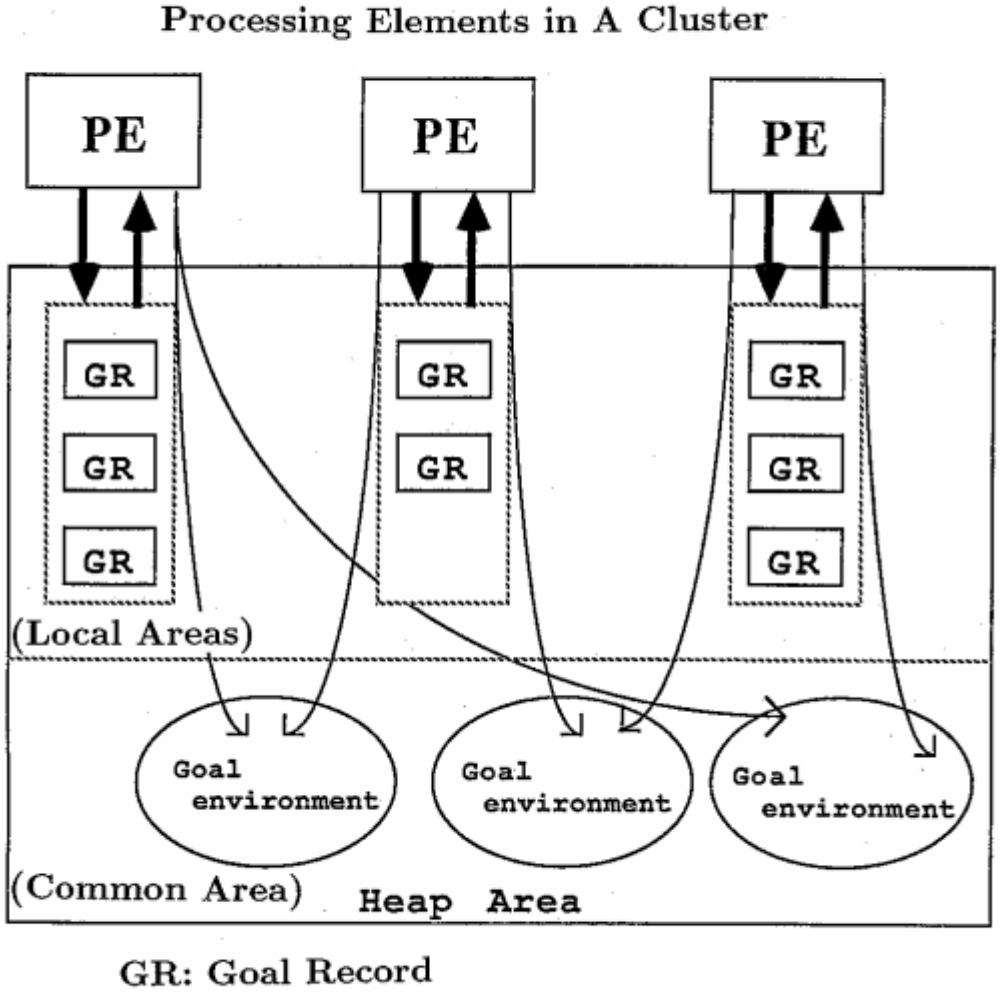


Figure 11: Execution of a KL1 program

PSI-V2) is still under development. The PIMOS-M and PIMOS-S are almost the same except that the PIMOS-M runs on a real parallel environment and will produce more bugs than the PIMOS-S because of real parallel execution of its program. These two versions of the PIMOS are planned to be released in the summer of 1989. The PIMOS-M and PIMOS-S are shown in Figure 12.

6.2 Function and Organization

6.2.1 Main Features of The PIMOS

The main role of the PIMOS is to provide its users with an efficient and safe program execution environment by managing a variety of computational resources such as computing resources, memory resources and input/output devices.

The most basic and important function of management is protection of the operating system against user program bugs. Reflecting on this function, user programs are also protected. To implement the management, structuring of program execution is indispensable. For instance, some conventional operating systems use a layered ring structure for program execution management to protect themselves from user program bugs.

In the PIMOS, this structuring mechanism was implemented as one of the program execution control functions. It was named "Sho-en". Using this function, resource management functions were implemented.

Functions for program execution control:
PIMOS needed the structuring of program execution to implement management mechanisms as described above.

However, FGHC, which was the base of KL1-C, lacked any structuring mechanism. Its execution structure is flat. Then, this mechanism was added as an execution mechanism of KL1-C using meta programming . Meta-programming separates program execution into two levels, a meta level and an object level, using a special program call called a meta-call.

In KL1-C, this call was named a "Sho-en"call. "Sho-en" in Japanese corresponds to "manor" in English. Execution of KL1-C programs repeats a call of a goal which is something like a subroutine call. The Sho-en call is a special goal call. Program execution being expanded under this call is treated as a unit of computation being managed separately. This unit is called a Sho-en. A Sho-en call seems like the entrance to a Sho-en.

A Sho-en call can be made in any Sho-en recursively. Thus, this call makes a tree-like structure in program execution. In this case, Sho-ens are nested making a parent Sho-en, children Sho-en and grandchildren Sho-en. Using this structure, program execution control functions of the PIMOS are implemented.

If a program executed in a Sho-en fails or encounters an unexpected event, it is reported to its parent Sho-en. The parent Sho-en can control the execution of the children Sho-en in many ways such as continuing and aborting. With this mechanism, protection function is realized in the PIMOS. A Sho-en is shown in Figure 13.

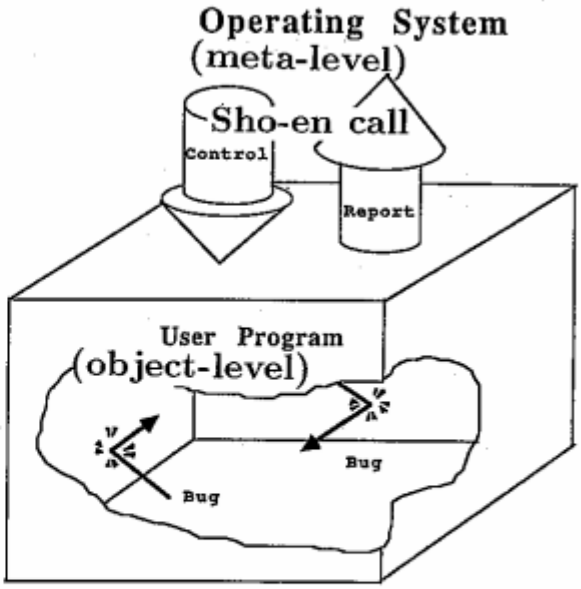


Figure 13: An image of Sho-en

Another important function of the program execution management is priority control. In the PIMOS, each parallel processe made by goal calls is given some priority in execution. This priority is used to control the order of execution. It is given to each process in two ways. One is to give it to each goal call using KL1-P. This is fine-grained control. Another is to give it to each Sho-en using a Sho-en call. This is coarse-grained control.

Functions for resource management:
The purpose of resource management of the PIMOS is to prevent unnecessary consumption of computational resources, for example, caused by program bugs such as an endless loop.

This management is performed for the following resources. One is the management of computing resources and memory resources. This is implemented using the Sho-en mechanism. Another is the management of input/output devices. They are managed by the PIMOS using a resouce tree.

A Sho-en or usage of a device is treated as a unit of management and called a task. A Sho-en is a task. For a Sho-en, the resources are managed as follows. When a Sho-en call is made, the amount of computing resources and memory resources can be specified by the parameters in the call. If this amount is used up in the Sho-en, it is reported to its parent Sho-en. Then, the program of the parent Sho-en can make it start, suspend, resume or abort.

Current implementation of the PIMOS:

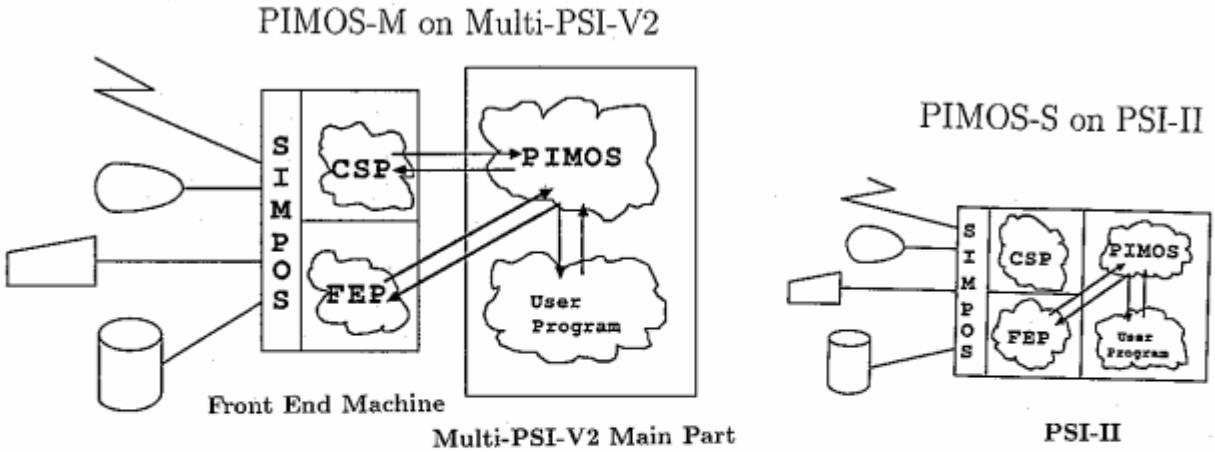


Figure 12: Organization of PIMOS

A total PIMOS system consists of a PIMOS main module, an FEP module and a CSP module.

When the PIMOS is used on the multi-PSI-V2, the main module is loaded to all the PEs before the PIMOS starts. Users programs can be loaded in two ways. One is to load a program which has been linked statically beforehand. Another is to dynamically link programs in a module database and load it to specified PEs. The contents of the module database are dynamically changed. Thus, efficient implementation is used for this.

While a user program is being executed, many Sho-ens are made. Some Sho-ens are made spanning two or more PEs. In this case, a foster parent is made in each PE in which a descendant Sho-en is made. When the program terminates, its Sho-ens also disappear. Because of the delay in packet transfer, racing will occur just as the case of reclaiming of the import/export table entries in 4.3.2. This problem is solved by a method called weighted throw count (WTC) [12].

Currently, the PIMOS is designed as a single-user multi-task operating system. To extend it to a multi-user system, additional functions must be added for appropriate distribution of computational resources among users, resolution of resource access conflict, and so on. This needs further study.

However, the current PIMOS has a function to divide PEs logically into several groups and assign them to multiple users. This function is currently sufficient to use the multi-PSI-V2 for software experiments.

7 CONCLUSION

This paper describes the research activities of the parallel inference system in the intermediate stage. They include the multi-PSI system, the PIM, KL1 language processors and the PIMOS.

The problems to be solved to develop these hardware and software systems were not clear at the beginning of the intermediate stage. As they appeared, we solved them one by one, repeating many experiments.

In this problem solving, the multi-PSI system played a much more important role than had been expected. This meant that the development of the loosely distributed KL1 language processor was much more difficult than had been anticipated.

As we tried to make the hardware systems tools for software development, their specifications had to be conservative. However, we had many difficult problems in building the hardware, especially in chip design and hardware inspection.

Finally, the PIMOS-S has partialy begun operation on a PSI-II with many bugs. Although we are still doing our best to make the PIMOS-M start on the multi-PSI-V2 before the FGCS'88 conference, we are quite sure that it will be completed and released to our users at the beginning of the final stage.

The PIM/p is now under production. It is a very complex hardware system although it consists of only two cabinets. No serious technical problems are left; however, we have to create efficient methods for its debugging and testing. Then, we shall have the fastest and the most sophisticated inference machine in the world.

The multi-PSI-V2 and the PIM/p will enable us to make much larger scale parallel software experiments in the final stage.

ACKNOWLEDGMENT

This research was conducted jointly by many researchers at ICOT and cooperating manufacturers. We

would like to express our gratitude to Dr. K. Fuchi, the director of the ICOT research center, Dr. K. Furukawa, the vice director of the ICOT research center, and Prof. H. Tanaka of the University of Tokyo, who is also the chairman of the PIM working group of ICOT, for their support and encouragement. We would also like to thank all the members of the fourth research laboratory of ICOT including Dr. E. Tick and many people at the cooperating manufacturers in charge of joint research work: Mr. Ueda and Mr. Hiratsuka at Mitsubishi Electric Co., Mr. Hattori at Fujitsu Limited., Mr. Sugie at Hitachi, Limited., Mr. Hayashi and Yamamoto at Oki Electric Industry Co., and many others.

REFERENCES

[1] T. Chikayama. Unique features of ESP. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 292–298, Tokyo, 1984.

[2] T. Chikayama. Load balancing in a very large scale multi-processor system. In *Proceedings of Fourth Japanese-Swedish Workshop on Fifth Generation Computer Systems*. SICS, 1986.

[3] T. Chikayama, H. Sato, and T. Miyazaki. Overview of the Parallel Inference Machine Operating System (PIMOS). In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, Japan, November 1988.

[4] Keith L. Clark and Steve Gregory. Parlog: A parallel logic programming language. Research Report TR-83-5, Imperial College, March 1983.

[5] A. Goto, M. Sato, K. Nakajima, K. Taki, and A. Matsumoto. Overview of the Parallel Inference Machine Architecture (PIM). In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, Japan, November 1988.

[6] A. Goto and S. Uchida. Current Research Status of PIM: Parallel Inference Machine. TM 140, ICOT, 1985. (Third Japan-Sweden workshop on Logic Programming, Tokyo).

[7] S. Habata, R. Nakazaki, A. Konagaya, A. Atarashi, and M. Umemura. Co-operative High Performance Sequential Inference Machine: CHI. In *Proc. of IEEE International Conference on Computer Design: VLSI in Computer and Processors*, Oct 1987.

[8] N. Ichiyoshi and K. Rokusawa. A New External Reference Management and Distributed Unification for KL1. TR 390, ICOT, 1988. (Also submitted to the FGCS'88).

[9] Y. Kimura and T. Chikayama. An Abstract KL1 Machine and its Instruction Set. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 468–477, 1987.

[10] A. Matsumoto et al. Locally Parallel Cache Designed Based on KL1 Memory Access Characterestics. TR 327, ICOT, 1987.

[11] R. Nakazaki, A. Konagaya, S. Habata, H. Shimizu, M. Umemura, M. Yamamoto, M. Yokota, and T. Chikayama. Design of A High-speed Prolog Machie (HPM). In *Proc. of the 12th Annual International Symposium on Computer Architecture*, pages 191–197, June 1985.

[12] K. Rokusawa, N. Ichiyoshi, T. Chikayama, and H. Nakashima. An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume 1 Architecture, pages 18–22, August 1988.

[13] M. Sato, A. Goto, et al. KL1 Execution Model for PIM Cluster with Shared Memory. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 338–355, 1987.

[14] E.Y. Shapiro. A subset of Concurrent Prolog and Its Interpreter. TR 003, ICOT, 1983.

[15] T. Shinogi, K. Kumon, A. Hattori, A. Goto, Y. Kimura, and T. Chikayama. Macro-call Instruction for the Efficient KL1 Implementation on PIM. In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, Japan, November 1988.

[16] Y. Takeda, H. Nakashima, K. Masuda, T. Chikayama, and K. Taki. A Load Balancing Mechanism for Large Scale Multiprocessor Systems and its Implementation. In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, Japan, November 1988.

[17] K. Taki. The parallel software research and development tool : Multi-PSI system. In *France-Japan Artificial Intelligence and Computer Science Symposium 86*, pages 365–381, October 1986.

[18] K. Taki et al. Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI). In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 398–409, Tokyo, 1984.

[19] S. Uchida. Inference Machines in FGCS Project. TR 278, ICOT, 1987.

[20] K. Ueda. Introduction to Guarded Horn Clauses. TR 209, ICOT, 1986.

[21] D.H.D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, Artificial Intelligence Center, SRI, 1983.

[22] K. Yoshida and T. Chikayama. A'UM–a atream -based concurrent object-oriented language. In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, Japan, November 1988.