ICOT Technical Report: TR-201

TR-201

Toward a High Performance Parallel Inference Machine
—— The Intermediate Stage Plan of PIM ——

by
A. Goto and S. Uchida

September, 1986

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456 3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Toward a High Performance Parallel Inference Machine — The Intermediate Stage Plan of PIM —

Atsuhiro GOTO Shun-ichi UCHIDA*

June 7, 1986
draft

Abstract

The parallel inference machine (PIM) is the most important hardware research target of the FGCS project. The initial stage mainly aimed to conduct R&D of individual component technologies by studying parallel inference mechanisms from various standpoints. Three basic mechanisms for PIM were studied by software simulators and by developing experimental machines with about 16 modules: the reduction mechanism, the data flow mechanism and the kabu-wake method. PIM R&D in the initial stage revealed the structures and characteristics important to an effective PIM. It also clarified many of the problems associated with the development of more practical experimental systems. In the intermediate stage, both parallel hardware mechanisms and parallel software systems will be studied based on a philosophy that integrates both the hardware and software aspects of the research. Component hardware modules will be developed with the accumulation of implementation techniques such as appropriate hardware building blocks and common software tools. Realistic software research environments will be provided by connecting PSIs to encourage kernel language implementation and parallel operating system development. Efforts to integrate them into a total PIM system will start around the middle of the intermediate stage.

1 Introduction

The FGCS Project started in June 1982 with the establishment of ICOT (Institute for New Generation Computer Technology). The project spans ten years, which is divided into three stages, namely the initial stage (1982–1984), the intermediate stage (1985–1988) and the final stage (1989–1991). The project aims at the research and development of fundamental computer technologies for knowledge information processing based on logic programming[9]. The parallel inference machine (PIM) is the most important hardware research target of the FGCS project. Its ultimate aim is to develop a machine enabling execution of parallel inference, the central concept of the fifth-generation computer.

PIM research in the initial stage[2] was mainly intended to clarify problems by investigating various parallel processing mechanisms. We studied several PIM models such as reduction and data flow for the parallel execution mechanisms of logic programming languages. Then we built several software simulators and three hardware simulators.

These experiments convinced us that parallel processing is applicable to logic programming languages and that programs could be processed in parallel if they contain parallelism in them. We produced various implementation techniques and obtained valuable know-how on implementing processing elements and for accelerating their performance.

*Fourth Research Laboratory, Institute for New Generation Computer Technology (ICOT), Mita-Kokusai Building, 21F, 4-28, Mita 1, Minato-ku, Tokyo 108 JAPAN

Table 1: Overview of PIM R&D in the initial stage

Research theme	PIM model	Experimental system
Data-flow mechanism	PIM-D	$PE \times 16, SM \times 15$ by TTL and Am2900 series
Reduction mechanism	PIM-R	$m68000 \times 16$
Load distribution	Kabu-wake system	$m68000 \times 16$

Concerning logic programming languages, we understood that it is essential to provide parallel control/communication mechanisms among inference processes running in parallel as basic language mechanisms. We decided to use GHC[18] as the basis of the kernel language (KL1), the PIM machine language. This is because GHC can be implemented more efficiently than other parallel logic programming languages. GHC is a stream-based language in which we can explicitly write synchronization among processes running in parallel. We concluded that this feature is important for the PIM operating system, although we must design the system description language by adding several important features such as a modularization function to the current GHC.

In addition, we realized the importance of cooperation between the architecture research and parallel software research. Especially in view of many extremely difficult problems, such as dividing inference tasks and assigning them to processing resources. Such problems can only be dealt with effectively by integrating software and hardware technologies. So we are pursuing parallel software research and parallel hardware research in collaboration in the intermediate stage, building the R&D basis for the final stage by integrating them.

The intermediate stage target of PIM R&D is to establish the parallel inference machine architecture for over 100 processing elements by building practical machines. We set the performance goal of *PIM*[1] at about 50–100 KLIPS[2] per processing element and 2–5 MLIPS per system, so that we can get adequate performance for running the parallel operating system (PIMOS). We also make much of the accumulation of PIM implementation techniques for the final stage. In addition, the *PIM* implementation must be stable enough for the software researchers and programmers in the final stage. As for the architecture research, our research emphasis will be on the use of locality, synchronization and scheduling, communication process by distributed unification, and stream processing.

This report first gives an overview of PIM research and development in the initial stage[2], then it outlines the research philosophy and plan for the intermediate stage, and finally it gives the current research status of *PIM*.

2 PIM Research in the Initial Stage

2.1 Objectives and Research Themes in the Initial Stage

The main objective of the initial stage PIM research was to establish the base for the PIM hardware architecture to be built in the intermediate stage (see Table 1).

However, first there were many unsolved problems in PIM architecture to be addressed.

The first important issue was to analyze the behavior of logic programs precisely. Thus, PIM R&D began with static and dynamic analysis of several sample programs written in Prolog or Concurrent Prolog[12]. Prolog was selected to describe don't-know-

[1] In the followings, "*PIM*" means the target machine we will develop in the intermediate stage.

[2] LIPS: Logical Inferences per Second

nondeterminism (OR parallelism) and Concurrent Prolog[14] to describe concurrent processes or stream processing (AND parallelism). Both languages were used as examination bases for PIM in the initial stage. The results of this analysis were incorporated in the architectural design of the following PIM models.

Research on the PIM models followed, studying various mechanisms of parallel inference from various standpoints, the objectives of which were:

- the design and evaluation of PIM models from various viewpoints,
- the trial manufacture of component modules, and
- the accumulation of sufficient evaluation data.

Several software simulators were built for each machine model, and three of them were also tested by developing experimental machines with about 16 modules.

2.2 Data Flow Mechanism (PIM-D)

In the data flow concept, each execution of an instruction starts when all necessary data are ready, resulting in parallelism regardless of whether it is explicitly indicated in the program. Therefore the data flow mechanism is expected to be a low level parallel hardware mechanism in PIM[4,6,5,7]. The PIM project selected the data flow mechanism as a candidate for approaching the parallel inference machine capable of exploiting parallelism in logic programs naturally.

PIM-D executes logic programs in a goal-driven manner: the execution of a clause is initiated when a goal is given and it returns the results (solutions) to the goal. In this execution, the PIM-D can exploit OR and AND parallelism as well as low the level parallelism in unification.

A software simulator was developed in C on the VAX to confirm the detailed structure of the PIM-D. A Prolog or Concurrent Prolog program is compiled into a data flow code, and runs on this simulator as well as on the following experimental machine.

The hardware simulator of the PIM-D was developed. This machine consists of 16 processing element (PE) modules and 15 structure memory (SM) modules, connected by a hierarchical bus network, as shown in Figure 1. Each PE consists of several APUs as execution units and an ICU, a data-driven mechanism. Each ICU, APU, and SM is made of bit-sliced microprogrammable processors (Am2900 series) and TTL ICs.

We obtained various evaluation data and know-how through this research. Using the software simulator, it became clear that the performance of the PIM-D increases in proportion to the number of PEs, if the given programs contain parallelism in them. We also found how the language features affected dynamic behavior such as memory consumption, by comparing OR-Prolog, Concurrent Prolog and GHC. As for the hardware simulator, we obtained not only the performance and dynamic behavior (Figure 2), but also gathered many implementation techniques.

2.3 Reduction Mechanism (PIM-R)

Logic programs generate several pieces of resolvent from a body goal in a clause. This can be regarded as a process in which a goal modifies itself using a clause as a rule. The reduction mechanism can also be viewed as a kind of self-modification. Considered in top-down manner as above, the reduction mechanism can be used as a basis for PIM[11,13].

The conceptual configuration is shown in Figure 3. The PIM-R consists of two types of modules, inference modules and structure memory modules, with networks connecting

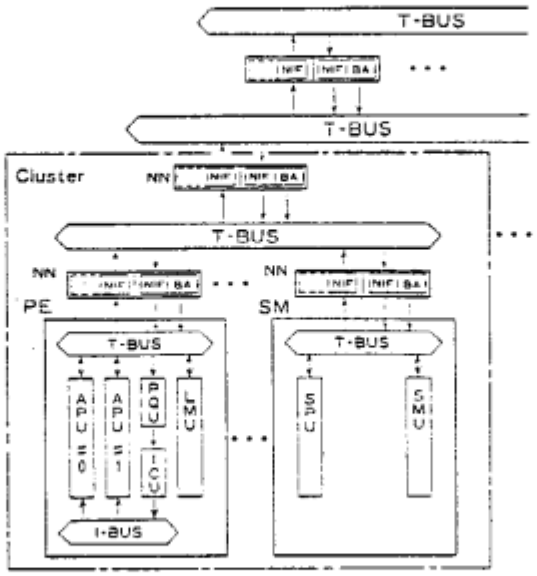

Figure 1: A Configuration of the PIM-D Hardware Simulator

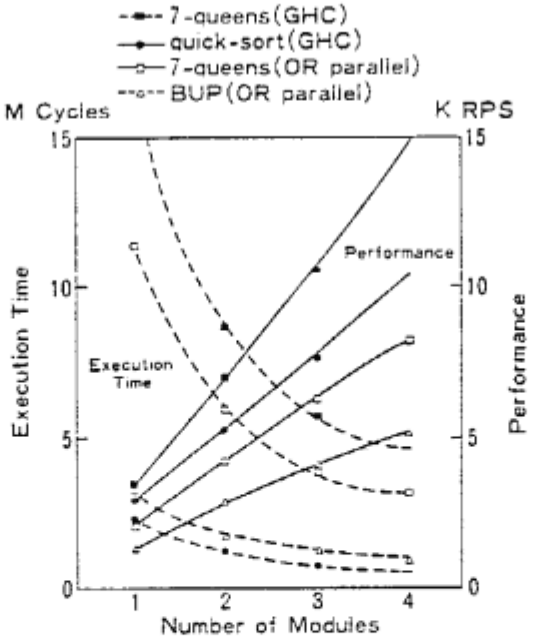

Figure 2: Performance Improvement Ratio in PIM-D Hardware Simulator

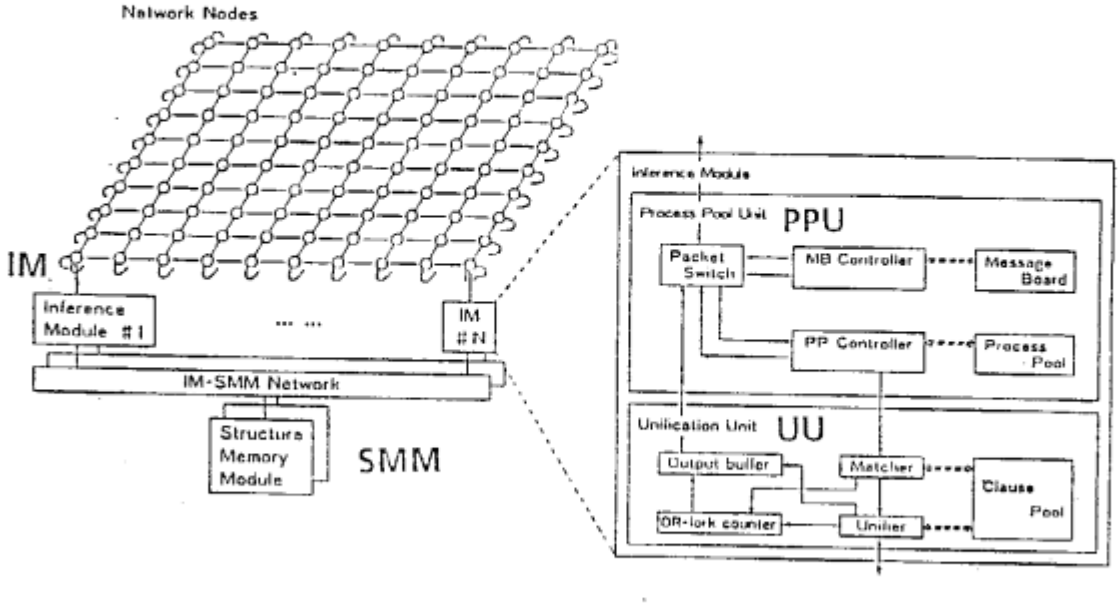

Figure 3: A Conceptual Configuration of the PIM-R

them. An inference module (IM) consists of a process pool unit (PPU) and a unification unit (UU). A PPU stores and manages reducible processes. A reducible process is sent to a UU, unified with an appropriate clause and then reducible goals are generated. The results are returned to the the PPU. The PIM-R executes Prolog programs in OR parallel and Concurrent Prolog programs in AND parallel.

A basic software simulator was developed to confirm the fundamental validity of PIM-R mechanisms, written in Prolog/C-Prolog, it runs on DEC2060 or VAX-11. A detailed software simulator was developed using occam precisely reflecting the detailed structure of the PIM-R, such as internal data formats. It also handles more than 64 IMs.

The PIM-R experimental system was built to examine the reduction mechanism. This system consists of 16 PEs (m68000 boards) connected by a common bus with a shared memory. The shared memory is used to simulate the various connection networks to be tested.

Using this software simulator we tested various methods, such as the efficiency of sharing candidate clauses or structure data, and the effect of a multi-environment. We also obtained many interesting results about the relation between the dynamic behavior, the network structure and load balancing strategies, using the hardware simulator (see Figure 4).

2.4 Kabu-wake Method

It is an important problem to examine how to divide a job, or how to distribute each piece of a job among PEs. The Kabu-wake method is one of the hardware supported mechanisms for job division and allocation in the multi-inference processor environment. The Kabu-wake method uses an effective job allocation mechanism for getting all solutions in a large tree search[8,15].

In the Kabu-wake method, each inference processor, having a job, searches solutions in

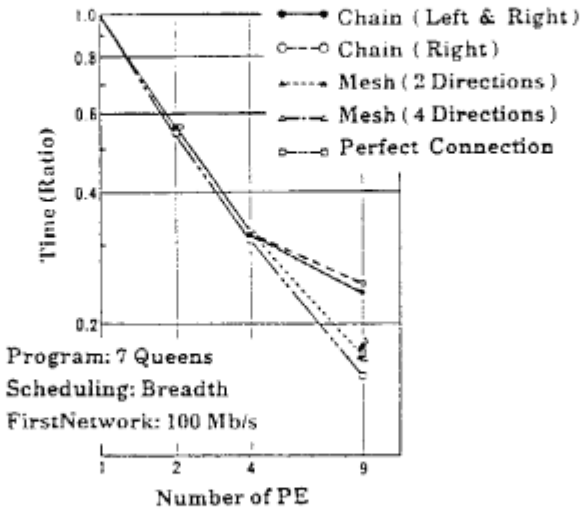

Figure 4: Dynamic Behavior in PIM-R Hardware Simulator

a depth first manner. Idle processors issue requests for jobs to the busy processors. If one processor requests a job from another processor, it splits up its own job and passes search of the remaining branches of the tree to the other processor, so that they perform OR-parallel inference. This execution feature is expected to minimize job allocation overhead among inference processors.

The experimental system was built to test the effectiveness of the Kabu-wake method quantitatively. The hardware configuration is shown in Figure 5. The system consists of 16 PEs (one PE for input/output), connected by two kinds of exclusive networks; CONT-network and DATA-network. The CONT-network is a ring network for job requesting packets. The DATA-network is a high throughput switching network for transferring a split job (kabu). We obtained various interesting results (Figure 6), on relationships between the size of given problems, network traffic and system performance.

The compiler has been developed for the Kabu-wake method, so that each PE can execute given jobs more quickly. New evaluation results will be published in near future.

2.5 Approaches of PIM Models Research

The approaches of the above three PIM models can be summarized as in Figure 7. First let us consider PIM-D and PIM-R research. In the PIM-D research, we aimed to apply the data flow mechanism to logic programs as a basic hardware mechanism. This is the reason we regard the PIM-D research as a bottom-up approach. The PIM-R research started by regarding the execution models of logic programs as reduction. Thus, PIM-R research can be seen as a top-down approach. On the other hand the Kabu-wake method was aiming to approach the PIM architecture from the view point of load division and balancing that is one of the important functions in parallel inference processing.

We should integrate these research results in PIM R&D in the intermediate stage, rather than simply compare them and select one approach. This is because we found

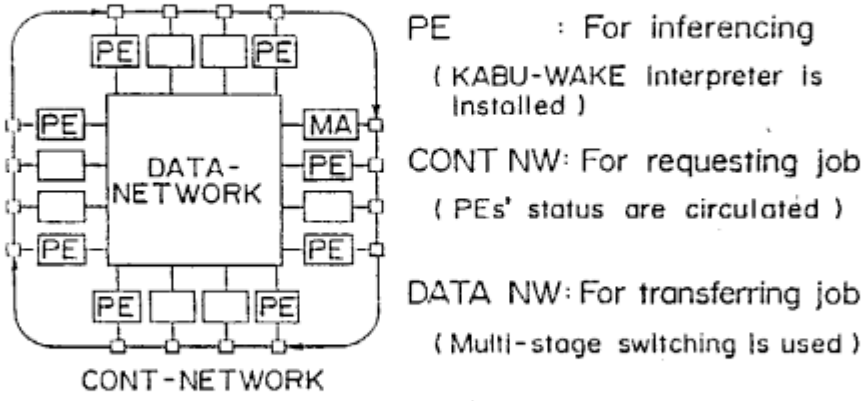

Figure 5: The Kabu-wake Method Experimental System

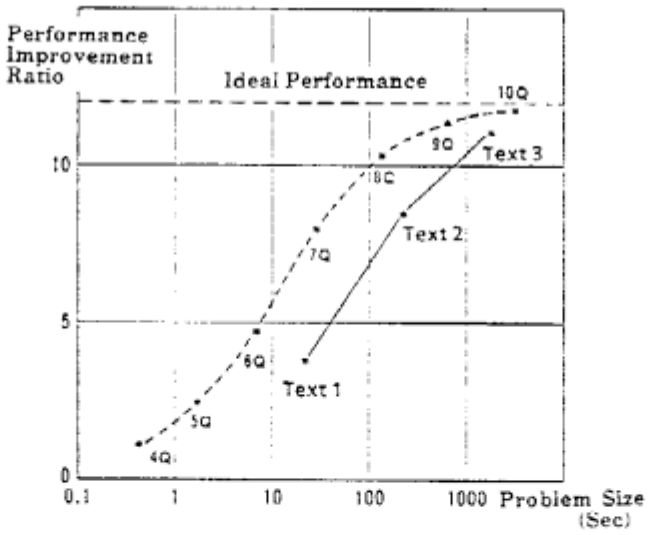

Figure 6: Problem Size and Performance Improvements Ratio in Kabu-wake Method

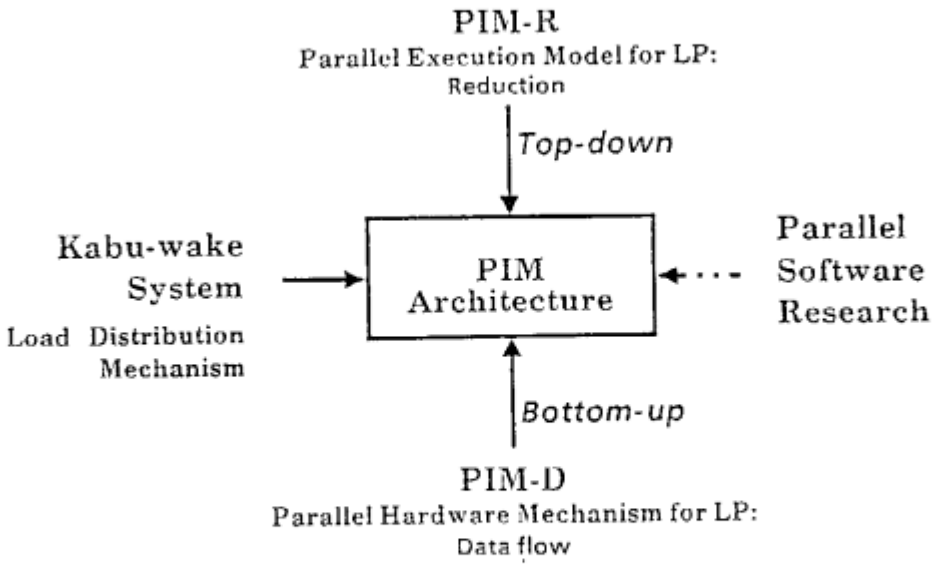

Figure 7: Research Approaches in the Initial Stage

many similar features in the parallel processing mechanisms of PIM-D and PIM-R. In addition the concept of the Kabu-wake method can be amalgamated with others as a load balancing method.

2.6 Subjects clarified in the initial stage

In the initial stage research for PIM models, we found that parallel processing is applicable to logic programming languages and logic programs. We also obtained various implementation techniques and know-how for implementing and accelerating processing elements' performance. The following are the issues in future PIM research.

The first is the cooperation with parallel software research. This cooperation covers:

- large scale application programs,
- parallel algorithms,
- operating systems for parallel inference machines,
- debugging methods for parallel software, and
- programming environments for parallel software.

As for the hardware architecture, we emphasize:

- increasing elementary processor performance,
- accumulation of implementation techniques, and
- implementation know-hows sufficient for using as hardware building blocks in future.

We plan to attempt:

- implementation of a network for over 100 modules,

- communication function in processing elements,
- highly integrated implementation of elementary modules, and
- development of reliability and debugging methods for parallel hardware,

so that the PIM can work effectively with high performance. The above discussion clearly illustrates the vital importance of emphasizing the architectural research from parallel software viewpoint as well as the integration of the previous three kinds of architectural research.

3 Intermediate Stage Plan

Parallel processing research should be performed as a close collaboration of the work on application fields and problems, algorithms and parallel processing systems. The parallel processing systems must be a stable base for research, and have a consistent hardware and software view.

PIM R&D in the intermediate stage, which started last year, is being performed according to the following plans that integrates both the hardware and software aspects of the research.

3.1 Basic Philosophy in the Intermediate Stage

PIM R&D in the initial stage has clarified many problems associated with the development of more practical experimental systems.

- Integration of Software and Hardware Research

In the design of a total system architecture, functions of the hardware part and the software part of the system must be efficiently divided so that the hardware part can be optimized, and thus simpler and faster. This requires study on the parallel software system for static and dynamic resource allocation, parallel job monitoring, and also the implementation of parallel language interpreter for KL1. Thus, the intermediate stage plan includes such parallel software research mainly aiming at the development of the parallel operating system (PIMOS). To encourage this research activity, the development of multi-PSI systems is planned to provide software researchers with more realistic research environments. Figure 8 shows the relationship between parallel software research and the PIM architecture research.

- Consistency through Hardware and Software

PIM research needs to give priority to software requirements, especially from PIMOS. This is because the machine structure should be adaptable for the PIMOS which monitors hardware resources and allocates jobs. Additionally, *PIM* must have total performance sufficient to encourage the next stage software research. In order to increase total performance, it is necessary that all hardware elements should balance with and enhance each other.

- Continuity to the Final Stage

Figure 9 shows research continuity from the intermediate to the final stage. In this, the cooperative research between software and hardware is the most important point. In addition to this, accumulation of implementation techniques are also important to build faster

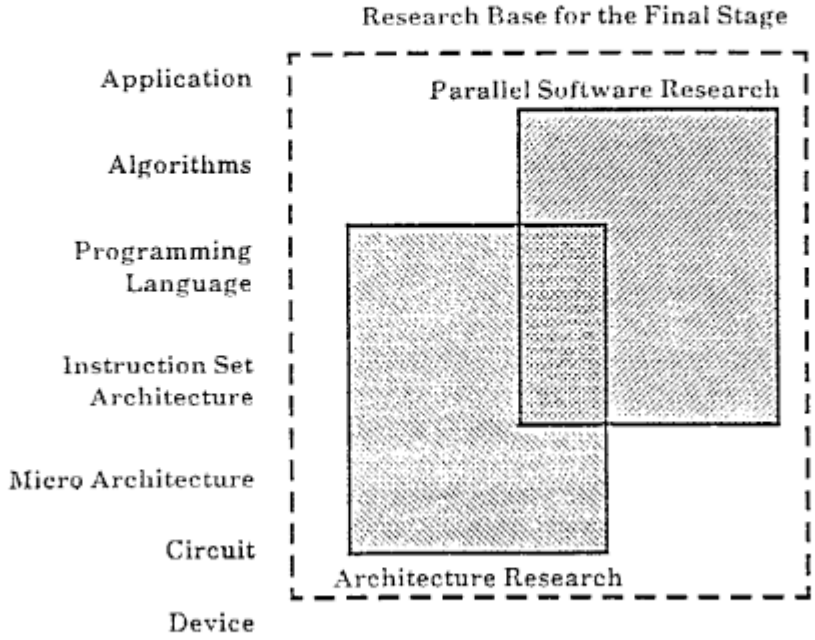

Figure 8: Software and Hardware Research Cooperation

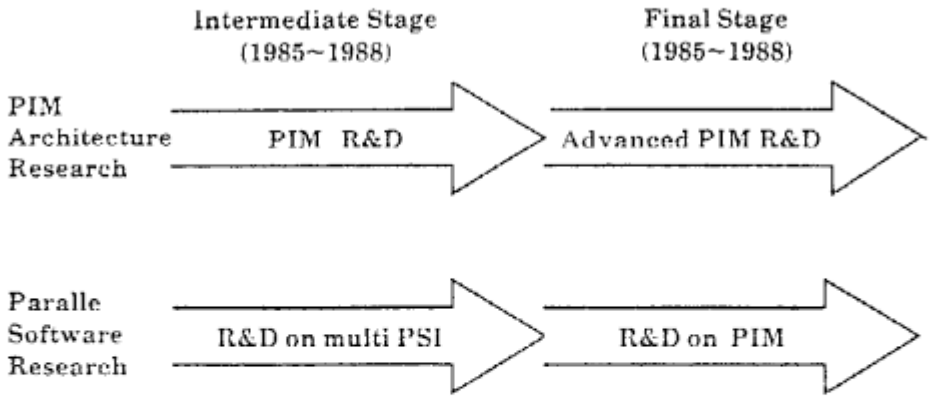

Figure 9: Continuity from the Intermediate Stage to the Final Stage

and smaller hardware components and reduce labor for hardware debugging and maintenance in the design of component hardware modules. Large-scale PIM systems require stable and easy-to-handle hardware elements with software tools for simulators and debugging tools. Thus, the intermediate stage plan includes an effort to find out appropriate hardware building blocks and common software tools to make hardware development a little more comfortable.

3.2 Research Subjects for the PIM Software and Hardware System

In the intermediate stage, both hardware architecture and software systems will be developed for the PIM. The research subjects in the intermediate stage are summarized below.

(1) Large Scale PIM Architecture Research

The inter-PE parallel processing mechanism, in particular, a highly parallel connection network and its control mechanism, will be studied in cooperation with the PIMOS R&D. The intermediate research goal is an experimental machine *PIM* consisting of about 100 PEs on which PIMOS will run.

R&D of High performance elementary processor for PIM: First we will begin by studying the parallel processing mechanisms of KL1. Next, an efficient virtual machine code, called KL1-B, will be designed.

Building Blocks for PIM: The PIM research in the initial stage revealed clearly that commonly used hardware elements should be developed to enhance and to ease the R&D of PIM. Experimental processors, parallel memory systems, and networks will be designed in detail, through developing commonly used elements, such as multi-port page memory, packet send/receive hardware modules, and tag handling hardware.

Developing tools for PIM: The research environment plays an important role in PIM R&D. Researchers usually develop their own tools, such as software simulators, for their own objects. Although these tools have similar functions and structures, they are not always passed around among researchers. PIM R&D will study several alternatives. Therefore it seems valuable to develop commonly used software tools as a development base.

(2) PIM Software System Research

R&D of software development pilot machines: The following workbenches will be developed at an early stage to study parallel software systems for PIM. They are called multi-PSI systems. The PSI[10,17,20] is a personal inference machine developed by ICOT and it is expected to be a prime candidate processor for parallel software development systems here.

- Pseudo-multi PSI : a simulator on a PSI machine,
- multi PSI v.1 : 6–8 PSI system,
- multi PSI v.2 : 16–64 PSI-II[3] system,

[3]The PSI-II is a new PSI system, which is currently being developed. The PSI-II will have better performance than the present PSI with a smaller size

Pseudo-multi PSI and multi PSI v.1 will be available in the middle of 1986, and multi PSI v.2 in the end of 1987. PIM applications as well as the following language system and operating system will be developed first on these workbenches step by step. Then they will be integrated on *PIM*.

R&D of the PIM language systems: Kernel language systems for PIM will be hierarchically developed by extending GHC[18]. GHC is a logic programming language enabling parallel programming. A high-level language for system programming, called KL1-U, will have parallel object concepts. The PIM kernel language, called KL1-C(P), will be a machine independent low-level language enabling pragmatic control.

R&D of the operating system for PIM (PIMOS): PIMOS, the operating system for PIM, will be developed to facilitate resource allocation and management from the software. First, we will attempt to describe stream-based input/output facilities and goal scheduling based on the locality of multi-PSI system configuration.

4 KL1 and Parallel Software Development on Multi PSI

4.1 KL1 Language Feature

As described above, KL1 can be regarded as a super-set of GHC. GHC has various unique features. In view of logic programming language, clauses in GHC programs are selected in pattern-driven manner as in Prolog, however unification of logical variables are performed in single assignment manner. Parallel processing are described in GHC programs as follows: programmers can describe various processes of flexible size in GHC, communications among such processes are realized using logical variables, and GHC has simple language principles for parallel process synchronization. We briefly show such language features of GHC, before describing the design feature of the PIM machine language. A detailed description of GHC can be found in [18].

A GHC program is a finite set of guarded Horn clauses of the following form:

$$H :- G_1, \ldots, G_m | B_1, \ldots, B_n . (m \geq 0, n \geq 0)$$

where H, G_i's, and B_i's are called *a clause head*, *guard goals*, and *body goals*, respectively. The operator '|' is called a commitment operator. The part of a clause before '|' is called a passive-part (or guard), and the part after '|' is called an active-part (or body). A guarded clause with no head is a goal clause, as in Prolog.

The semantics of GHC is quite simple. Briefly speaking, execution of a GHC program proceeds by reducing a given goal clause to the empty clause under the following rules[4]:

Rule 1: Any piece of unification in the guard of a clause cannot instantiate a variable in the caller.

Rule 2: Any piece of unification in the body of a clause cannot instantiate a variable in the guard, until that clause is selected for commitment.

Rule 3: When there are several clauses of the same head predicate (candidate clauses), the clause whose guard is first succeeded is selected for commitment.

Rule 1 is used for synchronization. Rule 2 guarantees selection of one body for one invocation, and Rule 3 can be regarded as a sequencing rule for Rule 2. Under the above rules, each goal in a given goal clause can be reduced to new goals (or null) in parallel.

[4]These rules are informal. The formal rules can be found in [18].

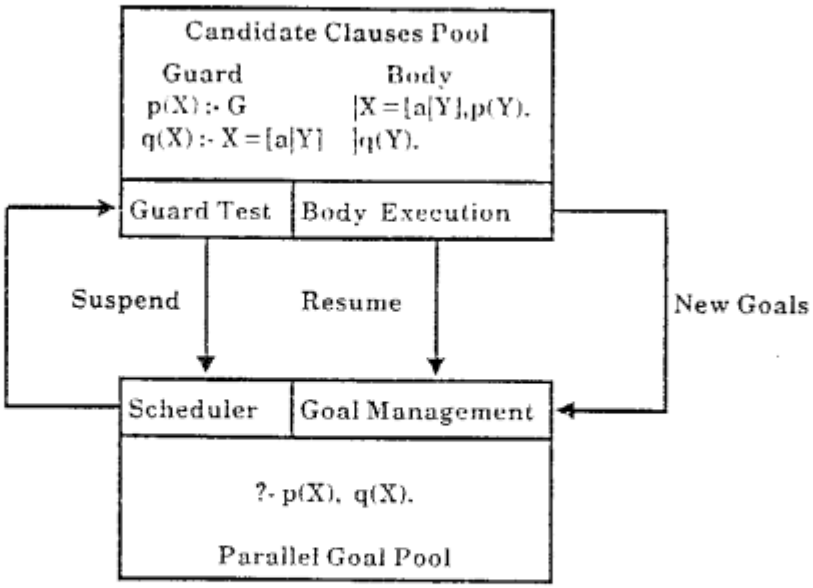

Figure 10: Processing Mechanism of KL1

4.2 Processing Mechanism of KL1

As described above, it seems to be natural to regard the processing mechanism of KL1 as reduction. Figure 10 shows the execution feature of KL1. Assuming that there is a goal clause with two goals[5] $p(X)$ and $q(X)$ in the goal-pool, the scheduler picks up one of the parallel goals in the goal-pool first. Then its passive part is checked. Both goals may be picked up. If the execution of the guard of $p(X)$ ends successfully, its body is selected. Then the variable X is instantiated to $[a|Y]$, and a new goal $p(Y)$ is generated. This new goal is returned to the goal-pool and registered within a kind of goal managing structure.

If $q(X)$ is executed before $p(X)$, execution of $q(X)$ is suspended. This is because the unification of $q(X)$ with a candidate clause needs to instantiate the variable X. Such a goal, waiting for variables to be instantiated, is called a *suspended goal*[6]. In the case of Figure 10, the suspended-goal $q(X)$ will be resumed by the execution of $p(X)$.

4.3 KL1 Language System

KL1 is a hierarchical language system: KL1-U, KL1-C and KL1-P, and KL1-B, as shown in FIgure 11. The KL1 R&D group members are now specifying the each language specification and implementing on PSIs and some conventional machines.

KL1-C[7] is a core language system. KL1-C is based on flat-GHC with built-in predicates and *meta-calls*. Flat-GHC is a subset of GHC, whose guard goals are all built-in predicates. This restriction makes its language implementation more efficient keeping its description power. Meta-calls are special functions to enable programmers to handle the logical values of goals.

KL1-P[8] is the attached language functions to KL1-C such as job allocation and goal

[5] These goals are called *parallel goals*.

[6] It is natural to assume that these suspended goals are returned to the goal-pool, waiting for the variable instantiations.

[7] 'C' is an abbreviation for 'core'.

[8] 'P' is an abbreviation for 'pragma'.

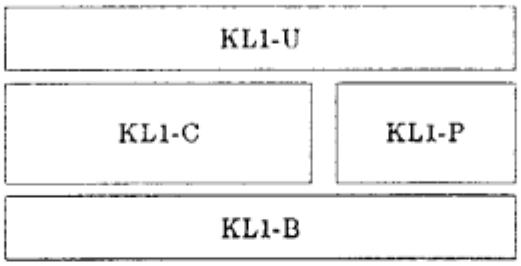

Figure 11: The KL1 Language Systems

priority control. KL-P may depend lower level hardware construction such as the network topology of PIM. So these functions exceed the logical framework of GHC, however they are necessary to describe the operating system (PIMOS).

KL1-U[9] is a high-level system programming language for system programmers. KL1-C and KL1-P specify the overall language functions of KL1. Such functions are included in KL1 with some modular programming concepts. KL1-U will be extended to have a parallel object oriented.

KL1-B[10] is a virtual machine code interfacing between the *PIM*and KL1. So KL1-B can be regarded as a compiler target language of KL1-U and KL1-C with KL1-P. KL1-B also includes some special functions to directly control and maintain the *PIM*hardware. In an actual programming environment, most system programmers are expected to develop using KL1-U, then the compiler systems from KL1-U to KL1-B are very important.

4.4 PIMOS and Process Allocation

Operating systems play an important role in recent computer systems, however parallel computer researchers have seldom considered them as main research issues. So we will develop the experimental version of PIMOS on multi PSI systems first. Then the functions of PIMOS will be stepped up.

The PIMOS research must solve many difficult problems, such as:

- process allocation and load balancing, using localities,
- hardware resource management,
- object program codes allocation and their management,
- user task management, and
- input/output functions.

In them, process allocation is the most difficult research issue. We studied it only from hardware architecture point of view in the initial stage PIM research. Then we found that we should also solve this problem from PIMOS point of view. It means that programmers should design the parallel algorithms, considering their communication localities, and describe programs with hints for suitable load allocation to PIMOS. Here programmers should only assume the abstract image of PIM, which is a kind of homogenius processing power plane with logical distance of communications. (See Figure 12) As shown above, KL1-P is used to present such hints. Then the PIMOS allocates processes corresponding

[9] 'U' is an abbreviation for 'user'

[10] 'B' is an abbreviation for 'base'

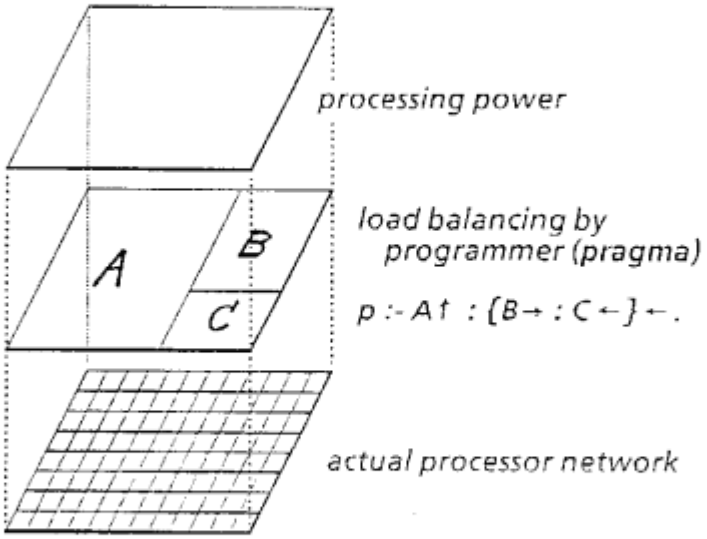

Figure 12: Process Allocation Strategy

to the given hints. Of course, the processing load may be unbalanced as execution goes on. In such case, PIMOS re-allocates processes considering their localities.

4.5 Multi PSI Systems

Multi PSI systems are the workbenches for studying parallel software systems on PIM. Pseudo-multi PSI is a simulator on a PSI. It will be used to develop multi PSI system's software and test them. Multi PSI v.1 is the 6-8 PSI system, connected via a network hardware whose transfer rate is about 500K Byte/sec, as shown in Figure 13 and 14. The network hardware is installed in each PSI's CPU option slot. KL1 is implemented by ESP[1] on SIMPOS[16]. Then parallel unification and PIMOS will examined on it. Multi PSI v.2 will be the 16-64 PSI-II system, more tightly connected than multi PSI v.1. On this machine, KL1 will be fully implemented by firmware, so that parallel application programs can be develop on PIMOS.

5 Overview of PIM Architecture Research

5.1 Target Machine Specification

Our target machine *PIM* will consist of about 100 processing elements. We set the performance goal of the *PIM* at about 50–100 KLIPS per processing element and 2–5 MLIPS per system, so that we can get adequate performance for running the parallel operating system (PIMOS). We also give weight to the accumulation of PIM implementation techniques. The *PIM* hardware system must be stable enough for the parallel software research in the final stage. We will implement the *PIM* mainly using gate-arrays and some custom LSI's.

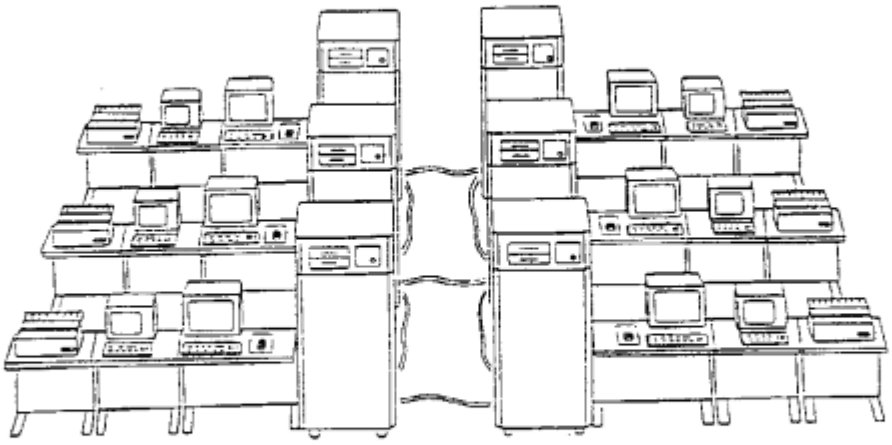

Figure 13: The multi PSI version 1

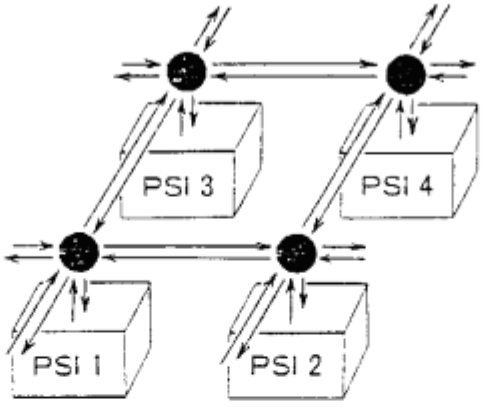

Figure 14: Connection Hardware of multi PSI

5.2 PIM Machine Language

We are now designing the *PIM* machine language, called KL1-B, considering the above execution features of KL1. In this case KL1-B can be regarded as a virtual machine code and its interpreter. KL1-B should be designed putting emphasis on the following points.

(1) Synchronization and Scheduling

As described above, it is important to provide effective mechanisms for synchronizing and scheduling parallel goals. The internal structure of a goal should be examined first. Next, several control structures such as a scheduling queue and a goal tree should be designed. A goal tree is used to manage the logical relationship of parallel goals. To resume suspended goals effectively, it is necessary to provide a kind of bind-hook mechanism, and to use multiple scheduling queues with priorities. KL1-B has some synchronization primitives for suspending/resuming and scheduling mechanisms.

From the viewpoint of processing element design, functions such as suspending, resuming, or scheduling goals can be regarded as context-switching between goal reductions Therefore it is necessary to support efficient context-switching as well as to provide an effective strategy to decrease their occurrence.

(2) Communication

Communication among processing modules is necessary for load balancing in the system, remote data access in distributed unification. and resuming goals between processors. In order to realize such communication efficiently, both shared buffer communication mechanisms and packet communication mechanism are being examined. In addition it is important to keep processing elements busy during communications latency. So we should also consider low-cost context-switching mechanisms such as concurrent virtual machines in each processing element to minimize communication overhead.

(3) Streams

Stream programming is one of the important paradigms in KL1 and should be supported by low-level procedures. An efficient stream implementation technique, such as CDR-coding, and its primitives will be provided in KL1-B.

(4) Locality in the Problem

Application programs for PIM consist of various sized activities (parallel processing components). Some of them are lager than processing elements, and some are smaller. In general, the former may be treated as the problem of communications locality among processors, and the latter as the problem of parallel processing granularity. Initial stage PIM R&D pursued rather fine-grained parallel processing. In the intermediate stage, larger sized granules should be also considered in cooperation with the modularity of programs.

Fine-grained locality will be treated as follows. KL1 semantics can express both fine-grained activities and large-grained activities. However, in practical programs, it seems unnecessary to handle very fine-grained activities just as they are. Therefore KL1-B should be designed for internal execution in a processing element, assuming optimizing compilers. For example, a scheduling method to execute a goal repeatedly in each processing element is introduced in KL1-B. The machine instructions will be designed at a low level like Warren's code[19] or even lower than that. The parallel cache mechanism is important for the hardware mechanism.

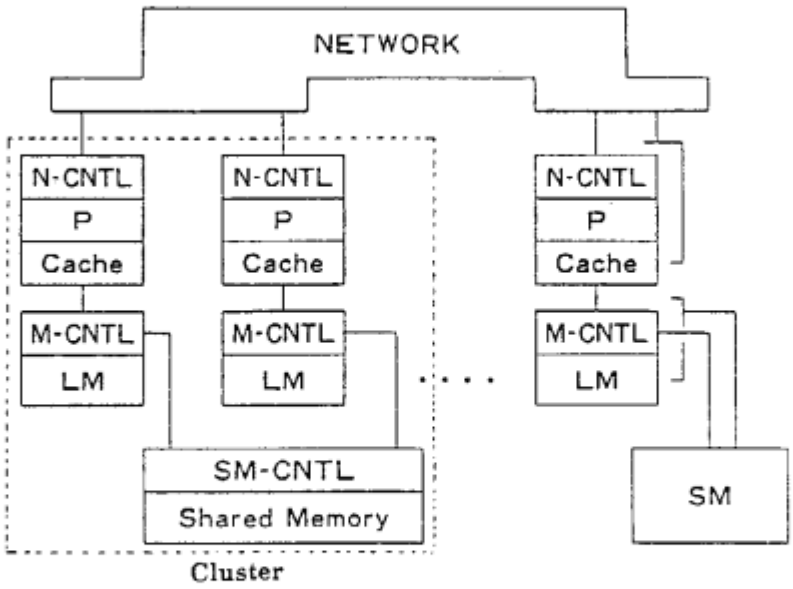

Figure 15: The Hardware Construction Image of PIM (1)

As for the former locality, the important issue is hardware architecture whose locality can be easily handled in software (i.e. by PIMOS). We will design the machine language with load distribution primitives. Then we will introduce the concept of clusters, and study the parallel memory and network system for shared buffer communication.

5.3 Construction of the Intermediate Stage PIM

(1) Overall design and cluster concept

Figures 15 and 16 show the two overall construction images of *PIM* that we are now designing. Both machines have cluster concepts. Physical shared memories form clusters in Figure 15 and a global address space is distributed in each processing element memory to form clusters in Figure 16. In both construction images there are about 10 processing elements in a cluster.

The concept of clusters will be introduced in two senses: the logical cluster and the physical cluster. The logical cluster is a group of processing elements that have one address space. These processing elements share their data space such as parallel goal environments. Thus some address transformation tables and their management will be necessary for inter-logical-cluster communication. By introducing logical clusters, garbage can be collected in each logical cluster. This is because inter-cluster pointers are gathered in such tables and their entries can be used as roots of inter-cluster references. The physical cluster is the group of processing elements which are connected closely from the view point of physical implementation. In such a physical cluster, each processor can communicate with each other faster than with processors in another physical cluster.

In Figure 15, a cluster is a group of processors connected with the same shared memory, and the inter-processor network is a two level network. These processors communicate with each other using both an intra-cluster network (i.e. a lower-level network) and their shared memory. The network response is more important than throughput for an intra-cluster network communication. These clusters are connected with each other by a inter-cluster network (a upper level network). High-throughput networks like cross-bar networks are

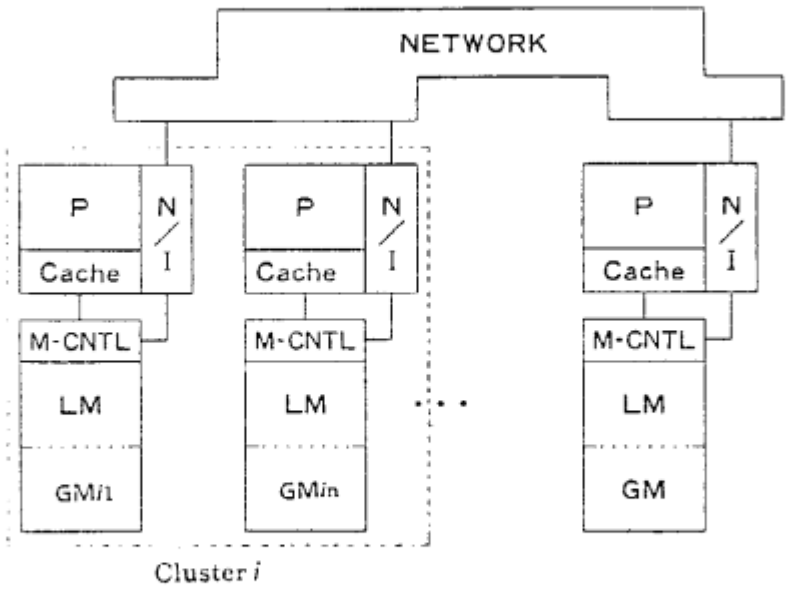

Figure 16: The Hardware Construction Image of PIM (2)

suitable and available because there are about 10 clusters in the *PIM*.

In Figure 16, a cluster is a group of processors whose global memories (GM) have a same address space. So $GM_i1 \ldots GM_in$ in the $cluster_i$ form one global address space. The inter-processor network can be designed in one level network. However a two level network is better to make the best use of the locality in application programs.

(2) Memory system design

In the design of the *PIM* hardware configuration, it is important to design the parallel memory systems such as private cache memories and global shared memories. Generally parallel cache mechanisms have the so-called cache coherence problem[3], so that we will design efficient parallel cache hardware to overcome this problem. In addition to this, it is important how to properly use such memories as cache, local and shared (or global), and how to reflect the localities of KL1 execution. So we are studying specialized memory mechanism for KL1 execution and extending cache concepts to reflect the localities.

(3) Processor design

Important issues to design processing elements are tag architecture for unification, efficient context-switching mechanisms, and interrupt handler for inter-processor communication. The experience of PSI R&D helps us to design the tag handling mechanism in processing elements. In particular context-switching and interrupt handling are key issues to enable the communication between parallel goals. Context switching will occur both in goal suspension and in unification with remote data. The concept of concurrent virtual machines will be introduced to realize efficient context-switching. Concurrent virtual machine mechanism can be regarded as a logical cache of of goal contexts.

6 Conclusion

This report gave a research and development overview of PIM in the initial stage, tentative plans for the intermediate stage, and the current research status. In the initial stage, three basic mechanisms for PIM were studied with software simulators and experimental machines. In the intermediate stage, we will study both parallel hardware mechanisms and the parallel software system. Efforts to integrate them into a total PIM system will start around the middle of the intermediate stage.

Acknowledgment

The research and development described in this article are being conducted mainly by the members of the PIM, multi-PSI and KL1 groups both in the ICOT Research Center and the participating companies. We also wish to thank to ICOT Director Kazuhiro Fuchi for valuable suggestions and guidance.

References

[1] T. Chikayama. Unique features of ESP. In *Proc. of the International Conference on Fifth Generation COmputer Systems*, Tokyo, 1984.

[2] 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).

[3] K. Hwang and F.A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.

[4] N. Ito, A. Kishi, E. Kuno, and K. Rokusawa. The Dataflow-Based Parallel Inference Machine to Support Two Basic Languages in KL1. In *IFIP TC-10 Working Conference on Fifth Generation Computer Architecture*, July 1985.

[5] N. Ito and K. Masuda. Parallel Inference Machine Based on the Data Flow Model. In *Proceedings of International Workshop on High-Level Computer Architecture*, pages 431–440, Los Angels, May 1984.

[6] N. Ito, M. Sato, A. Kishi, E. Kuno, and K. Rokusawa. The Architecture and Preliminary Evaluation Results of the Experimental Parallel Inference Machine PIM-D. In *Proc. of the 13th Annual International Symposium on Computer Architecture*, June 1986.

[7] N. Ito, H. Shimizu, A. Kishi, E. Kuno, and K. Rokusawa. Data-flow based execution mechanisms of Parallel and Concurrent Prolog. *New Generation Computing*, 3(1):15–41, February 1985.

[8] K. Kumon, H. Masuzawa, A. Itashiki, K. Satoh, and Y. Sohma. Kabu-wake: A New Parallel Inference Method and its Evaluation. In *COMPCON Spring 86*, pages 168–172, IEEE Computer Society, San Francisco, March 1986.

[9] K. Murakami, K. Kakuta, R. Onai, and N. Ito. Research on parallel machine architecture for Fifth-Generation Computer Systems. *IEEE Computer*, 18(6), June 1985.

[10] K. Nakajima, M. Yokota, K. Taki, S. Uchida, H. Nishikawa, A. Yamamoto, and M. Mitui. Evaluation of PSI Micro-Interpreter. In *COMPCON Spring 86*, pages 173–177, IEEE Computer Society, San Francisco, March 1986.

[11] R. Onai, M. Aso, H. Shimizu, K. Masuda, and A. Matsumoto. Architecture of a Reduction-Based Parallel Inference Machine:PIM-R. *New Generation Computing*, 3(2):197–228, June 1985.

[12] R. Onai, H. Shimizu, K. Masuda, and M. Aso. *Analysis of Sequential Prolog Programs*. TR 048, ICOT, May 1984.

[13] R. Onai, H. Shimizu, K. Masuda, A. Matsumoto, and M. Aso. Architecture and Evaluation of a Reduction-Based Parallel Inference Machine: PIM-R. In *Lecture Note in Computer Science*, Springer-Verlag, to appear.

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

[15] Y. Sohma, K. Satoh, K. Kumon, H. Masuzawa, and A. Itashiki. A New Parallel Inference Mechanism Based on Sequential Processing. In *IFIP TC-10 Working Conference on Fifth Generation Computer Architecture*, July 1985.

[16] S. Takagi, T. Yokoi, S. Uchida, T. Kurokawa, T. Hattori, T. Chikayama, K. Sakai, and J. Tsuji. Overall design of SIMPOS. In *Proc. of the Second International Logic Programming Conference*, Uppsala, 1984.

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

[18] K. Ueda. *Guarded Horn Clauses*. TR 103, ICOT, 1985.

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

[20] M. Yokota and et al. A Microprogrammed Interpreter for the Personal Sequential Inference Machine. In *Proc. of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1984.