ICOT Technical Report: TR-324

TR-324

Parallel Inference Machine Research in FGCS Project

by
A. Goto

November, 1987

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

Parallel Inference Machine Research in FGCS Project

Atsuhiro GOTO
Fourth Research Laboratory,
Institute for New Generation Computer Technology (ICOT) *

Abstract

The parallel inference machine (PIM) is the most important research target of the FGCS project. The initial stage(1982–1984) mainly aimed to conduct the research and development(R&D) of individual component technologies by studying parallel inference mechanisms from various standpoints. PIM R&D in the initial stage revealed the structures and characteristics important and effective for PIM. It also clarified many of the problems associated with the development of more practical experimental systems, e.g. how to how to start and stop user programs and how to perform input/output.

In the intermediate stage(1985–1988), both parallel hardware mechanisms and parallel software systems have been studied based on the framework of logic programming. KL1, the kernel language system of PIM, is designed based on a parallel logic language GHC. The operating system for PIM is written in KL1. Realistic software research environments are provided by connecting the processors developed as the personal sequential inference machines (PSIs) to encourage kernel language implementation and parallel operating system development.

The most important policy of the PIM architecture is to make the best use of communication locality in given applications. Therefore PIM is designed accepting parallel software requirements. In addition PIM hardware architecture is designed by accumulating implementation techniques.

1 Introduction

The parallel inference machine PIM aims to be at the frontier of parallel processing in artificial intelligence (AI) application fields[7]. The principal aim of parallel processing is to increase the execution performance so that users will be able to solve big application programs. AI machines should have many additional features compared with conventional general-purpose machines. For example, pattern matching operations are important in many AI applications. However, it is insufficient to increase the efficiency of only the limited functions in AI application. In other words, development of AI machines should strive not to make singular machines, e.g. a conventional machine with pattern match accelerator, but to pursue more general and powerful machines than conventional ones. AI machines should also cover the functions of

*Mita-Kokusai Building 21F., 4-28, Mita 1, Minato-ku, Tokyo 108 JAPAN, CSNET: goto%icot.jp@relay.cs.net, ARPA: goto%icot.uucp@eddie.mit.edu, UUCP: ihnp4!kddlab!icot!goto

conventional computers, because AI machines are not simply game tree searching machines. The personal sequential machine (PSI[33,14,27]) developed by ICOT is an example of AI machines with general functions. The operating system of PSI, SIMPOS[24], is written in the logic programming language ESP[4] in which many application programs are also written. SIMPOS includes general function such as a window system and local-area network system. Many built-in predicates are provided for not only AI applications but also these SIMPOS functions. As the result, the total performance PSI including programming environments increased.

Parallel machine architecture research to date has explored many new technologies[8], such as interconnection networks[3], dataflow computing mechanisms, structure memory mechanisms, and their hardware implementations[31]. Considering the integration of technology into a total and actual system, however, there remain many unsolved problems. For example most data structures used in AI programs are dynamic data structures which need efficient memory management and garbage collection functions. It is important to manage processing loads which may vary dynamically. In addition, process resource management is essential to use the parallel machines as practical tools for AI research. It is difficult to solve such problems in parallel processing only by discussing the machine architectures. These problems must be solved by integrating parallel software and hardware architectures.

In the intermediate stage of the ICOT FGCS project, the research and development for the parallel inference machine system is based on the framework of logic programming. The kernel language system is designed, which includes a user-level language, a core language and a machine level language. Parallel inference machine optimized to the kernel language is studied and being developed. The parallel operating system is designed as *self-contained* operating system in the kernel language. To enhance the kernel language and the operating system implementation, parallel machine workbenches are developed. This report gives an overview of PIM research.

2 PIM Research Directions

2.1 Research issues

[1] Parallelism and Granularity

In designing parallel processing systems, architects must consider the trade-offs between the effect of parallel processing and the overhead in performing parallel processing. Parallelism must be discussed from the viewpoint of efficiency. In general-purpose parallel machines, e.g. distributed MIMD machines, the overhead in parallel processing mainly consists of communication cost and synchronization cost between parallel processes, and some of communication cost is caused by synchronization.

Considering the sequential processing of concurrent processes, performance will become lower if many process switches occur. This is because the context switching cost is high with relation to sequential execution cost. However, the execution within a same context is efficient. Most of the synchronization and communication cost in parallel processing is the

same magnitude as such context switching cost. Therefore an ideal parallel machine has low communication cost retaining the efficiency of each parallel process.

There are two primary directions to approach the above ideal architecture. One direction is to make parallel processing granularity small, down to elementary operations. Then synchronization within small operations is treated by special hardware. An example is an instruction-level dataflow machine[31]. However, each elementary operation suffers from synchronization overhead even though such overhead is small. The other direction is to make each granule large. Even though the synchronization cost may become larger than in the above case, there is a possibility to make the best use of the concept of locality. We chose this direction in the design of PIM architecture, The basic unit of parallel processing in PIM is one goal reduction (see Figure 1). Moreover the PIM execution model tries to avoid synchronization overhead by continuously executing several goal reductions.

[2] Locality in architecture *v.s.* Locality in given program

The locality of architecture exists in any kind of hardware configuration. Considering data transfer, the communication cost for register-to-register data transfer within a processor is low. On the other hand, the cost of data transfer from memory to memory is high. As far as such differences exist in hardware configurations, the parallel processing model should make best use of locality at the hardware level. In other words, we have designed the PIM architecture so that hardware locality can be easily used by parallel processing model as in Figure 3.

The execution of a given program can be assumed to be a set of processes. In the parallel processing of PIM, each process consists of a set of goal reductions (See section 3.3 and 3.4). Then some processes communicate with each other very often and some do not. Considering recursively invoked goal reductions, most of goal context or environment is used by the next goal reductions. Thus these recursively invoked goal reductions relate to each other strongly. The locality in programs can be defined as a group of processes with a strong relationship. Therefore, the execution of given programs should be divided into sets of processes or subprocesses by their locality. Then these parts of a program should be mapped the locality of architecture. In order to this, the followings should be considered:

- enhancing the parallelism and locality in given problems by the design of parallel algorithms,
- expressing parallelism and locality in programming,
- providing the dynamic control mechanism for managing processor loads.

The locality in problems can be first specified in the algorithm design and programming. In parallel programming, a given problem can be expressed as parallel processes which communicate with each other[22]. As for programmers, it is important to be able to express in their programs what they want to solve. In addition to this, programmers can be expected to take care of the locality in the program, even though in an abstract level. The parallel algorithms and/or techniques for numeric processing have been studied in the literature[8].

However, parallel algorithms for AI applications have not been studied so much[1]. The PIM development also aims to build a work bench for parallel algorithm research.

[3] Memory Management in PIM

Recently, so-called AI workstations have been developed[13]. They have clarified the importance of memory management functions, such as garbage collection. This is because the actual performance of such AI-oriented systems critically depends on garbage collection performance.

In parallel inference systems, the requirement of efficient memory management functions is also important. First, we found that garbage collection should be done on each processor, because global garbage collection requires enormous data communications in parallel computer systems. Next, on-the-fly garbage collection is necessary because otherwise, if one processor stops execution and starts garbage collection, other processors can seldom communicate with the garbage collecting processor. As the result, such a garbage collecting processor disturbs all other processors. To realize efficient memory management functions, it is necessary to cope with this problem from various points of view.

In the parallel processing of PIM, the multiple reference bit MRB[5] method is used to manage reference pointers in data structures instead of a traditional reference counting method[31]. By using MRB, on-the-fly garbage collection, as well as efficient data structure manipulation, are implemented on PIM.

[4] Operating system for PIM

Operating systems play an important role in all computer systems, however operating systems for parallel computers have not been major research issues[2]. The principal functions of parallel operating systems are almost the same as those of conventional operating systems:

- hardware resource management
- object program and data allocation and their management
- user process management
- input/output.

Each of these includes many difficult problems due to the nature of parallel processing environments. For example, to allocate processes and to balance loads of processor elements are difficult problems. In addition, the loads should be balanced considering the locality in given programs.

The PIM operating system, as well as a high-level system programming language $\mathcal{A}'\mathcal{UM}$ [34], are being developed (See section 3.4). The system programming language is based on a parallel logic programming language object-oriented programming features, as in Vulcan[10]. The operating system is designed as a *self-contained* operating system, e.g. each function in the

[1] See however [21]

[2] See however Logix, a parallel operating system implemented in FCP[23]

operating system is written in the system language. So, the operating system itself is executed as a set of parallel processes which communicate with user processes through logical variables in logic.

2.2 By Logic Programming Framework from Hardware to Users

One of our most important policies in the above research is to build up a total PIM system based on logic programming[6]. AI software can be implemented in logic programming, which may include the basic AI functions such as meta-reasoning, learning and knowledge acquisition[2]. This AI software will act as a high-level human interface in the Fifth Generation Computer. So the logic programming framework is specified as the *kernel language*.

Interesting techniques such as partial evaluation, program transformation, and algorithmic debugging[19] are now being studied in the framework of logic programming. These techniques will be effective in the design of an efficient compiler and the programming support system for PIM. Operating systems and system software will also play an important role to offer a practical PIM system to users. The major roles of such operating systems are to manage parallel processes and resources. The non-side-effect characteristics of logic programming is necessary in describing this software for PIM. The clarity of logic programming also offers PIM architects many benefits in the design of PIM architecture, such as on-the-fly garbage collection design[5].

Finally, by designing both software and hardware architectures based on a logic programming language, namely the kernel language, the system designers of PIM can easily look through all levels of the PIM system in logic programming framework. This is an important method to solve the so-called *semantic gap* argument, i.e. application and implementation are closer, therefore execution is faster.

3 The Kernel Language and its Role

3.1 What is necessary in the Kernel Language Design

This section describes the features that are required in the kernel language (this covers all levels of PIM programming, i.e. from the machine language of PIM to the user language).

First, the kernel language must be a general-purpose language by which programmers can express important concepts in parallel programming, such as inter-process communication and synchronization. The kernel language at the user level must also be a parallel language *by nature*. It must not be similar to a conventional sequential language augmented with constructs for parallelism. This is because the inter-process communication and synchronization should be treated by basic functions. The kernel language should have clear and simple semantics. Finally it must be an efficient language. It means that even small programs such as simple UNIX utilities can be written in the kernel language and executed as efficiently as in a conventional language like C.

In the initial stage(1982–1984), we first studied pure Prolog as a candidate for the PIM kernel language[7]. However, it was difficult to extend pure Prolog to the kernel language sat-

isfying the above functions. Primarily we could not find how to describe the operating systems which manages the overall processing of AI programs. Concurrent Prolog(CP)[20] was studied next because with it the control of concurrent processes could be easily described[25]. However, CP has a complex language specification[28]. Moreover it seemed difficult to implement efficiently. Through the experience of pure Prolog and CP, GHC(Guarded Horn Clauses)[29,30] was born in ICOT. GHC satisfies both simplicity in language semantics and facilitates expression of parallelism. Languages like GHC and CP have been called *AND-parallel Logic Programming Languages*. In these kinds of languages, parallel processes are described as *goals*, and synchronization and/or stream communication are defined by shared logical variables in goals. By the above programming features, we can describe not only AI application programs but also the operating systems which control parallel processes.

3.2 Brief Introduction to GHC

GHC is a logic programming language enabling parallel programming. Clauses in GHC programs are selected in a pattern-driven manner as in Prolog, however unification of logical variables are performed in a single assignment manner. Parallel processing is 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.

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 *the 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. Execution of a GHC program proceeds by reducing a given goal clause to the empty clause under the following rules[3]:

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.

[3]These rules are informal. The formal rules can be found in [29,30].

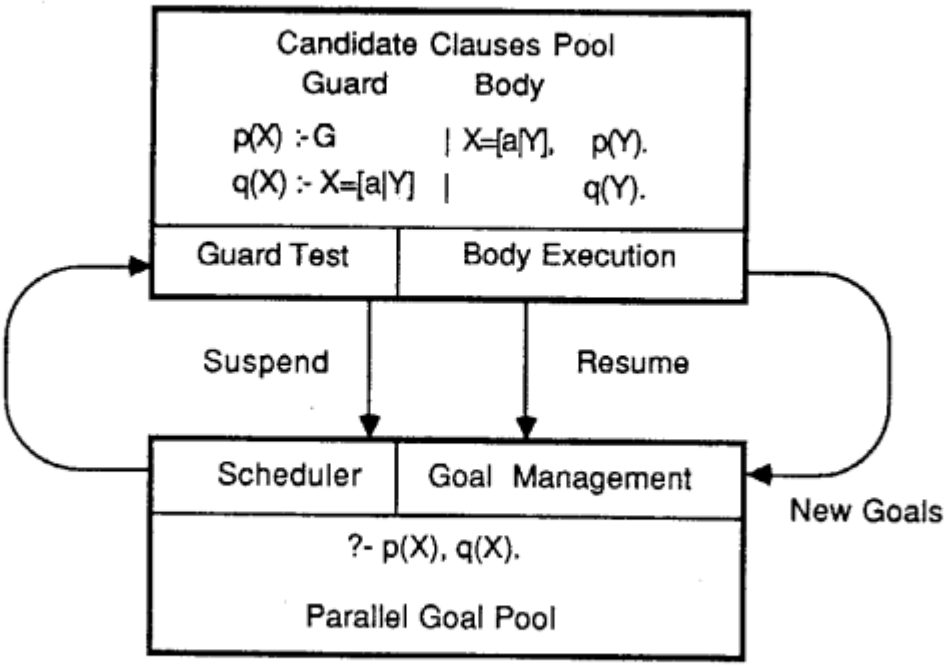


Figure 1: Execution Feature of GHC

3.3 Execution Feature of GHC

It is natural to regard the processing mechanism of GHC as reduction[16]. Figure 1 shows the execution feature of GHC. Assuming that there is a goal clause with two goals[4] $p(X)$ 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*[5]. In the case of Figure 1, the suspended-goal $q(X)$ will be resumed by the execution of $p(X)$.

3.4 The Kernel Language of PIM: Current Status

The kernel language system called *KL1* is now being developed by extending GHC. *KL1* is a hierarchical language system consisting of *KL1-U* , *KL1-C* , and *KL1-B* , as shown in Figure 2.

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

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

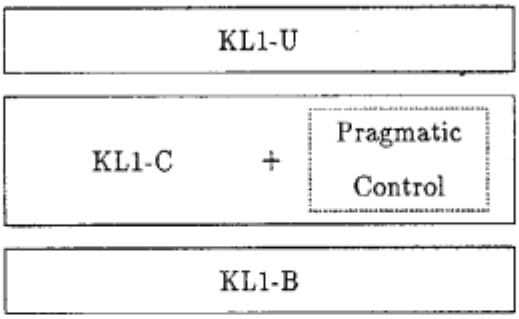


Figure 2: The KL1 Language Systems

KL1-C

KL1-C works as a core in the *KL1* language system. *KL1-C* was initially specified as flat-GHC. Flat-GHC is a subset of GHC, whose guard goals are all built-in predicates. This restriction makes the language implementation more efficient while keeping most of GHC's descriptive power. Starting from flat-GHC, *KL1-C* has been extended, accepting the requirements from the user language *KL1-U* , the machine language *KL1-B* , and the PIM operating system *PIMOS* . These extensions are as follows.

First, *KL1-C* has meta-logical functions defined in several built-in predicates. Meta-logical functions, in general, enable *KL1* programmers to handle the logical values of goals. In the *KL1* system, meta-logical functions are essential for the process resource management in *PIMOS* .

Next, the user language *KL1-U* is designed based on parallel objects[10,22]. Each object communicates with other objects by sending and receiving messages. Thus, the message merge operation is used most frequently. *KL1-C* has built-in predicates to increase the performance of message merge operations. Although these predicates are implemented by the machine language *KL1-B* , their semantics can be described in flat-GHC. In addition, *KL1-C* has the built-in predicates for array operations implemented by *KL1-B* in the same way as the merge operations.

The *KL1* system processes extra-logical parameters which describe the control of parallel goals in *PIMOS* , and which specify goal scheduling in application programs. For example, goal scheduling priority can be specified in each *KL1* goal. By using this priority, the operating system *PIMOS* can be executed in higher priority than user programs. These extra-logical parameter values may depend the lower-level hardware construction such as the network topology of PIM. Although these functions are not necessary for *KL1* logical program semantics, they are important for the actual PIM system with *PIMOS* . In the *KL1* system, such extra-logical features are called as *pragma*.

KL1-U

High-level user languages implemented in *KL1* are called *KL1-U* . Currently a system programming language $\mathcal{A}'\mathcal{UM}$ [34] is being designed for *PIMOS* programmers as well as for large application programs. The main features of $\mathcal{A}'\mathcal{UM}$ are its high-level abstractions based on pure parallel objects. In addition, $\mathcal{A}'\mathcal{UM}$ is characterized by its stream merging, name abstraction, macro expansion, and modular programming support by class inheritance and method wrapping. A prototype compiler from $\mathcal{A}'\mathcal{UM}$ to *KL1-C* will be available soon.

KL1-B

KL1-B [11] is a virtual machine language interfacing the PIM hardware and *KL1* just as WAM[32] interfaces Prolog. *KL1-B* can be regarded as a compiler target language of *KL1-U* and *KL1-C* . *KL1-B* also includes some special functions to directly control and maintain the PIM hardware.

To implement on-the-fly garbage collection, the multiple reference bit MRB[5] is maintained in each *KL1-B* instruction. In addition, several garbage collection instructions are implemented in *KL1-B* . The compiler analyzes and detects the chance when garbage cells can be collected. The compiler then generates object programs which include garbage collection instructions. The MRB is also used to implement the efficient merge and array operations mentioned previously.

Currently the first version of *KL1-B* [11] has been specified, and a prototype compiler from the subset of *KL1-C* programs and emulators on conventional machines are available. In an actual programming environment, most system programmers are expected to develop using *KL1-U* , so the compiler from *KL1-U* ($\mathcal{A}'\mathcal{UM}$) to *KL1-B* is very important. We are now extending and tuning up the *KL1-B* specification. Then, the compiler from $\mathcal{A}'\mathcal{UM}$ to *KL1-B* will be designed based on the prototype compilers.

4 PIMOS and Multi-PSI Systems

4.1 Parallel Operating System: PIMOS

PIM operating system called *PIMOS* is being designed, aiming for implementation on *Multi-PSI v.2* [26] and on *PIM-1* (see 5.1). The basic policies in the design and development of *PIMOS* are as follows. *PIMOS* is written completely in *KL1* , without side-effects. Even the hardware interruption signals will be handled as messages in *KL1* streams. *PIMOS* is designed as a single operating system to be executed in parallel. It is not an aggregate of individual operating systems such as network operating systems. From a users point of view, the parallel inference systems will be seen as a single system even if it consists of many hardware processors. In addition, *PIMOS* should be suitable for practical use in parallel algorithm research. *PIMOS* is developed not only for an experimental system but also for a research tool to the final stage of FGCS project.

Currently the *PIMOS* development support system, PDSS, is implemented on UNIX machines. The PDSS system is a sequential *KL1-B* emulator with a simple user interface, on

which *PIMOS* functions are tested. Then the *PIMOS* development on an actual parallel system, *Multi-PSI v.2* , will start in the beginning of next year.

4.2 Multi-PSI Systems

Workbenches for studying parallel software systems for PIM are developed. *Multi-PSI v.2* is designed based on the experience of *Multi-PSI v.1* [7], six loosely coupled *PSI-I* machines. *Multi-PSI v.2* includes up to 64 processing elements based on *PSI-II* [15]. These processors are connected by a two-dimensional mesh-like packet switching network. *KL1-B* is interpreted in microcode on each processing element.

The following issues have been studied for the parallel implementation of *KL1* :

- how to locally collect garbage cells on each processor (global garbage collection on a loosely coupled multiprocessor is very costly)
- how to distribute processing load(i.e. *KL1 goals*), and how to detect the end of execution[9],
- how to decrease the repeated copying of large data structures,
- how to perform inter-processor unification and/or stream communication
- how to interface with *PIMOS* written in *KL1* framework.

The first *Multi-PSI v.2* hardware will be available at the end of 1987, and the second one will be in the middle of 1988. The *KL1* system will be implemented on *Multi-PSI v.2* , followed by *PIMOS* implementation. To enable many researcher to join *PIMOS* development, a simulator of *Multi-PSI v.2* called *Pseudo Multi-PSI* is also being developed on *PSI-II* . About 100 *PSI-II*s are used in whole ICOT project. Pseudo multi-PSI will be widely used not only for *PIMOS* development but also for parallel applications.

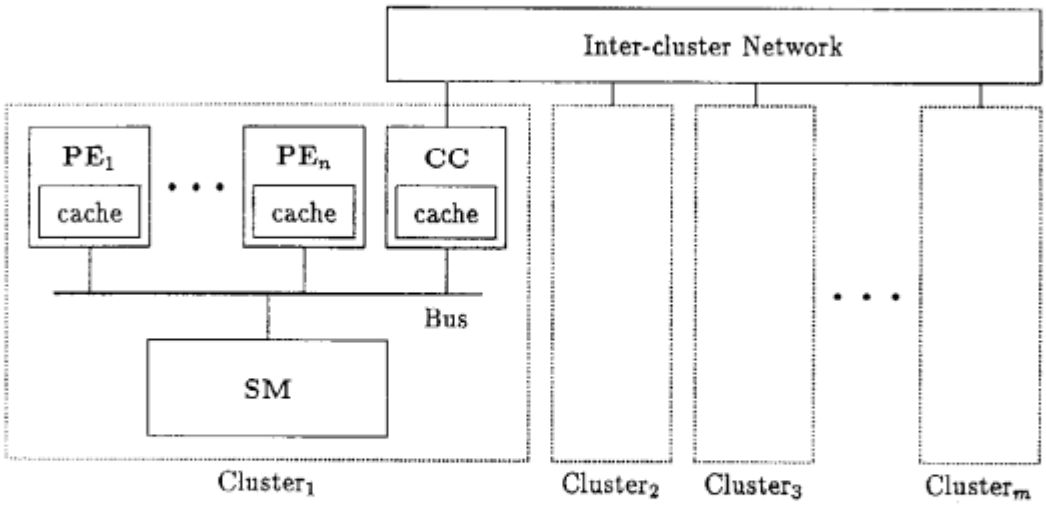
5 *PIM-1*: Target Pilot Machine in Intermediate Stage

5.1 Overview of *PIM-1*

PIM-1 , the fist PIM to be developed, is expected to have 100 processing elements. The target processor performance is 200 to 500 KRPS[6] for *KL1* , so that 10 to 20 MRPS is expected as the total performance for applications within actual environments. *PIM-1* has a hierarchical structure with a cluster concept (Figure 3). Each cluster consists of eight or more processors (PE) which communicate through shared memory (SM) over a common bus. The processor element (PE) is now being designed with a tag architecture and the hardware instruction set optimized to *KL1-B* . PEs within each cluster share one address space. Therefore each PE can speedily communicate by reading/writing shared memory.

Focusing on *KL1* parallel execution in each cluster, quick and exclusive accesses to shared data are the key issue. Thus, cache memories are important elements in providing quick

[6]RPS: *KL1* goal reduction per second



PE : Processor Element

SM : Shared Memory

CC : Cluster Controller

Figure 3: *PIM-1* Overview

data access. Several cache protocols have been proposed so far, each of which aims to solve the so-called *cache coherence* problem[1]. Each PE in *PIM-1* has a coherent cache memory designed specifically for *KL1* execution[12]. These cache memories increase the efficiency of local execution. In addition, they offer a high speed communication path within the cluster. On the other hand, it is necessary to provide an efficient mechanism to exclusively access shared memory. In a *PIM-1* cluster, exclusive memory accesses can be obtained at small cost by using the cache block status of coherent cache memory[12].

The clusters are connected by a switching network. Because each cluster has its own address space, inter-cluster parallel processing is performed by communicating message packets with address transformation. A cluster controller on each cluster manages the message communications between clusters.

With the above hardware configuration, the *PIM-1* architecture will offer several kinds of hardware locality, namely local execution on each PE, parallel execution within each cluster, and inter-cluster parallel execution.

5.2 Parallel processing feature

Two kinds of *KL1* parallel execution models have been studied for *PIM-1* : the message-oriented model[9,26] for inter-cluster parallel execution and the shared heap model[17] for the tightly-coupled multiple processors in each *PIM-1* cluster. Both execution models are introduced in the *PIM-1* global architecture.

Shared heap model

The following data/control structures are used in KL1 goal reduction. Parallel *KL1* goals are represented by goal-records and their environments. Goal-records includes atomic goal arguments or pointers to their environments consisting of logical variable cells or structures. The reducible goal-records are stored as a ready-queue. Clauses in *KL1* programs are compiled into *KL1-B* . Each processor dequeues a goal-record from a ready queue, then performs a goal reduction by executing *KL1-B* instructions, accessing to the goal environment. Some goals are waiting for the instantiated values of variable cells in order to synchronize with other parallel goals. Such goal-records are bind-hooked with the variable cells by suspension-records. The shoen-records form a tree-like structure, whose leaves are the goal records, to manage their logical results (success/failure) and process resources as mentioned later.

In the *PIM-1* cluster, a *KL1* program is executed using the following shared heap model[17]. Even though PEs can access physical shared memory within a cluster, the parallel execution mechanism should use local data/control structures as much as possible. Therefore, *KL1* goals and control structures are examined first to determine they can be treated as local data structures. We decided to use a local ready queue on each processor to store goal-records. On the other hand, goal environments, suspension-records and shoen-records are stored in shared heap. Then, an exclusive memory access is used when a processor instantiates an unbound logical variable. However, it is not necessary to access exclusively when each processor allocates a new data structure even though they are shared between processors. This is because each processor has its own free memory area

The parallel processed granules are *KL1* goal reductions in *PIM-1* . However, several repeated goal reductions have strong dependency. Therefore such goal reductions should be performed within one PE as a sequence, e.g. by depth-first scheduling. The local ready queue on each processor also enables the depth-first scheduling. On the other hand, processing loads are balanced by distributing goal-records. The goal distribution is initiated by an idle processor. The idle processor sets a global flag to request a goal distribution. to other busy processors. Then a busy processor, which first finds the flag, sends a goal-record to the idle processor.

A software simulator on the sequential machine was developed to evaluate the shared heap model[17]. A parallel emulator is also implemented on an actual multiprocessor, Balance 21000[18]. The shared heap model is being evaluated in detail on both emulators.

Message oriented model

Inter-cluster parallel processing is performed by the message oriented model, which is designed on *Multi-PSI v.2* . Each cluster has its own address space. Therefore, one processor in *Multi-PSI v.2* corresponds to one cluster in *PIM-1* .

When one processor tries to access a variable shared by another goal in a different cluster, a message to ask its value is sent between clusters through the inter-cluster network. Because each cluster has its own address space, it is necessary to provide address transformation tables to manage outgoing and incoming pointers on each cluster. By using such tables, it becomes

possible to implement dynamic memory management functions.

Parallel goal management

Both in intra-cluster and inter-cluster parallel processing, it is important to manage parallel goals which are distributed and then terminated dynamically. In addition, *PIMOS* tries to avoid resource exhaustion by never-ending goal reductions or illegal goal reductions. Therefore the parallel execution mechanism in *PIM-1* has the following functions.

A tree-like data structure is introduced to manage parallel goals[9]. The nodes of this tree structure are *shoen*[7] or *sato-oya*[8] nodes. Every group of goals, corresponding to user jobs or sub-jobs, belongs to each shoen node. Thus, a shoen node may have lower level shoen nodes. On the other hand, currently used processing resources are measured by both the number of goal reductions and the amount of memory usage. Then the used computing resources of user jobs or sub-jobs are managed by the shoen nodes. The sato-oya nodes are used to handle distributed goals over clusters which belong to one shoen node. In addition, shoen and sato-oya nodes have special control streams to inform exceptional signals, such as a goal reduction failure, to upper-level shoen nodes. The goals are scheduled by priority so that *PIMOS* goals can be invoked as soon as special events occur. This priority scheduling can also be used within user programs.

6 Conclusion

Parallel processing researchers have a great interest in the essence of computing which is revealed in the research process of parallel processing. This report gives a research and development overview and the current research status of parallel inference machines in ICOT. To extend the AI application field, more general and more powerful computers, such as the PIM are needed. The logic programming framework plays an important role in the PIM research. The research issues in both parallel hardware mechanisms and parallel software systems are studied based on the logic programming framework. Efforts to integrate them into a total system are essential in the PIM research.

Acknowledgement

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. I wish to thank to Dr. Evan Tick for his fruitful comments. I also wish to thank to ICOT Director Dr. Kazuhiro Fuchi and the chief of the fourth research section Dr. Shunich Uchida for valuable suggestions and guidance.

[7] *Shoen* is a Japanese word which means a minor. In [9], a *shoen* node is called a metacall node

[8] *Sato-oya* is a Japanese word which means a foster parent

References

[1] P. Bitar and A. M. Despain. Multiprocessor cache synchronization. In *Proc. of the 13th Annual International Symposium on Computer Architecture*, June 1986.

[2] K.A. Bowen. Meta-level programming and knowledge representation. *New Generation Computing, OHMSHA Ltd. and Springer-Verlag*, 3(4):359–384, 1985.

[3] G. Broomell and J.R. Heath. Classification categories and historical development of circuit switching topologies. *ACM Computing Surveys*, 15(2):95–133, 1983.

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

[5] T. Chikayama and Y. Kimura. Multiple reference management in flat ghc. In *Proceedings of the Fourth International Conference on Logic Programming*, 1987. Also in ICOT Technical Report, TR-248.

[6] K. Fuchi and K. Furukawa. The role of logic programming in the fifth generation computer project. *New Generation Computing, OHMSHA Ltd. and Springer-Verlag*, 1(5):3–28, 1987.

[7] A. Goto and S. Uchida. *Toward a High Performance Parallel Inference Machine –The Intermediate Stage Plan of PIM–.* TR 201, ICOT, 1986. Also in Future Parallel Computers, LNCS 272, Springer-Verlag.

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

[9] N. Ichiyoshi, T. Miyazaki, and K. Taki. *A Flat GHC Implementation on the Multi-PSI.* Technical Report, ICOT, 1986. To appear as ICOT Technical Report.

[10] K. Kahn and et al. *Vulcan: Logical Concurrent Objects.* Technical Report, Xerox Palo Alto Research Center, 1986.

[11] Y. Kimura and T. Chikayama. An abstract KL1 machine and its instruction set. In *Proceedings of the 1987 Symposium on Logic Programming*, 1987. Also in ICOT Technical Report TR-246.

[12] A. Matsumoto and et.al. *Locally parallel cache optimized for KL1 execution.* TR, ICOT, 1987. Also submitted for ISCA 1988.

[13] D.A. Moon. Architecture of the Symbolics 3600. In *Proceedings of the 12th Symposium of Computer Architecture*, 1985.

[14] 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.

[15] K. Nakashima and H. Nakajima. Hardware architecture of the sequential inference machine: PSI-II. In *Proceedings of 1987 Symposium on Logic Programming*, pages 104–113, San Francisco, 1987.

[16] J.A. Robinson. A machine-oriented logic based on resolution principle. *Journal of ACM*, 12(1):23–41, 1965.

[17] M. Sato, A. Goto, and et al. Kl1 Execution Model for PIM Cluster with Shared Memory. In *Proceedings of the Fourth International Conference on Logic Programming*, 1987. Also in ICOT Technical Report.

[18] Inc. Sequent Computer Systems. *Balance 8000/21000 Technical Summary.*

[19] E.Y. Shapiro. *Algorithmic Program Debugging.* MIT Press, 1983.

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

[21] E.Y. Shapiro. Systoric programming: a paradigm of parallel processing. In *Proceedings of the Internaltional Conference on Fifthe Generation Computer Sytems*, pages 458–470, 1984.

[22] E.Y. Shapiro and A. Takeuchi. Object oriented programming in Concurrent Prolog. *New Generation Computing, OHMSHA Ltd. and Springer-Verlag*, 1(1):25–48, 1983.

[23] W. Silverman and et al. *The Logix Sytem User Manual.* Technical Report CS-21, Weizmann Institute of Science, 1986.

[24] 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.

[25] A. Takeuchi and K. Furukawa. Bounded buffer communication in Concurrent Prolog. *New Generation Computing, OHMSHA Ltd. and Springer-Verlag*, 3(4):359–384, 1985.

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

[27] 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.

[28] K. Ueda. *Concurrent Prolog Re-Examined.* TR 102, ICOT, 1985.

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

[30] K. Ueda. *Guarded Horn Clauses: A Parallel Logic Programming Language with the concept of a Guard.* TR 208, ICOT, 1986. (Also to appear in Programming of Future Generation Computers, North-Holland, Amsterdam, 1987.).

[31] A.H. Veen. Dataflow machine architecture. *ACM Computing Surveys*, 18(4):365–396, December 1986.

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

[33] 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.

[34] K. Yoshida and T. Chikayama. *A'UM – Parallel Object-Oriented Language upon KL1 –.* TR 308, ICOT, 1987.