ICOT Technical Report: TR-659

TR-659

PIM Architecture and Implementations

by
A. Imai, K. Hirata & K. Taki

June, 1991

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

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Fourth Franco Japansese Symposium
Rennes, France, July, 1991

PIM Architecture and Implementations

Akira Imai * Keiji Hirata Kazuo Taki
Institute for New Generation Computer Technology (ICOT)

Abstract

In the Japanese Fifth Generation Computer Systems (FGCS) project, parallel inference machine (PIM) systems are being developed. Multi-PSI system has already been developed as an experimental hardware systems to provide a practical tool for parallel software research. Several PIMs are now under development as prototype hardware systems of Fifth Generation Computer System.

This paper describes a brief summary of (1) the architecture of Multi-PSI and PIM, and (2) KL1 language implementation on these hardwares.

1 Introduction

Research and development (R & D) of the parallel inference machine (*PIM*) [4] system is one of the most important issues in the Fifth Generation Computer Systems (FGCS) project. The R & D includes (1) hardware architecture, (2) implementation of kernel language (KL1), and (3) the PIM operating system (PIMOS)[2] to build up a total system for knowledge information processing systems.

KL1 [27], the kernel language of PIM is a concurrent logic programming language [22] based on flat-GHC [26]. Since KL1 uses clear and simple semantics as a concurrent programming language, it is very easy for programmers who write large applications to express concurrency control such as sychronization and communication.

The *Multi-PSI* system [25] was built to provide a practical tool for parallel software research and to enhance the research for KL1 parallel implementation. This has been available to parallel software researchers for over two years. We have used the system to study several concurrent programming paradigms such as dynamic load balancing schemes [3] and how to build large scale applications [14].

Five PIM systems are now under development as prototype hardware systems of FGCS. The purpose of the development of more than one PIM was to examine and compare the technical issues for different architectures.

The hierarchy of the PIM system is summarized in Figure 1. The lowest layer is hardware. The second layer is a KL1 runtime system which executes KL1 programs by accessing hardware resources.

*Mita Kokusai Bldg. 21F, 4-28, Mita 1-Chome, Minato-ku, Tokyo 108, Japan E-mail: imai@icot.or.jp

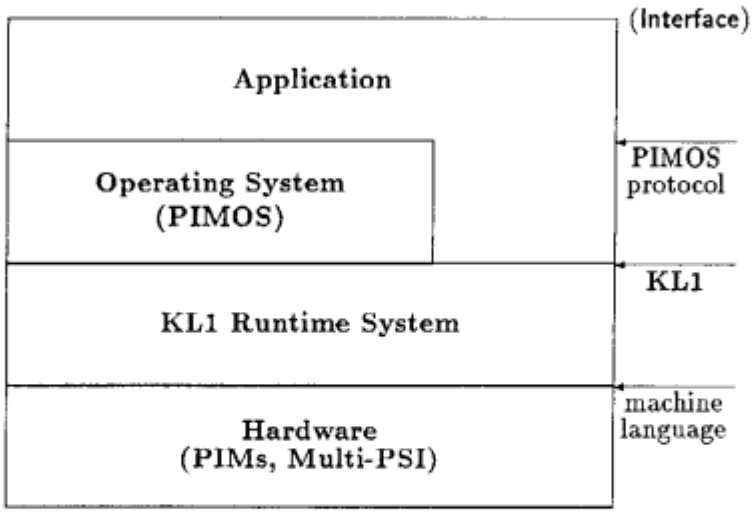

Figure 1: Hierarchy of PIM system

This level includes (1) memory management, (2) process control and (3) distributed variable coherence control functions as OS kernel. The third layer is an operating system PIMOS which controls user tasks and file systems. This layer includes programming systems such as debuggers and compilers, which are all written in KL1. The upper-most layer is a parallel application program written in KL1.

This paper focuses on the lower two layers, picking up several technical issues. This paper is organized as follows. Section 2 reviews the Multi-PSI hardware system and KL1 implementation on it. Section 3 explains PIM architectures and KL1 implementation on them. Section 4 describes the current status of PIM R & D and future plans.

2 Multi-PSI : Experimental Hardware System

2.1 Architecture

2.1.1 Overview

The *Multi-PSI* [25] is a MIMD (multiple instruction-stream, multiple data-streams) type non-shared memory multi-processor consisting of up to 64 processing elements (PEs).

To build a large scale multi-processor architecture, a distributed memory architecture (c.f. shared memory architecture) is indispensable, however it is said that programming on a distributed memory system is more difficult than on a shared memory system. One approach to deal with this problem is to build physically distributed, logically shared memory architectures such as DASH architecture [12]. Our approach looks like this approach from the view of a KL1 programmer, however it is different from the view point of architectural design. Global consistency in the logical variables among PEs are maintained by KL1 runtime systems sending/receiving unification messages on demand. The reason why we did not choose a DASH like approach is that KL1 processes of fine-grain communicate so

frequently using shared variables that the logical shared memory based on large memory blocks does not work effectively.

The PEs of Multi-PSI, which are the CPUs of personal sequential inference machines (PSI-II) [17], are connected by 8×8 mesh networks. Since two-dimensional mesh network topology is scalable, the Multi-PSI architecture can be extended to larger-scale configurations with little modification.

2.1.2 Processing Element

The processing element (PE) is a 40-bit (8 bits for tag, 32 bits for data) CISC processor controlled by the horizontal micro-instruction. Its micro-programmability enables flexible implementation for incrementally enhancing the performance and adding various functions. It has up to 16M words (5 bytes/word) of memory and a 4-Kword direct-map cache memory.

The performance of Multi-PSI PE is 130 KRPS (Kilo Reductions Per Second) for the append program including incremental garbage collection.

2.1.3 Network

Each processing element has a specially designed network-controller to send/receive messages to/from other PEs.

The network-controller has an automatic routing facility (the worm-hole routing). It can route messages according to the destination PE number in the message header without disturbing the CPU. The bandwidth of each channel is 5 Mbyte/sec.

2.2 KL1 Implementation for Distributed Memory Environment

KL1 implementation techniques for distributed memory multi-processors have been developed on Multi-PSI [15]. KL1 is compiled to KL1-B [11] before execution, which is just like the WAM code of Prolog, and KL1 runtime system interprets KL1-B code. The KL1 runtime system includes the following facilities, which are managed by conventional operating systems.

Memory management

The storage allocation/reclamation system is maintained by the KL1 runtime system because it is essential to free programmers to perform memory management when writing large scale applications by symbolic operation.

Process control & resource management

The meta-programming facility by 'Sho-en' [2] enables a operating system to be written in KL1 because it can encapsulate user program failures in 'Sho-en', and protect the operating system from it. 'Sho-en' functions are supported by the KL1 runtime system.

Goal scheduling

Goal scheduling and synchronization are major issues of parallel programming systems. Synchronization among goals, that is suspension/resumption of KL1 goals, is managed by the KL1

runtime system automatically even if goals communicating each other are on different PEs. However, programmers are currently responsible for goal distribution among PEs because communication cost among PEs are so high that automatic goal distribution does not work well.

Exception handling

As we described before, internal exceptions such as program failure can be encapsulated by 'Shoen'. External exception handling such as I/O to FEP (Front End Processor) or disk is handled by special built-in predicates.

Because logic programming languages are based on dynamic structure allocation, a garbage collection (GC) mechanism is crucial for reclaiming storage during computation. Some novel methods of efficient GC have been developed and implemented on the KL1 runtime system for Multi-PSI. An incremental GC scheme using a single bit (MRB-GC) was proposed [1], and evaluated intra processor on Multi-PSI [9]. A GC for inter-processor pointers (WEC) was also proposed [7] and implemented on Multi-PSI.

Other techniques for KL1 implementation on distributed memory multi-processors are as follows:

- Process control facility using the WTC scheme [19];
- Distributed unification scheme [7];
- Debugging support facility such as detecting perpetual suspension goals [10].

These were implemented and evaluated on Multi-PSI [16].

3 PIM: Fifth Generation Computer Prototype System

3.1 Architecture

3.1.1 Overview

As we described in section 1, we are manufacturing several PIMs: *PIM/p*, *PIM/c*, *PIM/m*, *PIM/i*, and *PIM/k*.

All PIMs except PIM/m, which is designed to be a large scale multi-processor where the locality in communication cost can easily be used from software, have hierarchical architectures. That is they are *clusters* formed by 10 or so PEs connected by shared memory and shared bus with coherent cache [24], and the clusters are connected by asynchronous network. The global configuration of five PIMs is summarized in table 3.

An Overview of PIM/p architecture is shown in Figure 2. PIM/p consists of up to 512 PEs. 8 PEs inside a cluster share one address space, and data coherence is maintained by a snooping cache mechanism. Each PE has a network interface unit to provide sufficient performance for short and long message packets.

Table 1: Global Configuration

	Topology	Number of Clusters	Total number of PEs
PIM/p	hypercube	64	512
PIM/c	crossbar	32	256
PIM/m	mesh	256	256
PIM/i	—	2	16
PIM/k	—	4 †	16
Multi-PSI	mesh	64	64

† : mini-clusters

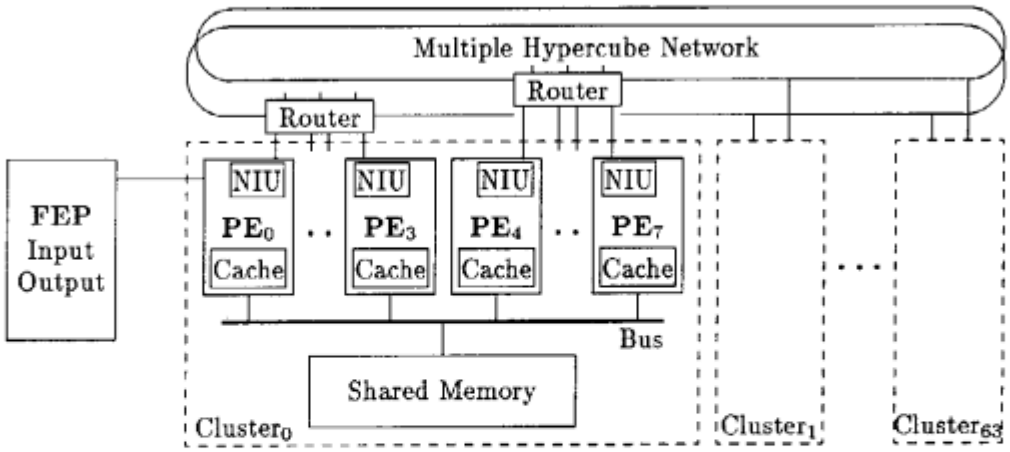

Figure 2: Overview of PIM/p Architecture

Table 2: Specification on Processing Element

	Instruction set	Cycle time	LSI fabrication	Line interval
PIM/p	RISC + macro instruction	60 nsec	standard-cell	0.96 μm
PIM/c	CISC (micro programmable)	50 nsec	gate-arrays	0.8 μm
PIM/m	CISC (micro programmable)	65 nsec	standard-cell	0.8 μm
PIM/i	RISC	100 nsec	standard-cell	1.2 μm
PIM/k	RISC	100 nsec	custom	1.2 μm
Multi-PSI	CISC (micro programmable)	200 nsec	gate-arrays	2.0 μm

3.1.2 Processing Element

Since KL1 implementation requires frequent runtime type checking, all CPUs of PIM are designed as the tagged-architecture following the Multi-PSI.

PIM/p, PIM/i and PIM/k have RISC-like instruction set whereas PIM/m and PIM/c have CISC-like micro programmable instruction set (Table 2). The former processors execute machine instructions, which are at a level still lower than KL1-B, the latter processors interpret KL1-B code by horizontal micro program.

The CPU of PIM/p [6] has a unique feature called *macro-call* [23] instructions for light-weight subroutine calls. The instructions enable the size of compiled user program codes to be kept small and to reduce the overheads of subroutine calls. It also has some more instructions dedicated to KL1 implementation, such as dereference instructions and MRB [1] incremental garbage collection instructions.

The estimated performance of one processing element of PIM/p is 600 KRPS, which is over 4 times as fast as that of Multi-PSI.

3.1.3 Network

Network connecting clusters are summarized in table 3.

In PIM/p, four NIs are connected to a router, which works as a node in a global network. PIM/p also has two SCSI channels per cluster, which are used for connecting PIM/p and FEP (Front End Processor) or PIM/p and disks.

PIM/c has one special processor called cluster controller per cluster, which is connected to a shared bus. The cluster controller has overall responsibility for network communication.

3.1.4 Cache System

Since we had studied that it was effective for KL1 execution to exploit data access localities, we designed coherent cache protocols [5][13] which can keep the locality high and reduce the shared bus

Table 3: Network

	# PEs in a cluster	# NIs in a cluster	Comment
PIM/p	8	8	each PE has NI
PIM/c	8	1	NI is connected to a bus
PIM/m	1	1	
PIM/i	8	1	NI is connected to a bus
PIM/k	16	—	
Multi-PSI	1	1	

(PE = processing element, NI = network interface)

Table 4: Specification on Cache

	Coherence Control		Mapping	Cache Size	
	Protocol	# States †		Instruction	Data
PIM/p	invalidation	4	4 way	64 KB	
PIM/c	invalidation	5	2 way	80 KB	
PIM/m	—	—	direct-map	5 KB	20 KB
PIM/i	broadcasting	6	direct-map	160 KB	160 KB
PIM/k	hierarchical invalidation	4	(1st) direct-map	128 KB	256 KB
			(2nd) 4 way	1 MB	4 MB
Multi-PSI	—	—	direct-map	20 KB	

(† does not include *locking* state.)

traffic [18]. All PIMs have write-back type coherent cache protocols (Table 4). Low cost locking mechanisms are also supported by using the cache block status of cache memory.

PIM/k has hierarchical cache system. Four PEs connected to a bus (cache bus) form a mini-cluster, and four mini-clusters are connected to another bus (memory bus).

3.2 KL1 Implementation on Shared Memory Environment

KL1 implementation techniques of PIM's inter-cluster follow in that of Multi-PSI's inter-processor. The biggest difference is inside the cluster (shared memory environment) where all PEs share address space. The techniques include automatic load distribution among PEs, unification with mutual exclusion, and executing garbage collection in parallel.

Since communication costs inside clusters is low, the KL1 runtime system can distribute goals

automatically. Each processor has a local run queue called a "ready goal stack" to keep memory access locality high, and automatic load distribution is performed in the following sequence [20];

1. An idle PE_i broadcasts a signal to all processors.
2. PE_j, which receives the signal first, pops one goal from its ready goal stack.
3. PE_j links the goal to the mailbox of PE_i.

The scheme was evaluated on a parallel simulator on Sequent Symmetry [21].

Executing stopping GC (clusterwise) in parallel combined with MRB-GC was invented and evaluated on a PIM simulator [8].

4 Current Status and Future Work

The five models of PIM hardware are now under production. The development of KL1 implementation for PIM is almost complete as a common specification. Now we are modifying the common specification to fit the respective PIM hardware.

We hope we will be able to demonstrate large applications running on PIMs at the international conference on FGCS to be held in July, 1992.

Acknowledgment

We would like to thank the researchers of ICOT and cooperative companies who have been working on PIMs and Multi-PSI.

References

[1] T. Chikayama and Y. Kimura. "Multiple Reference Management in Flat GHC", In *Proceedings of Fourth International Conference on Logic Programming*, pages 276–293, University of Melbourne, MIT Press, 1987.

[2] T. Chikayama, H. Sato and T. Miyazaki. "Overview of the Parallel Inference Machine Operating System (PIMOS)", In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 230–251, Tokyo, Japan, 1988.

[3] M. Furuichi, N. Ichiyoshi and K. Taki. "A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI", In *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 50–59, Seattle, USA, 1990.

[4] A. Goto, M. Sato, K. Nakajima, K. Taki and A. Matsumoto. "Overview of the Parallel Inference Machine Architecture (PIM)", In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 208–229, Tokyo, Japan, 1988.

[5] A. Goto, A. Matsumoto and E. Tick. "Design and Performance of a Coherent Cache for Parallel Logic Programming Architectures", In *Proceedings of 16th Annual International Symposium on Computer Architecture*, pages 25 – 33, Jerusalem, Israel, 1989.

[6] A. Goto, T. Shinogi, T. Chikayama, K. Kumon and A. Hattori. "Processor Element Architecture for a Parallel Inference Machine, PIM/p", Journal of Information Processing Vol.13, No.2, pages 174–182, 1990.

[7] N. Ichiyoshi, K. Rokusawa, K. Nakajima and Y. Inamura. "A New External Reference Management and Distributed Unification for KL1", In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 904 – 913, Tokyo, Japan, 1988.

[8] A. Imai and E. Tick. "Evaluation of Parallel Copying Garbage Collection on a Shared-Memory Multiprocessor", ICOT Technical Report 650, 1991.

[9] Y. Inamura, N. Ichiyoshi, K. Rokusawa and K. Nakajima. "Optimization Techniques Using the MRB and Their Evaluation on the Multi-PSI/V2", In *Proceedings of North American Conference on Logic Programming 1989*, pages 907–921, Cleveland, MIT Press, 1989.

[10] Y. Inamura and S. Onishi. "A Detection Algorithm of Perpetual Suspension in KL1", In *Proceedings of the Seventh International Conference on Logic Programming*, pages 18–30, Jerusalem, MIT Press, 1990.

[11] Y. Kimura and T. Chikayama. "An Abstract KL1 Machine and Its Instruction Set", In *Proceedings of 1987 Symposium on Logic Programming*, pages 468–477, San Francisco, IEEE Computer Society, 1987.

[12] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta and J. Hennessy. "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor", In *Proceedings of 17th Annual International Symposium on Computer Architecture*, pages 148–159. Seattle, IEEE Computer Society, 1990.

[13] A. Matsumoto, T. Nakagawa, M. Sato, K. Nishida and A. Goto. "Locally Parallel Cache Design Based on KL1 Memory Access Characteristics", ICOT Technical Report 327, 1987.

[14] Y. Matsumoto and K. Taki. "Parallel Logic Level Simulation System on a Distributed Memory Machine", In *Proceedings of Fourth Franco-Japanese Symposium on Artificial Intelligence and Informatics*, Rennes, France, 1991.

[15] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa and T. Chikayama. "Distributed Implementation of KL1 on the Multi-PSI/V2", In *Proceedings of the Sixth International Conference on Logic Programming*, pages 436–451, Lisbon, MIT Press, 1989.