ICOT Technical Report: TR-460

TR-460

The FGCS Computing Architecture

by
K. Taki

March, 1989

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

THE FGCS COMPUTING ARCHITECTURE

Kazuo Taki

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

ABSTRACT

In the fifth generation computer systems (FGCS) project of Japan, *logic programming* and *parallel processing* are adopted as the principles of both software and hardware system development. Their amalgamation, *parallel inference systems*, is being investigated at ICOT.

Both hardware and software development have been carried out based on the kernel language (KL1) as the software-hardware interface. KL1 is a concurrent logic programming language extended from flat GHC. The hardware development target is a parallel inference machine (PIM) with about 10^3 element processors. A smaller scale prototype machine, PIM/p, with about 10^2 processors is under development. The processors are dedicated for efficient execution of KL1 programs on a distributed implementation of the KL1 language processor.

One major software project is the development of a common operating system for all the parallel inference machines, named PIMOS (parallel inference machine operating system), which prototype has been working. Several program development environments have also been prepared. The Multi-PSI, another inference machine prototype, is one such powerful environment, which connects up to 64 personal sequential inference machines (PSI-II) relatively loosely. Further improvements of the PIMOS, and research on various concurrent algorithms and load distribution schemes are in progress on the Multi-PSI.

1 INTRODUCTION

The Japanese fifth generation computer systems (FGCS) project aims at building a prototype of a high performance knowledge information processing system (KIPS). The project spans ten years, from April 1982 to 1992. One of the principal functions of KIPS is its highly parallel inference feature. The target of the hardware system is a highly parallel inference machine (PIM) with about 10^3 processing elements and with an inference speed of more than 10^8 LIPS (logical inferences per second).

All R&D around the PIM (Uchida et al. 1988) has been based on a concurrent logic programming language called KL1 (Chikayama et al. 1988), which is an extension of Flat GHC (Ueda 1986). Machine hardware and the language processor have been carefully designed for efficient parallel execution of KL1. The parallel inference machine operating system (PIMOS) (Chikayama et al. 1988) and application programs (mainly knowledge processing systems) are and will be also written in KL1 or in its extensions. Various software technologies for highly parallel processing, such as highly concurrent algorithms, programming paradigms for practical concurrent programs, and experiments on load balancing schemes, will be studied and developed based on the KL1 language.

Everything, from language to the hardware system, operating system, concurrent algorithms, and programming style is completely new in our R&D approach. They are designed for highly parallel processing on large-scale network-connected MIMD-type multiprocessor systems. The program development environment is very important to cultivate these new computing technologies. Bootstrapping of the R&D, from a simple and small prototype to a large and complex target, is also important. The Multi-PSI was developed as an R&D tool for concurrent software technologies, and also as an early prototype of the parallel inference machines (Taki 1986, Taki 1987).

This paper reports the outline of our FGCS computing architecture by looking at the hardware system, language, operating system and program development, and also reports their progress.

2 LANGUAGES

KL1 is the common kernel language for parallel inference systems in the FGCS project, based on Flat GHC (Ueda 1986). GHC is a concurrent logic programming language similar to Concurrent Prolog (Shapiro 1983) and Parlog (Clark and Gregory 1984).

The advantage of using a concurrent logic programming language is in its implicit concurrency and synchronization feature. Without being explicitly specified in the program, concurrency of the program is exploited

and data-flow synchronization is made automatically at and under the language implementation level. The implicit data-flow synchronization mechanism has a greate advantage in eliminating synchronization errors. The language is powerful enough to describe *everything* as long as it can be modeled as communicating processes. Any processes with small grain size or short lifetime can be implemented without large overhead.

Flat GHC is a subset of GHC. Only unification and calls to certain built-in predicates are allowed in the guard part of a clause. This makes efficient implementation easier without losing the essential descriptive power of the language.

The KL1 language has several extensions from the original Flat GHC (Chikayama et al. 1988). One of the most essential extensions is the notion of *sho-en*. *Sho-en*, or *manor* in English, is similar to the meta-call mechanism seen in other concurrent logic programming languages. Like the meta-call, the *sho-en* mechanism can be used to protect the outside of the *sho-en* from failure inside the *sho-en*. In addition, limits on computational resources (e.g., execution time and memory) consumed in a *sho-en* can be controlled and monitored from outside. This feature is essential in writing an operating system in KL1.

The KL1 language also supports the functions of priority control and load allocation. Execution priority can be specified for a *sho-en* or goal. A goal can be allocated to a certain processing node specified by a node number. The annotation, which specifies the priority or node number for a goal (e.g., goal@priority(X) or goal@node(Y)), is called the pragma. The pragma only affects execution efficiency of a program but is independent of the program semantics. This feature is useful for tuning the execution efficiency and the load balance of a program. Current experimental implementation on the Multi-PSI supports 4096 priority levels which will be used in user programs.

3 PIM PILOT MACHINE: PIM/p

3.1 Target Performance

PIM/p is the first pilot machine for our target PIM system. Several other models are also being developed. The performance of a PIM/p processing element is 200K to 500K LIPS. A PIM/p system will contain 128 processing elements and will achieve 10M to 20M LIPS of effective performance.

3.2 Overall Structure

PIM/p has the hierarchical structure shown in Figure 1. Eight processing elements (PEs) form a cluster with shared memory and bus. The PIM/p consists of 16 clusters connected by the inter-cluster network. Processing elements in a cluster share the same address space, whereas address spaces for each cluster are separated. A cluster forms a substructure with low communication cost and response time which is utilized in the load allocation. The hierarchical structure allows an easier and better implementation of the dynamic memory management than the structure sharing all the memory. It is also more effective to reduce the physical memory size per a processing element than the size of the completely non-shared memory structure.

3.3 Processing Elements

The PIM/p processing element is designed for the efficient execution of KL1-B (Kimura and Chikayama 1987), which is the common abstract instruction set for the KL1 used in our inference machines. KL1 programs are compiled to KL1-B instructions and then translated to target machine instructions. PIM/p has a RISC-like instruction set which is executed in a four-stage pipeline (Goto et al. 1988). To reduce the static code size, PIM/p supports the conditional macro-call feature (Shinogi et al. 1988), which is a sort of subroutine call to the internal instruction memory (IIM) with a specialized argument passing mechanism. It is used to implement complicated KL1-B functions by common modules in the IIM with low invocation overhead. The tag architecture has been adopted for the PIM/p processor with an 8-bit tag and 32-bit data. Each data word is aligned with the 64-bit memory boundary in the current implementation.

The role of KL1-B is similar to that of WAM (Warren 1983). The major differences are the synchronization feature and functions for incremental garbage collection, called MRB (Chikayama and Kimura 1987).

A PIM/p processing element is implemented on a single board with several custom CMOS LSIs and about 20 static RAMs, as shown in Figure 2. The pipeline cycle is expected to be 50 nanoseconds. There are two cache memories, instruction cache and data cache with 64K bytes each. They are write-back caches with the cache coherency protocol (Matsumoto et al. 1987). They also support the word locking mechanism and software cache functions optimized for KL1 execution (Goto et al. 1988). The common bus cycle is the same with the processor pipeline cycle. The bus data width is 64 bits. The network interface unit (NIU) and the floating point unit (FPU) are the co-processors of the CPU. The peak performance for the append program will be over 600K LIPS including MRB garbage collection.

3.4 Inter-cluster Network

A message exchanging network with hyper-cube topology has been introduced to connect PIM/p clusters, placing each cluster on a hyper-cube node. Inter-cluster communication is invoked by a unification or a goal forking across the cluster boundary in KL1 program exe-

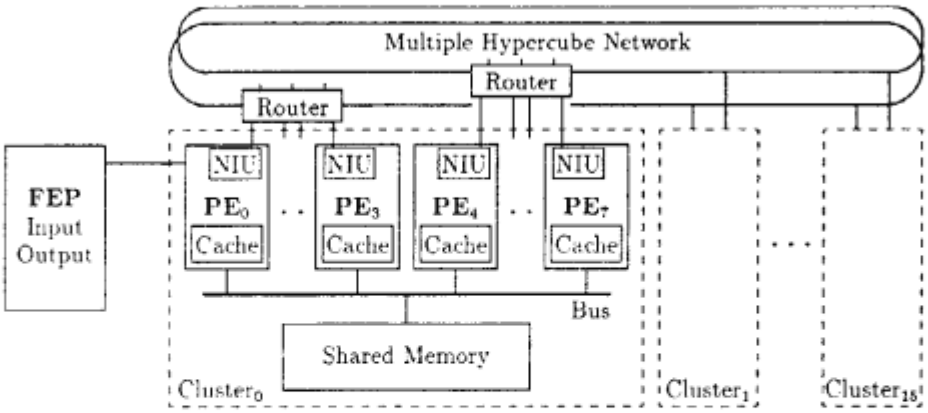

Figure 1: The Pilot Machine: PIM/p

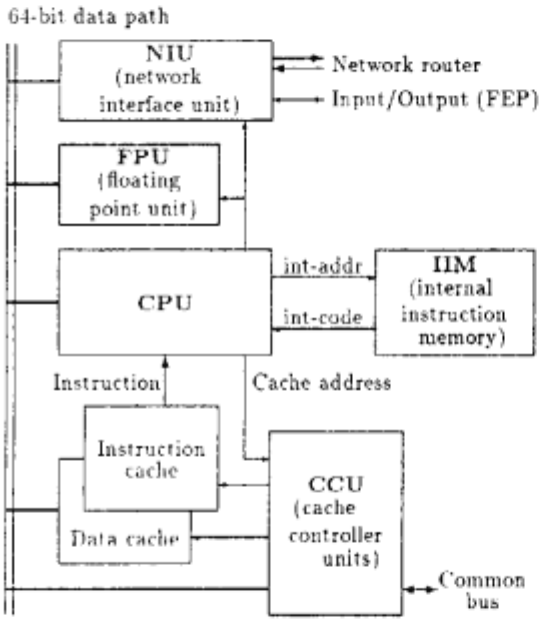

Figure 2: PIM/p Processing Element Configuration

cution. Two sets of hyper-cube networks are used to increase the network bandwidth as shown in Figure 1. Every processing element has a connection to one network. A message is routed to the destination cluster automatically, as preset. Each communication path has a throughput of 20M bytes/second in both directions.

3.5 KL1-B Implementation Issues

KL1-B is the abstract instruction set which defines the basic behavior of the KL1 language processor. Each KL1-B instruction is normally expanded into a RISC instruction sequence. However, several complicated functions such as resource management in the *sho-en*, inter-cluster processing mechanisms, and garbage collection, are not open-coded as are other more primitive features. They are implemented as common modules in the IIM, and invoked when certain conditions occur.

There were many difficulties in the KL1-B implementation, especially in the inter-cluster processing mechanisms, such as the distributed resource management for the *sho-en* (Ichiyoshi et al. 1987), the efficient goal termination detecting mechanism among clusters (Rokusawa et al. 1988), export and import mechanisms of reference pointers between clusters, and the distributed unification mechanism (Ichiyoshi et al. 1988). These problems were basically solved in the KL1-B implementation on the Multi-PSI system.

3.6 Garbage Collection

Efficient garbage collection is essential for the KL1 implementation because it had to be implemented in heap-based style rather than stack-based and consumes much memory at run time. Intra-cluster and inter-cluster garbage collection will be implemented separately for ease of implementation and efficiency.

Both incremental and stop-and-collect GC will be implemented for the intra-cluster GC. An incremental GC, MRB GC (Chikayama and Kimura 1987), is being experimented on the Multi-PSI. It is a simple subset of reference counting, only distinguishing between single and multiple references by one bit. Structures with single reference often appear in KL1 programs. MRB GC is effective in such cases, making the working set size smaller and the interval of stop-and-collect GC longer.

Inter-cluster reference pointers are implemented with export and import tables which are a sort of address translation table. The role of inter-cluster GC is to maintain these table entries and to remove garbage inter-cluster pointers. The weighted export counting (WEC) scheme (Ichiyoshi et al. 1988) has been introduced to realize the incremental GC for these pointers. WEC is an application of weighted reference counting (Watson and Watson 1987, Bevan 1987).

3.7 Goal Scheduling

A goal is a unit of parallel execution and scheduling. A goal is replaced or expanded into the body goals of a clause in the invoked predicate when the clause is committed. The leftmost body goal is executed immediately while others are pushed into the ready-goal stack. That is, the depth-first scheduling is adopted for body goals and also for the ready-goal stack. Since each goal is associated with its execution priority, ready-goal stacks are managed corresponding to each priority level. When all the guard unifications are suspended, the goal is suspended, hooking itself to the variables that caused the suspensions (Ichiyoshi et al. 1987). The goal is resumed when one of those variables is instantiated. That is, the *non-busy waiting* method has been adopted.

How to keep the processing load well-balanced is a key issue in making the best use of parallel processing resources. An automatic load balancing is adopted in a cluster. Each processing element has a goal stack for the highest-priority ready goals to avoid conflicts of access to the common ready-goal stack. The highest-priority goals are distributed to keep the processor loads in good balance. We found *on-demand* distribution to be an effective way of realizing a good balance within a cluster while reducing the amount of wasteful communication among processors (Sato and Goto 1988). In this scheme, an idle processor sends a request for load allocation to a busy processor.

Load distribution among clusters should be done carefully because the communication cost is more expensive than within a cluster. Several distribution schemes have been tested on the Multi-PSI in which the load distribution algorithms are buried in the KL1 programs specifying goal allocation by the pragma (goal@node(X)). Several standard schemes will be supported by the operating system in the future.

4 OPERATING SYSTEM

PIMOS is the common operating system for our inference machines. It has been developed on the Multi-PSI. The primary functions of the PIMOS are I/O resource management, execution control of user tasks, and management of programs (Chikayama et al. 1988). The programming system has not been implemented in the current version. PIMOS has the following characteristics.

Logic-based: PIMOS is described entirely in KL1, without using extra-logical features at all. Even I/O devices connected to the inference machine have *logical* interfaces. Each I/O device looks like a perpetual process from the user program, communicating through a stream interface.

Integrated: Although PIMOS is an operating system for parallel machines, it is an integrated operating system working as one unit, rather than consisting of many operating systems distributed on processing elements.

Born parallel: Various functions of the PIMOS, which exploit the power of parallel inference machines, have been implemented to work in parallel to prevent the PIMOS becoming a bottleneck in highly parallel systems.

Practical: Although PIMOS has many experimental features, its purpose is to provide a practical programming environment for parallel algorithm development, keeping robustness.

5 PROGRAM DEVELOPMENT

Three systems providing the program development environment have been developed. The primary system is called the PIMOS development support system (PDSS), written in C, and an operating system called Micro-PIMOS. All the KL1 features except for real parallel execution are provided with several debugging facilities. The system was mainly used for the development of the PIMOS and in the early stage of the application program development.

The second system, Multi-PSI, and the third system, Pseudo Multi-PSI, have been developed in parallel. The Multi-PSI (Taki 1986, Taki 1987) is a collection of PSI-II processors (Nakashima and Nakajima 1987) connected by a two-dimensional mesh network (Takeda et al. 1988). The full system contains 64 processing elements. KL1-B has been implemented in microprogram. The processing element speed is approximately 150K LIPS for the append program with MRB GC. The KL1 compiler and debugging support system were implemented on the front-end processor (FEP), PSI-II. The FEP also supports I/O functions controlled by the PIMOS. The Multi-PSI is used with the PIMOS as a main tool for the development and evaluation of various concurrent algorithms and load balancing schemes. Three full systems have been working since 1988.

The Pseudo Multi-PSI is a simulator of the Multi-PSI implemented on a PSI-II machine. The behavior of the Multi-PSI with any number of processing elements can be simulated. KL1 programs are executed by the same microprogram as that of the Multi-PSI, overlaid on the PSI-II micro code area. The KL1 execution speed for simulating one processing element is equivalent to that of the Multi-PSI, which is unusual for simulation systems. The system performs the same as the real Multi-PSI except for the round robin scheduling of the pseudo processors and smaller memory size. The Pseudo Multi-PSI is mainly used for general KL1 program debugging, both for intra-PE errors and inter-PE errors.

Four KL1 sample programs were developed for the demonstration at the FGCS'88 conference. They are the natural language parser, PAX; a board game, tsume-go; the packing piece puzzle; and the shortest path finding problem. There is a total of 14.5K source program lines in KL1, and the development period is around three months, with eight programmers. Several concurrent algorithms and load balancing schemes are being experimented in the program development.

6 CONCLUDING REMARKS

In the past, research on parallel computer hardware has been relatively independent from parallel software research. Basically, the hardware or system implementation research was for implementing more efficient environments for executing *already existing* software.

The principle of the parallel inference systems development in ICOT is rather different in this point; software and hardware research should be combined more closely. Software or even algorithms optimized for sequential machines may not be optimal for parallel machines. Thus, software should change when the hardware changes.

However, there is a chicken-and-egg problem: without parallel hardware, practical parallel software cannot be developed; without parallel software, it is hard to know what kind of parallel hardware is appropriate. ICOT's approach to solve this problem is stepwise bootstrapping. The first step was to settle on a software-hardware interface, namely, the KL1 language, and implement it (as the multi-PSI system). The next step is to develop various software systems on it (including PIMOS). By running the resultant software, many unknown parameters of the behavior of parallel software will be revealed. The next generation (the PIM/p system) will be based on these experiences, and software will be developed on this machine.

Development of the PIM/p hardware system will be completed in 1989, and then the improved KL1 language processor will be implemented on it, inheriting various research results from the software development on the Multi-PSI system.

Our challenge to develop the parallel processing technologies for large-scale MIMD multiprocessors from both software and hardware sides must be the *creation* of a *new computing culture*.

Acknowledgement

This paper is based on various R&D activities carried out by many researchers at ICOT and cooperating manufacturers.

REFERENCES

(Bevan 1987) D.I. Bevan. Distributed Garbage Collection using Reference Counting. In *Proceedings of Parallel Architectures and Languages Europe*, pages 176–187, June 1987.

(Chikayama and Kimura 1987) T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 276–293, 1987.

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

(Clark and Gregory 1984) K. Clark and S. Gregory. Notes on Systems Programming in PARLOG. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 299–306, Tokyo, 1984.

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

(Ichiyoshi et al. 1987) N. Ichiyoshi, T. Miyazaki, and K. Taki. A distributed implementation of flat GHC on the Multi-PSI. In *Proceedings of the Fourth International Conference on Logic Programming*, 1987.

(Ichiyoshi et al. 1988) N. Ichiyoshi, K. Rokusawa, K. Nakajima, and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, November 1988.

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

(Matsumoto et al. 1987) A. Matsumoto et al. Locally Parallel Cache Designed Based on KL1 Memory Access Characterstics. TR 327, ICOT, 1987.

(Nakashima and Nakajima 1987) H. Nakashima and K. Nakajima. Hardware Architecture of the Sequential Inference Machine: PSI-II. In *Proceedings of 1987 Symposium on Logic Programming*, pages 104–113. San Francisco, 1987.

(Rokusawa et al. 1988) K. Rokusawa, N. Ichiyoshi, T. Chikayama, and H. Nakashima. An Efficient

Termination Detection and Abortion Algorithm for Distributed Processing Systems. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume 1 Architecture, pages 18–22, August 1988.

(Sato and Goto 1988) M. Sato and A. Goto. Evaluation of the KL1 Parallel System on a Shared Memory Multiprocessor. In *Proceedings of IFIP Working Conference on Parallel Processing*, Pisa, Italy, April 1988.

(Shapiro 1983) E.Y. Shapiro. A subset of Concurrent Prolog and Its Interpreter. TR 003, ICOT, 1983.

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

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

(Taki 1986) K. Taki. The parallel software research and development tool : Multi-PSI system. In *Programming of Future Generation Computers, edited by K.Fuchi and M.Nivat, North-Holland, Amsterdam, 1988*, pages 411–426. Also in TR 237, ICOT, 1986.

(Taki 1987) K. Taki. Measurements and evaluation for the Multi-PSI/V1 system. In *Programming of Future Generation Computers II, edited by K.Fuchi and L.Kott, North-Holland, Amsterdam, 1988*, pages 365–391. Also in TR 370, ICOT, 1987.

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

(Uchida et al. 1988) S. Uchida, K. Taki, K. Nakajima, A. Goto, and T. Chikayama. Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project. In *Proc. of the International Conference On Fifth Generation Computing Systems 1988*, Tokyo, November 1988.

(Warren 1983) D.H.D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, Artificial Intelligence Center, SRI, 1983.

(Watson and Watson 1987) P. Watson and I. Watson. An Efficient Garbage Collection Scheme for Parallel Computer Architecture. In *Proceedings of Parallel Architectures and Languages Europe*, pages 432–443, June 1987.