ICOT Technical Report: TR-237

TR-237

The Parallel Software Research and Development Tool: Multi-PSI System

by

K. Taki

March. 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

The Parallel Software Research and Development Tool : Multi-PSI System

Kazuo Taki
ICOT

Abstract

The Multi-PSI system is a parallel software research and development tool. The purpose of the system is to proceed with parallel software research to realize a large-scale parallel inference machine (PIM) system for knowledge information processing.

The Multi-PSI contains several PSI machines CPUs (8 for Multi-PSI-V1 and 64 for V2) and a dedicated lattice network. It supports a real parallel execution environment and a software development system for parallel software research. The parallel software research themes are such issues as languages, operating system, program development system, basic research like load balancing and algorithms, and application programs. All of these are important for constructing the PIM system. The most important theme for the knowledge processing parallel machine is the dynamic load balancing technique, which is less important for numerical processing.

Relations between software research and PIM research, and also Multi-PSI architecture and PIM architecture are discussed. They are intimately related considered from the software point of view. System development schedules and the software research items are also discussed.

Multi-PSI-V1 currently consists of six PSI machines. The parallel logic programming language KL1 is based on GHC and implemented in ESP, the sequential language of the PSI machine. The basic part of the system is currently being tested. Small parallel programs like the eight queen have begun to work.

A dynamic load balancing method is also described as an example to clarify research directions.

1 Introduction

We have been pursuing research on the Parallel Inferance Machine (PIM). The aim is to build a parallel computer system focusing on high performance knowledge information processing [4,2]. This is the most important research theme of Japan's FGCS project. In the first three-year stage of the project, PIM research sought to develop basic technologies of machine architecture and parallel execution mechanisms, and accumulated several methods for constructing machine hardware and parallel execution methods of logic programs [3]. But several serious problems of parallel software were revealed by

the research, and we recognized that parallel software research has to be done along with parallel architecture research. Let us enumerate the parallel software problems; they can be thought of as general problems found not only in the PIM research, but also in various research projects on parallel machines around the world.

1. There are no (or few) examples of large-scale parallel programs for real world complex applications. This makes it difficult to evaluate an experimental machine or a machine architecture.

2. Since few implementation techniques of a parallel programming language have been accumulated and the language specification can vary because wide experience in using it is lackng, it is not easy to extract guiding principles for optimizing a machine architecture from language technologies.

3. Debugging and recompiling of even small parallel programs takes much time because experimental parallel machines are not intended as program development systems. Development of efficient debugging methods for parallel programs is a non-trivial problem.

4. Not much progress has been made on the basic research on running parallel machines efficiently to solve problems of knowledge information processing. Problem areas include parallel algorithms, dynamic load balancing methods, and programming paradigms of large scale complex parallel programs.

5. Basic operating system functions for a parallel machine, such as execution control, resource management, and input and output, have not been sufficiently studied yet.

To approach the study of these problems, we think it is important to prepare a parallel program research and development environment first in which programs can be executed in parallel and developed and evaluated efficiently. We will then proceed with extensive parallel software research within this environment. However, the PIM is not suitable for constructing a practical research and development environment. So we chose the more realistic approach of using the personal sequential inference machine (PSI) [9,6] developped in the first stage of the project for ease of system construction and modification.

PSIs are connected by a dedicated network to configure a loosely coupled multiprocessor system that can be used to construct a parallel software research and development environment. This is the Multi-PSI system, whose architecture is designed to share the architectural basis with the large-scale PIM, seen from the software point of view. We will implement a parallel language processing system and basic operating system on the Multi-PSI hardware first, and make it a core part of the research and development environment. Then using and improving the environment core, we intend to proceed with extensive parallel software research on load balancing, parallel application programs, etc.

We think the parallel software research and the PIM research are related like the chicken and the egg. Research on one encourages and supports that on the other, each providing guiding principles for the other. It is difficult to decide which is prior, because these individual research fields are intimately linked. The Multi-PSI system is the first core system to circulate "the chicken and egg loop". The system started as a core part of the research and development environment designed for ease of realization, and parallel software research will begin using it. As the environment is improved and expanded being used in the research, the parallel software research also expands. In turn,all this clarifies requirements for the real PIM architecture and provides basic technologies for PIM implementation.

Following this approach, we plan two systems; the first is Multi-PSI-V1(version 1) focusing on ease of realization, and the second is Multi-PSI-V2 improving machine scale, functions, and speed. We are designing the system to have good reliability, ease of use and good performance for writing, debugging, and executing large-scale parallel programs.

In this paper, we present the research themes, relations to PIM research, schedule, and concrete research items of the parallel software research and development tool : the Multi-PSI system.

2 Parallel Software Research Theme

Problems of parallel software revealed in the PIM research have already been listed in Section 1. In this section we will reformulate these problems as research themes for the Multi-PSI system. The purpose of parallel software research is to build a software system based on new concepts which can utilize the large-scale PIM hardware with high-performance for knowledge information processing. We also intend to extract guideline principles for PIM architecture from the software research process. A software system based on new concepts means that all research items like programming languages, system software, application software, algorithms, program development support, etc., have to be completely reformulated for parallel execution. The research themes are arranged as follows.

1. Parallel logic programming language KL1 (Kernel Language version 1) and language processing system.
2. Operating system PIMOS (Parallel Inference Machine Operating System), focusing on execution control and resource management.
3. Program development system and debugging method for parallel programs.
4. Basic research on issues such as parallel algorithms and dynamic load balancing.
5. Large-scale parallel application programs.

Each item will be discussed separately later. The first reserach step is to implement the basic parts of 1, 2, and 3 for building an R&D environment core. We can then proceed to 4 and 5 using the environment, feeding back results to improve the core, and so on. The core will meet half of the research goal when it is implemented, whereas the latter require trials of several methods on the environment. We think that much of this research is useful for general parallel machine research around the world, but some of it is particularly important for large-scale parallel machines designed to handle high-volume knowledge information processing. This also will be discussed in a later section.

3 Relationship with PIM Research

Parallel software research needs to concentrate on the large-scale PIM for the future, but directions in parallel software research may be influenced by the Multi-PSI architecture. In this section, the relationship between the target PIM and Multi-PSI architecture will be clarified, and the study of parallel software problems shared by the target PIM and Multi-PSI will be touched on.

3.1 Basic Image of the PIM

PIM is a general-purpose, parallel symbolic processing machine for large-scale knowledge information processing at speeds in the range 100 - 1000 M LIPS (Logical Inference Per Second). To achieve this speed, it seems that PIM needs to be a large-scale multi-processor system containing more than 1000 processing elements (PE), based on calculations from single processor performance. And PIM cannot use tightly coupled architectures like those of the shared memory type, because of its large number of PEs. It has to employ a loosely coupled architecture using some network connections. We think the MIMD machine model is suitable for PIM, because knowledge information processing has the tendency to require complex computing processes: dividing one big problem into subproblems, each of which requires different knowledge data and different computing procedures, and each executed concurrently. We also think a dynamic process allocation to each PE is required, because computation in knowledge information processing strongly depends on the input data, and computing load in each PE varies dynamically depending on data.

The target machine requires more local memory for each PE than a MIMD machine for numerical processing. The first reason for this is that a symbolic processing machine consumes a lot of memory at run time. The second is that each PE has to keep program copies to eliminate program transfer, because process allocation occurs dynamically. The last reason is that knowledge information processing requires so much knowledge data, stored as data or programs.

We intend to build a logic programming machine, but the sketches above are not restricted to such a machine, for they can be applied generally to large scale parallel machines for symbolic processing. Basic requirements appear to be :

1. Large scale MIMD machine containing more than 1000 PEs.
2. Loosely coupled architecture with network connection.
3. Dynamic process allocation to each PE.
4. Large amount of local memory for each PE.

3.2 Relations between Architectures and Software Research

The Multi-PSI system adopts an architecture of loosely coupled MIMD-type multi-processors, containing a small to middle number of PEs (64 max.) connected by a network, and each PE (PSI machines) has a large amount of main memory (16M words). The Multi-PSI architecture aims to realize the PIM described above. However, network architecture and topology, not mentioned above, may be different with the target PIM's. How do such differences affect parallel software research?

We assume that the software processes communicate with each other, and that the communication cost of internal PE communication and inter PE communication are very different for loosely coupled multi-processor systems. On the basis of this assumption, we think that the parallel software system has to pay close attention to the communication locality, which at least distinguishes internal PE communication from inter PE communication. Multi-PSI and PIM look the same as software systems if such care is taken over this limited communication locality, although the networks are different.

A network that connects all PEs at the same distance is difficult to realize when it is applied to the very large scale PIM. We assume that the cost of communication to an adjacent PE is less than that to a distant PE. The lattice network, used in the Multi-PSI system, has the same character. Research on parallel software systems which control the communication locality is very important for construction of very large-scale parallel machines, no matter which network topology is chosen as far as the assumption can be applied. In other words, it is important to find a basic software method which maintains communication locality on networks with different topologies, because the network architecture of the target PIM has not been fixed yet.

If the PIM contains clusters, in which several PEs are tightly coupled, and contains an inter-cluster network, a cluster can be thought of as a PE of the Multi-PSI. That is, if the PIM takes a double-layered network architecture, in which the upper network is loosely coupled and the lower is tightly coupled, the upper loosely coupled network should correspond to the Multi-PSI network as far as parallel software is concerned.

3.3 Important Research Theme for PIM

Imbalance in computing loads between PEs tends to arise in parallel machines for knowledge information processing when static process allocation is used. This was discussed in Section 3.1. Dynamic load balancing methods, which realize dynamic process allocation to PEs efficiently, have to be found for execution of large scale knowledge processing

programs on PIM in order to decrease effects of large and frequent load changes. This is in contrast to the numerical processing MIMD-type machine, which often requires only static process allocation.

The dynamic load balancing method discussing here has to cover not only computing load balance control, but also control of communication locality referred to in the last section. The term "dynamic load balancing" is used in this sense below.

4 Research and Development Schedule

We plan to develop the following systems to promote parallel software research.

4.1 Multi-PSI Version 1 (1986 - 1987)

Multi-PSI-V1 contains 6 to 8 PSI machines connected by a dedicated lattice network; each PSI has its own I/O devices (Fig.1). The purpose is to implement a basic part of the KL1 language processing system and PIMOS experimentally in a short period. The language processing system is written in ESP, the system description language of PSI [1], and runs on SIMPOS, the operating system of PSI [5]. Simple parallel programs for system evaluation are executed, and dynamic load balancing and debugging methods for the distributed execution environment are studied. The dynamic characteristics are measured for construction of the next system.

4.2 Multi-PSI Version 2 (1987 - 1989)

A smaller and faster version of the PSI machine without I/O is used for the PEs, and 16 to 64 PEs are connected with a new version of the network, which is improved in speed and functions. One PSI machine is used as a front-end processor (Fig.2). The KL1 language execution system is written in firmware to realize high execution speed. The full PIMOS operating system is implemented, and the dynamic load balancing method, parallel algorithms, and large-scale parallel application programs are studied.

4.3 Pseudo-multi-PSI

Two simulators of Multi-PSI on the single PSI machine are prepared, one for Multi-PSI-V1 and the other for V2. The Pseudo-multi-PSI system shares the basic part of KL1 language processing system with the Multi-PSI system. The pseudo system is used for initial debugging of the language execution system, debugging of user programs, measurement of programs' dynamic characteristics, and measurement of data required for reformulating the load balancing strategy. It is mainly used for program debugging until an efficient debugging method on the distributed execution environment is found, and also used for measurements which are difficult on the real parallel machine.

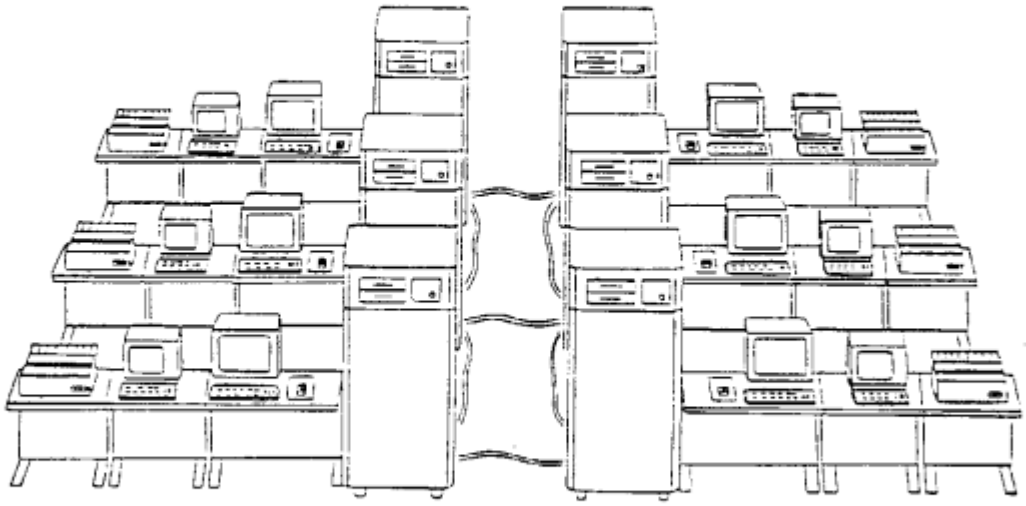


Fig.1 Multi-PSI-V1 System

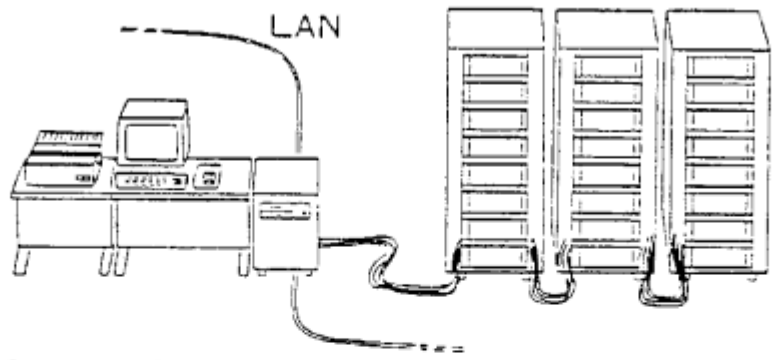


Fig.2 Multi-PSI-V2 System

5 Research Items

5.1 KL1

The parallel logic programming language KL1 implemented on the Multi-PSI has the layered structure shown in Fig.3. The KL1-c (core) is the base language which gives parallel logic programming language semantics to KL1. The specification of KL1-c is basically Flat GHC with meta-logical functions and system control functions added. Flat GHC is a subset of the parallel logic programming language GHC [7,8], that is, the user-defined predicate call in the guard part is inhibitted and a few other limitations are added for ease of implementation. Language features include commited-choice and-parallel execution, stream communication between body goals, and synchronization in the guard part.

KL1-u (user) is a user language constructed from KL1-c and modularization mechanisms. PIMOS and other parallel programs for system evaluation are written in KL1-u. We think that the KL1-u should be expanded to support not only the modularization mechanism, but also some specifications which guide the programming style of large-scale parallel application programs.

On the other hand, the language functions of KL1-c are broken down at the machine language level. This is KL1-b (base). The language execution system on the machine supports KL1-b, in which the distributed unification function, dynamic load balancing function, interrupt and trap functions, etc. are implemented. Distributed unification is the base function of inter processor communication.

KL1-p (pragma) consists of notations to control dynamic load balancing and execution priority level. Pragma can be attached to a program without changing the program's semantics. A user writes a program in KL1-u, then pragma is attached, the program is compiled into KL1-b, and then executed.

The language execution model of KL1 is based on the fine grain parallelism because each goal may be suspended, resumed and passed to another PE. However the coarse granuality is important when the execution efficiency on a sequential PE is considered. We are studying how to attain coarse granuality using a language based on fine granuality. Progarmming style and optimization mechanisms of the language execution system is being studied in order to eliminate the goal suspension which causes the context switch and decreases granuality. The advantage of using a fine grain language is that it makes dynamic load reallocation easier. Because the formal execution unit of the fine grain language is small, so there is much chance to divide the computing load in a computation of such a language. It is important that coarse granuality is attained only by optimization of the fine grain language.

5.2 PIMOS

PIMOS is an operating system mainly designed for execution control and resource management of PIM. The functions listed below are implemented, parts of which are written in KL1-u and parts in firmware (or ESP for Multi-PSI-V1).

1. Goal scheduler and execution priority management function as the internal PE execution control functions.

2. The dynamic load balancing function as the inter PE execution control and the computing resource management function.

3. Memory allocation and release functions, and parallel garbage collection as the resource management functions.

4. Interrupt and trap functions.

The above are the kernel functions closely related to the language execution system. The functions listed below are upper layer functions containing user services.

5. Input and output.

6. Program code distribution and management.

7. User process (task) management.

A minimum subset of these functions are implemented first, then improved and expanded in the process of being used. The load balancing function has a close relation to the basic research discussed in the next section.

5.3 Load Balancing and Parallel Algorithms

Dynamic load balancing with communication locality control is important for the loosely coupled parallel machine for knowledge information processing. It was touched on in Section 3 and is described in more detail below.

1. We think that there is no serious prospect of full automatic load balancing. It is difficult to get an adequate balancing result when the system controls the load balance without knowing the program characteristics beforehand. The user writes the expected load characteristics into the program using some kind of abstracted method with pragma, then the system allocates load (processes) to real PEs guided by the pragma. The system is also guided by the pragma in reallocating the load when imbalance occurs.

2. The dynamic load balancing method should be applied to a very large-scale PIM, containing more than 1000 PEs

3. Dynamic load balancing methods, which do not require the programmer to consider the number of real PEs, should be studied. That is, the same program should run on systems containing different numbers of PEs without modification. We need to find a method that allows a user to write a program considering network characteristics in some abstructed level, rather than number of PEs. That is, the user only needs to consider the hardware at an abstruct level with such general facts as that cost of communication to distant PEs is high, or that PEs are connected in two dimensions.

An example of the dynamic load balancing method under consideration is discussed in Section 7.

We think parallel algorithms should be studied focussing on these two issues.

1. Extraction of parallelism in the problem.
2. Maintaining communication locality between processes.

In these, 2 involves research of algorithms to attach the pragma to programs making it possible for PIMOS perform load balancing control effectively. For example, a programmer takes up some frequently used search problem, and during the programming process, he considers extraction of parallelism and attaching the pragma effectively for load balancing and communication locality, then he arranges the programming process into an algorithm. Research on parallel algorithms, with close relations both to dynamic load balancing and parallel programming, is extremely important.

5.4 Program Development Support

Program development support for Multi-PSI requires an editor, interpreter, debugger, compiler, program library, and a measuring tool for determining the program's dynamic characteristics used for reformulating the pragma.

The Pseudo-multi-PSI is utilized as a program development system first to realize these functions, because many difficulties are expected in parallel program debugging, especially on the real multi-processor system. The editor of the PSI machine is used for KL1 editing, the compiler and program library for the KL1 are written in ESP and executed as PSI programs. The interpreter, debugger, and measurement and evaluation functions are implemented in the Pseudo-multi-PSI system.

Programs, whose bugs are completely killed on the Pseudo-multi-PSI, are loaded and executed on the Multi-PSI. However, it is difficult to predict what type of bugs will arise in the real multi-processor system, so simple program tracer and measurement facilities are supported on the Multi-PSI initially, and they will be expanded later if necessary. At any rate, the parallel program debugging method and program development system require many rounds of scratching and building.

5.5 Console and Front-end Functions

Initialization and maintenance of hardware, tracing and debugging close to the firmware level, and initial program loading increase in difficulty with increase in the number of PEs. These functions are performed by the front-end machine, PSI-II, for the Multi-PSI-V2.

I/O device control, such as bit-mapped display and disk control, are also carried out by the front-end machine, supporting a higher level logical interface to the KL1 world.

For the Multi-PSI-V1, front-end and console functions are supported by software on each PSI.

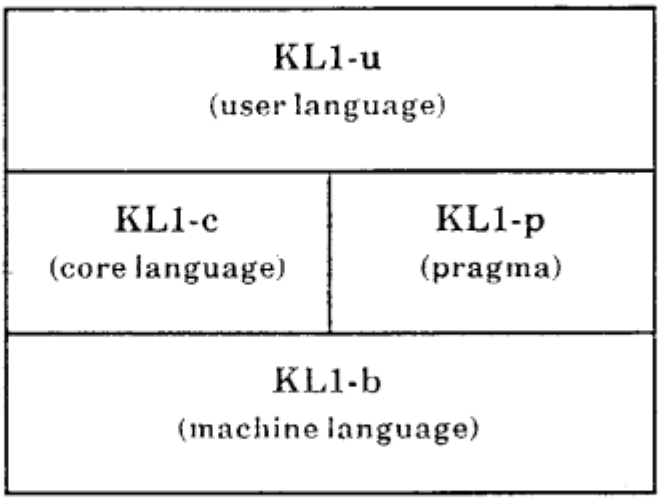


Fig.3 Layered Structure of the KL1

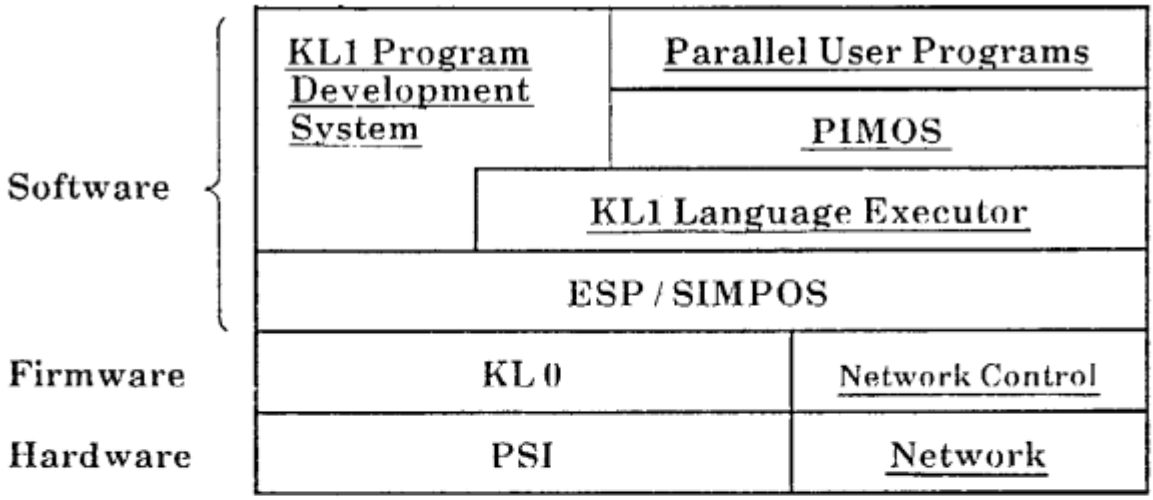


Fig.4 Configuration of the Multi-PSI-V1

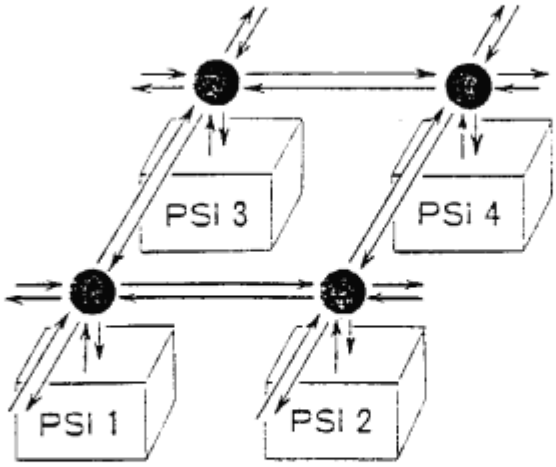


Fig.5-(a) Configuration of the Connection Network

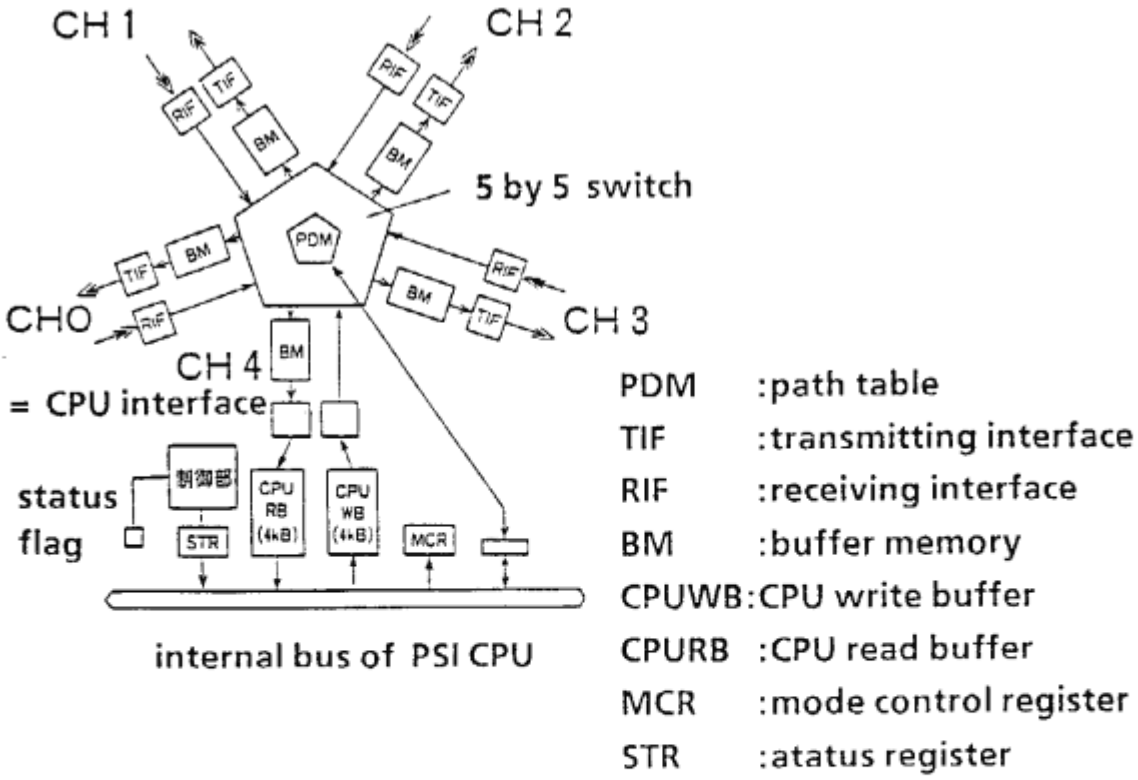


Fig.5-(b) Block Diagram of the Network Hardware

6 Multi-PSI Version 1 System

6.1 Overview of the System

The PEs are PSI machines, 6 to 8 PEs being connected on a dedicated lattice network. There is no shared memory and inter PE communications are performed by exchanging message packets. A user does not handle special message primitives, but simply writes goals with pragma and unifications; and messages are automatically generated by the language execution system. There are two major message types; one concerns KL1 goal management, such as goal sending and termination reporting; and the other concerns unification across the PEs. Each PE has a different address space, and when a reference pointer is passed from one PE to another PE, the internal address of the source PE is converted to a global identifyer (ID) and then sent. The address translation table between internal address and global ID is maintained in each PE. This address translation mechanism is introduced for the purpose of local garbage collection in a PE without affecting other PEs.

Fig.4 shows the configulrtion of the Multi-PSI-V1. The underlined sections have been newly developed, and the other sections represent the PSI machine system itself.

6.2 Connection Network Hardware

A lattice network is used because it is easy to install Another reason is fitness to the experimentation of dynamic load balancing since the lattice network is most general one which contains the logical distance between PEs. But some different network topologies can be simulated on the Multi-PSI-V1 because the network speed is relatively high compared with the PE execution speed.

Fig.5 shows the configulation of the network hardware. It is installed in the option slots of CPU internal bus, and four cables are used to connect adjacent four PEs. Each connection of PE to PE is called a channel. A channel contains two signal sets, one is for transmittion and another is for receiving. Data is transfered in 10-bit parallel containing a parity bit. The transmission rate is approximately 500k bytes/sec for each direction.

Message packets are constructed by software. Each packet has a destination PE number in the packet header. The network recognizes the destination PE number, and if equal to its own, the network takes the paket into its receiving buffer. If the PE number is different, the network looks up the path table to get the channel number to which the packet should be re-transmitted. The path table is supported in hardware and is initialized by software. If the path table is set up correctly, the message packet is relayed to the destination PE automatically. When packets arrive at each channel concurrently, re-transmission can also be done concurrently as long as the destination channel numbers do not conflict. If the destination channel numbers conflict, the packet arriving at the channel with the larger number is put on hold. There are simple routing methods which avoid this network deadlock. One involves passing the packet to the horizontal direction prior to the vertical direction when the packet should be passed

Assume a network in which the distance between PEs can be defined.

(1)

Assume computation power is distributed uniformly on unit plane (we call it PPP).

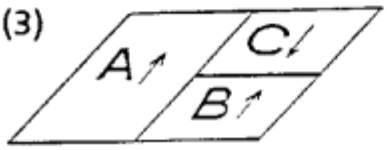
(2)

Initially, a goal has a given area and direction in PPP.

:- P.

Whenever a goal is reduced to its sub-goals, its given area is also split.

(3)



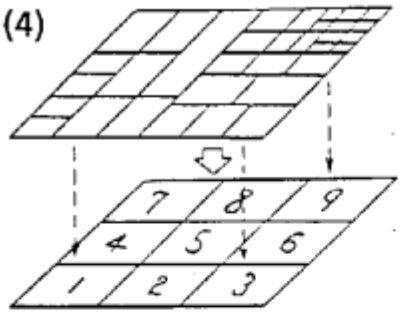
P :- A, B, C.

add pragma

P :- A↑: {B→:C←}←.

Sub-goals are further divided into sub-sub-goals.

(4)

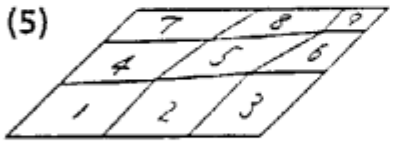


PPP, devided for sub-goal execution.

correspondence to physical processors.

The correspondence between PPP and physical processors are decided by the system

(5)



When there exists load imbalance, system changes the correspondence.

Fig.6 A Load Balancing Method

to the PE on the diagonal direction. CPU interface of the network hardware contains two 4K-word FIFO buffers for packet transmission and receiving. There is also a small FIFO buffer at the exit to each transmission channel. These buffers reduce network bottlenecks.

7 A Load Balancing Method

We are studying dynamic load balancing according to the approach discussed in Section 5.3. The following is an example of how this dynamic load balancing method works.

A programmer assumes a unit plane on which the computing power is distributed uniformly. In this model a lot of PEs are connected in two dimensions. We call it the processor power plane (PPP). The programmer writes a program to divide the problem into subproblems. He also divides the unit plane into subplanes and allocates subproblems to each subplane; this process is controlled with the pragma. In a subproblem level, he considers the corresponding subplane as an unit plane. He then continues to divide the problem and the plane, and repeats allocation. He controls division and allocation in this procedure in order to maintain the communication locality between subproblems (Fig.6 (1)-(4)).

On the other hand, the system manages the mapping between PPP and physical PEs, and allocates subproblems to each PE at run time. Initially the mapping between PPP and PEs is coherent. After a while, load imbalance between PEs may occur. This means that some subproblems create large computing load exeeding the programmer's estimation, but some don't. In this situation it can be said that computing load is distributed incoherently to the PPP. If the imbalance continues for a long time, the system reformulates the mapping between PPP and the physical PEs using local communication only. That is, PPP area assigned to a busy PE is decreased, while that assigned to an idle PE is increased. After rearrangement, the mapping is no longer coherent (Fig.6 (4)-(5)). The system compensates the imbalance of computing load which was beyond the programmer's control along with this process.

8 Conclusion

We have been developing the Multi-PSI system, a parallel software research and development tool. Multi-PSI contains several PSI CPUs and dedicated lattice network. Parallel software research is very important to realize a large-scale parallel inference machine (PIM) system, and a good research and development tool is strongly required to promote the parallel software research.

The tool, Multi-PSI system, supports a real parallel execution environment coupled with a software development system which have rarely been seen in the parallel machine research around the world. The system is also designed to keep ease of implementation and to share the same software problems with the target PIM. This enables bootstrapping of cooperative parallel software research and PIM architecture research.

We are studying and developping such issues as languages, operating systems, progam development system, basic research like load balancing and algorithms, and application programs using the Multi-PSI system.

The most important theme for the knowledge processing parallel computer system is to find a dynamic load balancing method which takes account of the communication locality between processing elements. Parallel algorithms, which extract computing parallelism and maintain communication locality, are also important. A user language is also required which can control load balancing and guide the programming style of large parallel application programs. These research themes are seen as very long range studies, whereas languages, operating system and program development system are essential not only for PIM but also for a research tool development.

We are developing two Multi-PSI systems, Multi-PSI version 1 (6 to 8 PEs) and version 2 (64 PEs max.). An overview of the Multi-PSI-V1 system and its network, and an example of how the dynamic load balancing method works were presented, along with brief accounts of each research item, to clarify our image of the goal of this research.

Multi-PSI-V1 currently contains six PSI machines and a lattice network, with packet routing functions and a speed of approximately 500K bytes/sec. The language KL1 is based on GHC, and the language processing system is written in ESP, the system description language of PSI. The hardware system and language processing system have already been completed, and they are currently being tested. Small parallel programs like the eight queens problem have started to run. Network design and machine language design for Multi-PSI version 2 are proceeding concurrently. Finally, processing elements for Multi-PSI-V2, to be also used for the next version of PSI (PSI-II), are now being manufactured.

Bootstrapping of the parallel software research utilizing Multi-PSI V1 and V2, and long range bootstrapping of parallel software research and PIM research have just started.

Acknowledgements

Valuable suggestions for the research strategy are given by Dr. Shunichi Uchida, chief of the fourth laboratory. A load balancing method shown in Section 7 is an idea of Dr. Takashi Chikayama. Research and development are carryed out by members of fourth and first labolatories of ICOT and collaborating companies.

References

[1] T. Chikayama. Unique features of ESP. In *Proceedings of FGCS'84*, ICOT, 1984.

[2] A. Goto and S. Uchida. Current Research Status of PIM: Parallel Inference Machine. TM-140, ICOT, 1985.

[3] A. Goto and S. Uchida. Toward a High Performance Parallel Inference Machine –The Intermediate Stage Plan of PIM– TR-201, ICOT, 1986.

[4] K. Murakami, K. Kakuta, R. Onai, and N. Ito. Research on parallel machine architecture for Fifth-Generation Computer Systems. *IEEE Computer*, vol.18, no.6, June 1985.

[5] S. Takagi et al. Overall design of SIMPOS. In *Proceedings of the Second International Conference on Logic Programming*, Uppsala, 1984.

[6] K. Taki et al. Hardware design and implementation of the personal sequential inference machine (PSI). In *Proceedings of FGCS'84*. Also in TR-075, ICOT, 1984.

[7] K. Ueda. Guarded Horn Clauses. TR-103, ICOT, 1985. Also in *Proc. Logic Programming Conf. '85*, E. Wada(Ed), Lecture Note in Computer Science 221, Springer-Verlag, 1986, pp.168-179.

[8] K. Ueda. Guarded Horn Clauses. *Doctorial thesis*, Information Engineering Course, Faculty of Engineering, Univ. of Tokyo.

[9] M. Yokota, A. Yamamoto, K. Taki, H. Nishikawa, and S. Uchida. The Design and Implementation of a Personal Sequential Inference Machine: PSI. TR-045, ICOT, 1984. Also in*New Generation Computing*, Vol.1 No.2, 1984.