ICOT Technical Report: TR-196

TR-196

Toward the Parallel Inference Machine

by
S. Uchida

August, 1986

Toward the parallel inference machine

Shunichi UCHIDA

Fourth Research Laboratory
Institute for New Generation Computer Technology
Mita-kokusai Bldg. 21F, 1-4-28 Mita, Minatoku, Tokyo, JAPAN

Abstract
This paper describes an approach to the parallel inference machine (PIM) which provides a parallel programming environment to support AI applications. The importance of parallel software systems, especially parallel operating systems which control job allocation and resource management, has been recognized by many architecture researchers. However, research on these software systems is still in immatured status. This causes many problems for parallel inference machine research. In Japan's fifth generation computer systems project, parallel software research is now being emphasized and plans are conducted more systematically. The work is being pursued with one eye on parallel hardware research using a dedicated tool called a multi-PSI system consisting of several personal sequential inference machines. This paper describes research and development plans for the parallel inference machine in conjunction with the parallel software research.

1. Introduction

Research on parallel machines has a long history. Many parallel machines have been proposed and actually developed, but, they have been confined to only a small part of computer applications, mainly as special purpose machines. The reason for this is very obvious. Research on parallel software systems have not yet matured enough to make parallel machines sufficiently programmable for sophisticated applications.

Requirements for extremely powerful machines have increased recently in areas like AI. In most AI applications, computing power requirement is crucial. The power of computers often limits the scale of AI systems built on them. Furthermore, recent research on various knowledge representations has made it clearer that parallel computational models are suited to represent knowldge. Parallel machines are expected to be the best solution to fulfil these requirements.

Parallel machines for AI applications of course have to be powerful and they are also required to support knowledge programming environments where large scale AI systems can be efficiently developed and executed. The machines must be general purpose, with parallel operating system. This indicates the necessity of parallel machine research putting more weight on parallel software research, unsolved with more challenging research topics increasing many unknown problems. Vast application fields of AI will gurantee the value of conducting this difficult research.

In Japan's fifth genration computer (FGCS) project, parallel machine research is being conducted as one of the major research and development targets. The FGCS project aims at the research and development of new computer technology for knowledge information processing that will be required in the 1990's. Knowledge information processing is considered as a subset of AI technology which will be realized in the next decade.

In this project, knowledge information processing systems (KIPS) will have logical inference mechanisms using knowledge bases as their central functions. Its target computer system is defined as a parallel computer system having parallel software systems based on logic programming. This project was started in April, 1982, as one of the Japan's national projects. The work is being carried out by the institute for new generation computer technology (ICOT) and eight major computer manufacturers in cooperation with many people from universities and research institutes in Japan.

The project contains many research and development items on software and hardware systems. Software systems research includes such items as natural language understanding, program verification and synthesis, and expert systems. In addition to these systems, research on knowledge representations, knowledge acquisition, and knowledge base management is also being conducted as the bases for these systems.

Hardware and architecture research include such research items as parallel inference machines (PIM) and knowledge base machines (KBM). PIM research aims at the research and development of highly parallel machines supporting parallel logic programming languages and parallel programming environments.

In addition to these research and development items, development support systems are also being developed. One of these systems is a personal sequential inference machine (PSI) a logic programming workstation to provide software researchers with an efficient logic programming environment. A new machine language (KL0) and a system description language (ESP) were designed for this machine. ESP has a modularization mechnism based on the object-oriented concept and is used for writing PSI's programming and operating system (SIMPOS). Currently more than sixty PSI machines have already been distributed to researchers and are being used as main research tools.

In the past five years, PIM research was devoted to determining feasible architectures to execute logic programming languages in parallel, using reduction and dataflow mechanisms. Several software and hardware simulators were developed and used to collect data to evaluate various possible architectures of PIM.

In parallel with this PIM architecture reserach, parallel logic programming languages were also studied. This language research aimed at the design of a new parilel logic programming language with features appropriate to describe a parallel operating system and simple and efficient for hardware implementation. Starting from the studies on such languages as Concurrent Prolog and PARLOG, a new language named Gurded Horn Clauses (GHC) was developed as the base of the project's parallel language, Kernel Language Ver. 1 (KL1).

While these parallel languages and architectures are studied, AI software research such as natural language understading systems and CAD systems is also progressing. Some of them have been expermentally implemented on PSI using ESP. The researchers working these software systems have learned a lot about the the limitations of implementable functions on currently available machines including PSI machines. They feel that current workstations are too weak in computing power to implement expert systems having more than 1,000 rules. In natural languge understsnding, rough estimates of computing power have been made which indicate that machines 1,000 times faster than currently available will be required for sufficient context analysis in machine translation systems. They feel that they will need machines 100,000,000 times faster to perform sophisticated discourse understanding. For their knowledge representaion language, they have designed a new language, CIL, which is based on an event-driven type, parallel model. They are developing a pilot system called DUALS for discourse understanding using CIL. The progress of theis AI software research is greatly encouraging parallel machine research.

In addition to these requirements, PIM research in the past five years strongly indicated the importance of the parallel software research on language processors and operating systems. Effective combination of parallel software and hadware is essential to build the practical PIMs for AI applications.

The PIM research plan has been extended to augment parallel software research in more systematic way than in the past, providing more researchers and tools. The goal of this parallel software research is to develop an operating system for PIM called PIMOS. Currently, the main functions of PIMOS are considered to be the controls of job allocation, resource management and monitoring of parallel jobs. PIMOS must be written in KL1. The KL1 language processor is also included in PIMOS, however, the programming environment is not included. It is planned to build programming environment on a PSI machine.

Experimental construction of many PIMOS software modules is essential to perform the PIMOS research effectively making many feedbacks to parallel hardware research. For this purpose, a multi-PSI system is now being developed to provided software researchers with a realistic and stable environment. On the multi-PSI systems, such problems as job allocation, resource management and parallel job monitoring will be studied in the form of PIMOS development.

The goal of the hardware research is to develop a PIM consisting of 100 processing elements (100-PE PIM). Many problems remain to be solved, for example, the design of the high-speed processing element (PE) making full use of static optimization by compiler, garbage collection methods for parallel environments, and an interconnection mechanism including the clustering of PEs.

An approach to the PIM realizing an effective combination of parallel software and hardware systems is described in this paper.

2. PIM reserch and development guideline

To build PIMs which can flexiblly support AI applications, effective combination of parallel software and hardware systems is very important. In the ultimate PIM, all the hierarchical layers of the software and hardware systems must be built based on parallel computational models. These layers may be divided as follows.

L1) Application software

Natural language understanding systems and expert systems are representative application software included in this lanyer. From the architectural view point, knowledge programming systems including knowledge reperesentation languages and related software tools may be included in this layer. In this layer, research on parallel algorithms and parallel programming paradigms, etc. is very important.

L2) System software

Programming systems including such software modules as editors and debuggers are included in this layer. These modules must be written in a parallel programming language and need man-machine interface. Operating systems must be provided as a lower sublayer of this system software. Generally speaking, one of the main functions of operating systems is to map given parallel jobs to parallel hardware systems. This mapping from the model of a given job to the model of the hardware system is the key problem in realizing flexible parallel machines. This mapping function is to be implemented in software modules for job allocation and resource allocation.

L3) Langage

Languages are obviously important because they define the interface between software and hardware systems. Languages discussed here may be classified into at least two sublayers, namely, system description languages and machine languages.

System description languages must have modularization functions to build complex software systems such as operating systems. The object-oriented concept, which uses class and inheritance mechanisms is very efficient and expected to be used not only for sequential languages but also for parallel languages without any great loss in processing speed. Operating systems for parallel inference machines are complex and must be written in logic or functional languages which have efficient modularizaton functions.

For system description languages for parallel machines, one additianal function must be considered to permit programmers to specify the way of dividing one job into several subjobs to be processed in parallel. It is ideal that operaing systems and hardware systems can automatically divide one given job into several parallel subjobs well enough to exploit maximum prallelism, however, it must be difficult especially for most AI applications.

The practical way is to persuade programmers to estimate the behavior of their jobs and explicitly specify an estimated optimal way of dividing their jobs into parallel subjobs. However, this specified way of division may be incomplete in most cases. Thus, operating systems and hardware systems must compensate for this specified way of division depending on the dynamic behavior of the jobs and the load balancing of the PEs. In some cases, system programmers must have the capability to specify job division and job allocation explicitly to control load balancing.

Hardware people insist that machine languages be simple so that firmware and hardware systems can be simple and fast. Recently, static optimization techiques by compilers have advanced especially in such languages as Prolog and Smalltalk. For logic programming languages, this optimization is quite effective for deterministic parts of programs. This must be considered also for parallel inference machines. However, for nondeteministic parts and complex unification parts, this optimization is not effective and the execution speed greatly varies depending on the characteristic of programs. More firmware and hardware supports are necessary to attain higher speed for these parts Machine languages must be designed considering the trade-offs described above.

L4) Firmware

Firmware is often used to implement high level language machines like Lisp and Prolog machines. In these machines, machine language interpreters are inplemented in firmware. Firmware implementation is flexible and thus suitable to implement experimental machines. In parallel inference machines, firmware implementation is adequate especially to implement some of machine language primitives the specifications of which are not completely fixed. In the case of parallel machine languages, management of parallel processes have to be mainly implemented in firmware to attain higher speed. Conversion between local names and global names for inter-processor communications and garbage collection are also suitable for implemention using firmware. Micro diagnosis is also effective for maintenance.

L5) Hardware

Hardware for parallel inference machines may be roughly divided into two parts, namely, processing elements (PE) and inter-connection mechanisms. For the implementation of PE, there are several alternatives for architectures such as sequential based tag architecture and full dataflow based architecture. For the inter-connection mechanisms, there are also many alternatives such as a shared memory with parallel cache and a variety of packet switching networks.

One of the important decision factors may be the amount of hardware components to implement PEs. To encourage parallel software research, the greater number of PEs is important as well as the higher processing power of each PE.

Given higher priority for harmonized combination of software and hardware systems, the hardware must be designed to be simple making full use of static optimization techniques. However, it is also required to execute such functions so quickly that static optimization cannot improve processing speed. Examples of these functions are the process management and the inter-PE communications, such as the conversion of names and interruption handling. Their hardware supports necessarily increase the amount of hardware.

In addition to the above conditions, a stable and maintenable implementation is crutial to make the machines available for larger scale software development including the development of operating systems and large evaluation programs.

L6) Device

The trade-off between the number of PEs and the processing power of each PE heavily depend on device technology. Currently, CMOS is considered the most suitable device for logic circuits. However, there are still several alternatives such as gate arrays, standard cells and custom LSIs. They differ in such paremeters circuit density, design and production cost, and turn-around time.

The ideal way of promoting parallel machine research is to proceed on all these levels, but it is not practical because software research and hardware research are mutually related to each other.

Generally speaking, less practical software research has been done on parallel processing than on hardware. This is mainly because software researchers have never been provided with attractive parallel hardware environments stable and powerful enough to build large practical parallel software systems like operating systems.

The lack of practical software research and the experience of running practical parallel programs are imposing difficulty in dividing the required functions of PE into software functions and hardware functions. This problem happens in the design of a machine language.

One of practical approaches is to design a pure parallel language at first. This language must have the features required of both a system description language and a machine language. The design of both of these languages involves experimental construction of software and hardware modules to evaluate the design. This evaluation work is essential and often needs large scale experimental software and hardware systems as tools. The research and development of parallel inference machines may be regarded as the repitition of the design and evaluation cycle.

This repetition must be started on a small scale and proceed gradually to larger scale software and hardware systems. However, at the beginning, software researchers must be provided with parallel hardware environments which can hopefully attract them. The larger scale software experiments will produce more valuable information about the behavior of parallel programs. This information is essential to design better parallel inference machines.

3. The approach to PIM in FGCS project

3.1 Outline of PIM research plan

The FGCS project was planned as a ten-year project. It is divided into three stages, namely the initial stage (1982-1984), the intermediate stage (1985-1988) and the final stage (1989-1991). In each stage, research goals are roughly determined for each research and development item.

For PIM, the initial stage goal is to study basic mechanisms necessary to organize PIMs. The intermediate stage goal is to build a medium scale experimental PIM which consists of about 100 PEs. The final satge goal is to build a large scale experimental PIM which includes about 1,000 PEs. The processing speed of the final stage goal is expected to be around 100 MLIPS.

The research goal for the software system for PIM was not explicitly determined because it was too dificult for us to estimate the progress of the parallel software research when we planned FGCS project. At that time, we simply imagined that the hardware system of the final PIM could include most functions which we are now thinking that the software system, PIMOS, must have. The research and development items related to PIMOS and KL1 were recently added as a new part of the intermediate stage plan.

The research results of PIM and its related items in the initial stage are summurized in addition to their current status. They are PIM itself, kernel language (KL), and the sequential inference machine (SIM).

3.2 PIM research in the initial stage

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

For dataflow, we designed a machine model which was named PIM-D. PIM-D executes logic programs in a goal-driven manner: the execution of a clause is initiated when a goal is given and it returns the results to the goal. In this execution, the PIM-D can exploit OR and AND prallelism as well as low-level parallelism in unification.

A hardware simulator of the PIM-D consists of 16 PEs and 15 structure memory (SM) modules, connected by an hierarchical bus network. Each PE and SM is made of bit-sliced microprogrammable processors (Am 2900 series) and TTL ICs.

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

For reduction, we designed a machine model which was named PIM-R. Logic programs generate several pieces of resolvent from a body goal in a clause. This can be regarded as a reduction process. The PIM-R consists of two types of modules, inference modules and structure memory modules with networks connecting them. In each inference module, logic programs are executed based on the reduction mechanism.

A hardware simulator and two software simulators were built for the PIM-R. The hardware simulator consists of of 16 PEs (m68K boards) connected by a common bus with a shared memory.

In addition to these two models, we built an experimental hardware system to examine a hardware support mechanism for job division and allocation in a multiprocessor environment. This mechanism was named the Kabu-wake method in which idle processors issue requests for jobs to the busy processors. If one processor requests a job from another processor, it splits up its own job and passes one of the alternative clauses to the other processor. Then, they perform OR parallel execution of logic programs.

The experimental system consists of 16 PEs (m68K based computers) connected by two types of networks. One network is a high throughput swithing network for transferring a split job and the other is a ring network for job requesting packets.

We obtained the following conclusions through the software experiments using these simulators and the experimental system.

1) Logic progrmming is suitable for parallel processing. The programs can be processed in parallel if they contain parallelism in them.
2) The machine language must be simple to obtain a fast and compact PE.
3) A stream-based logic programming language like GHC is adequate for the base of the machine language.
4) Static optimization techniques must be introduced to make PEs faster and smaller.
5) Various hardware supports for inter-process communications are essential.
6) The total hardware system of PIM must be compact for stable implementation.

This conclusion implies the importance of parallel software research to make the hardware functions of PIM clearer and simpler.

3.3 Design of Kernel Languages

1) Design of KL0

The research on the kernel language (KLn, n=0, 1 or 2) was started for the design of KL0 which was the machine language for PSI. KL0 was designed in almost the same level as DEC-10 Prolog adding several control promitives and new data types.

A new language, ESP (Extended Self-contained Prolog) was designed for the system description language coresponding to KL0 adding the object-oriented modularization function using class and multiple inheritance mechanisms. The effectiveness of ESP, especially its modularization function has been proved in writing SIMPOS (PSI's programming and operating system). Software productivity has been greatly improved by ESP and its programming environment in SIMPOS.

In the intermediate stage, KL0 has been redesigned for the improvement of PSI to incorporate static optimization techniques more effectively by a compiler. The new KL0 is based on the abstract prolog machine instruction set proposed by D.H.D. Warren, however, many functions are added to keep the compatibility to the previous KL0. The introduction of static optimization is expected to improve the execution speed of the new PSI (PSI-II) two to three times. This language design experience and usage has made a great contribution to the reseach and development of KL1.

2) Design of KL1

Research on KL1 was begun with the studies of such languages as Concurrent Prolog and PARLOG. We intended primarily to design the core language from which we could design the machine language and also the system description language. From the viewpoint of hardware, this core language is required to be as simple as possible. From the viewpoint of software, it must be clean and powerful to be able to introduce new programmig concepts and paradigms for describing operating systems and application programs.

Through discussions with researchers from abroad and experimental construction of language interpreters and sample programs, we designed a new language, GHC (Gurded Horn Clauses). It is a stream-base AND parallel language and fulfils most of the software and hardware requirements. Main features of GHC are summarized as follows:

For description of parallel processing;
1) Dynamic inter-process communications via logical variables
2) Flexibility in the size of processing granularity
3) Simplicity of description

As a logic programming language;
4) Pattern driven clause selection
5) Single assignment via logical variables

In the intermediate stage, we started to consider the detailed structure of KL1. Currently KL1 is divided into three layers, namely a machine language (KL1-b), a core language (KL1-c) a pragma (KL1-p), and a user language (KL1-u).

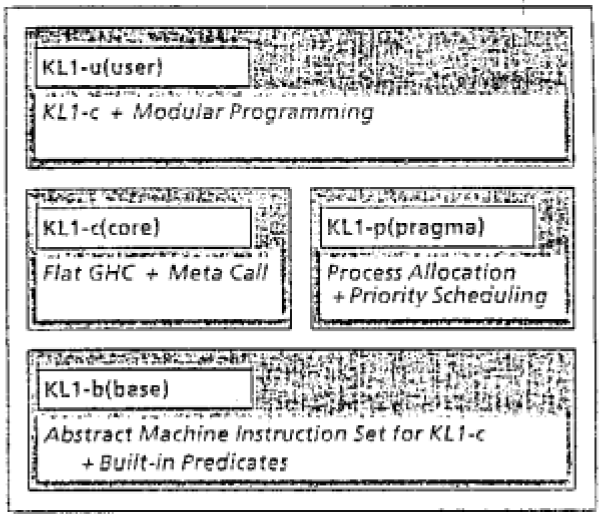
KL1-b is the machine language of PIM. Its design is almost the same as the design of PIM machine architecture, especially, the design of the PE. The execution mechanism for GHC is now being studied for an experimental interpreter in C. Optimization by compiler is also being considered.

KL1-c is the core language of KL1, and its language features determine the principles of KL1 itself. KL1-c is now being designed based on GHC and extended so that we can describe important functions of the operating system, PIMOS, such as interrupt handling and resource management.

KL1-p is a set of declarations to specify the way of dividing a job into parallel subjobs, and to specify the estimated amount of load for each subjob.

KL1-u is a user language used to describe the operaing system and users programs. KL1-u must have modularization functions and we are working on introducing the object-oriented modular structure.

The evaluation of these language specifications must be done by actually writing many sample programs. This work is currently done using PSIs and VAXs. Parallel exection environments are also desired and the multi-PSI system will be used for this purpose.



KL1 Language System

3.4 Development of SIM

In the initial stage, we developed the sequential inference machine (SIM) as a software development tool. In the development of SIM, we built two types of machines. One is named PSI and the other is named CHI. The programming and operating system, SIMPOS, was also developed on PSI. It is written in ESP and its current size is about 200K lines in ESP.

1) Personal sequential inference machine (PSI)

PSI is a personal workstation run under SIMPOS and provides researchers with an execution speed of about 30KLIPS with 80MB of main memory plus a multi-window based efficient programming environment. To run logic programming languages efficiently, PSI uses a tag architecture and a hardware supported stack mechanism. KL0 is interpreted by microprogram using specialized hardware mechanism. As we intended to distribute PSIs to many researchers, we used TTL devices and made a reliable and maintainable implementation. Its cycle time is 200 ns.

The high execution speed of PSI is attained by microprogramming techniques and specialized hardware supports. The key features of the hardware supports are for dynamic data type checking, shortening memory access time, especially stack access time, and dynamic memory allocation.

2) Cooperative high-speed sequential inference machine (CHI)

CHI was developed as a high speed prolog engine connected as a back-end processor to PSI, intended as the fastest possible machine for logic programming. It introduced the low level machine instruction set proposed by David H.D. Warren, called *abstract prolog machine instruction set*, and fully utilized the static optimizing technique by its compiler. CHI was implemented using CML (Current Mode Logic) device technology and attained 100 ns in machine cycle time. It attained about 280KLIPS in *append* operation and about 200 KLIPS for usual programs.

3) Improved version of PSI and CHI

In the intermediate stage, these machines are being improved using LSIs to make smaller-size models. They are called PSI-II and CHI-II.

PSI-II has the low-level machine instruction set similar to Warren's set, but, it has more optimized instructions. The design of PSI-II employs the following.

1) The low-level machine instruction set
2) Equipment of argument registers
3) Using 3 stacks, local, global, and trail stack
4) Structure copying for the structured data

Specifically, the new PSI's CPU will be used as element processors of the multi-PSI system. The execution speeed of PSI-II is expected to be improved to 150 KLIPS. We expect that this improvement will be effective to implement the interpreter of a parallel logic programming language such as GHC.

In the design of CHI-II, we intended to make it much smaller than the previous model because the implementation using CML device made the size of the previous model too large to use it as a practical programming tool. CHI-II uses CMOS VLSIs and 1 Mbit memory chips to obtain the size like a desk-side locker keeping almost the same performance as the previous model.

Through the development of SIM, we learned a lot of things about the hardware and software implementations for logic programming. They are effectively being applied for the research and development of PIM.

a) Effective implementation methods of the tag architecture for logic programming.
b) The static optimization techniques and design methods of the efficient machine instruction set.
c) Dynamic behavior of many logic programs through the evaluation of PSI's micro interpreter.
d) System programming techniques using logic programming through the development of SIMPOS.
e) Great merits of using logic programming for writing complex programs like operating systems.

f) Great contribution of the object-oriented modularization to software productivity and reliability through the use of ESP for SIMPOS and other various applications.

3.5 PIM research in the intermediate stage

Considering about the research results in the initial stage research and development, we made the more detailed plan for PIM research in the intermediate stage.

1) The intermediate stage goal for PIM

The research goal of PIM is redefined as follows:

1) The experimental hardware contains about 100 PEs.
2) Machine language is KL1-b based on GHC.
3) Target processign speed is 2 to 5 MLIPS.
4) The hardware systm must be stable and maintainable enough to support PIMOS.
5) The experimental PIM operating system is developed in parallel with the hardware development.

This new goal puts more weight on the effective combination of the hardware and software. The hardware system is required to be simple, fast and stable. The PE is based on a sequential execution mechanism and optimized for the execution of KL1-b. This experimental hardware system is informaly called PIM-I.

For the PIMOS development, we need a stable parallel hardware environment as its development tool. To encourage software researchers, this hardware environment is desired to be fast, flexible and stable as much as possible and also must be provided as soon as possible. The multi-PSI system is almost the best solution in this project to fulfil these requirements.

2) Design of the intermediate stage PIM, PIM-I

The research of PIM-I was begun with the design of the machine language, KL1-b. The execution mechanism of KL1-b is also being designed writing experimental interpreters in ESP and C. In this functional design phase, we are studying following items:

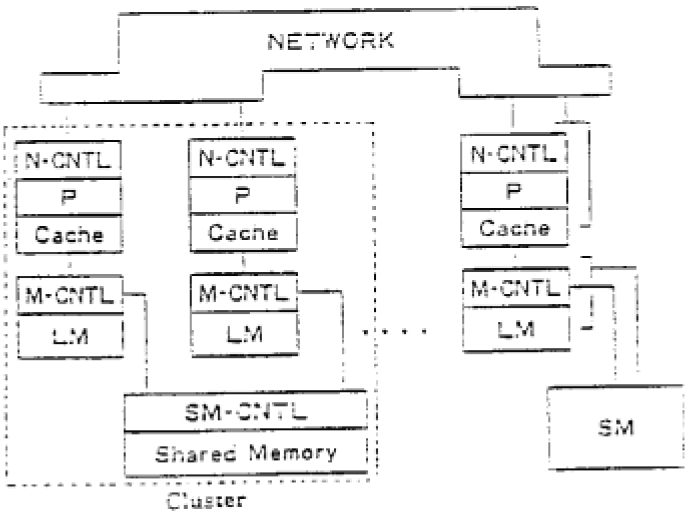
1) For synchronization and scheduling: internal structure of goals, priority scheduling queues and goal trees to manage logical goal relationship; suspend-hook mechanism; and efficient context switching mechansim.
2) For communication: shared buffer communication and packet communication; and efficient context switching mechanisms for remote data access.
3) For stream: CDR-coding and special primitives.
4) For fine grained activities: optimizing compilers; improved bounded depth-first scheduling; and a low level instruction set based on sequential execution.
5) For large grained activities: load distribution primitives.

The design of the machine language and its execution mechanism has to be proceed with the functional design of the PE and the connection mechanism. We are now intending to introduce a cluster concept and a shared memory to reduce inter-PE communication delay. Concerning to the hardware of PIM-I, we are studying following items:

1) Functions of the cluster: logical cluster and address transformation; physical cluster and its implementation; and inter-cluster network and cluster controler.

2) For memory system design: parallel cache mechanisms; proper usage of local and shared (global) memories; and specialized memory mechanisms dediacted to KL1-b.
3) For the design of PE: implementaions of tag architectures for KL1-b; efficient context switching mechanisms; and communication hardware and interrupt handling mechanisms.

To attain the target processing speed for PIM-I, we are trying to make the PE at least faster than PSI-II for KL1-b and also introduce fast inter-PE comunication mechanisms such as a global shared memory so that the overhead in the communication can be reduced by a variety of optimization techniques to be developed by software researchers.



Organization of PIM-I

3.6 PIMOS research and the multi-PSI system

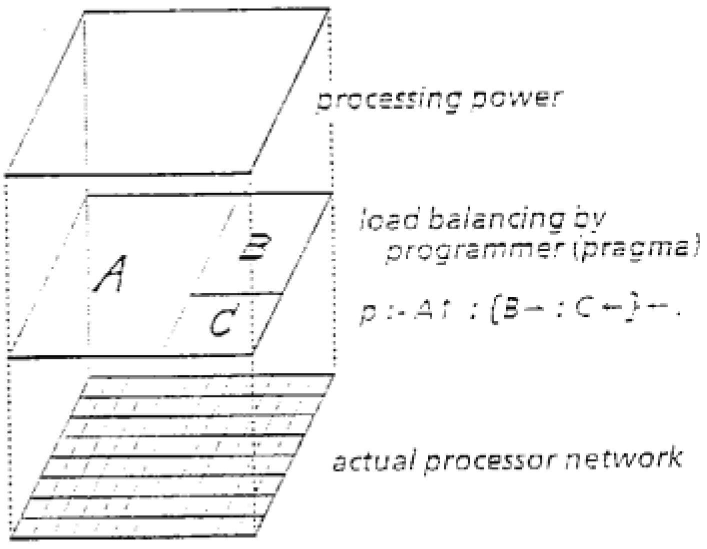
1) PIMOS research

The ultimate goal of the parallel software research may be to develop various useful software technology on parallel computational models and actually construct various software systems on prallel machines. As a small step toward this goal, we aims to build an experimental operating system, PIMOS which will be operational on PIM-I. The primary goal of PIMOS research is to build the experimental software system for hardware resource management of PIM-I. This software system may correspond to the kernel and supervisor layers of usual operating systems.

The research is begin with the studies on such items:
1) Control methods for job allocation and locality of communication between PEs.
2) Memory management and GC for distributed environments.
3) Program code management and distribution methods.
4) Methods to handle input/output and interrupt.
5) Debugging and monitoring of parallel programs.

These research items have long been realized as important but difficult research themes in parllel processing. We start this research with the implementation of KL1 interpreter on the multi-PSI system.

For the job allocation problem, we have decided to have a policy of persuading programmers to explicitly specify the way of dividing their jobs and the amount of load for each divided jobs. We are now trying to introduce a two-dimensional processing power plane (PPP) model where a programmer assumes that processing power is uniformely distributed on a two-dimensional plane. The size of the plane area corresponds to the amount of processing power. The distance between two points on the plane corresponds to the cost of the communication. By specifying the way of dividing this plane and the correspondence between the divided area and the divided job, the programmer can represent his or her suggestion about the job allocation. The specification is written using KL1-p (pragma). It is unlikely that this suggestion is completly correct. Then dynamic reallocation of divided jobs must be made by PIMOS and PIM-I to attain better performance. This idea of job allocation has not been examined deeply, however, it may be a suitable start point of this research. Many ideas of this kind will be studied actually writing programs on the multi-PSI system.



Process Allocation Strategy (Load Balancing)

2) Development of the multi-PSI system

The multi-PSI system is developed for parallel software experiments. It is desirable to be provided as soon as possible to promote PIMOS research. We plan to build two versions of the multi-PSI system, namely, MPSI-V1 and MPSI-V2.

MPSI-V1 uses 6 to 8 current PSIs (PSI-I) as its PEs. They are connected with a two-dimensional mesh type network. Each node of this network has five input/output channels and a simple routing mechanism. One of these channels is connected to the PSI's internal bus. Its basic data transfer is controlled by PSI's microprogram. Its data transfer rate is about 500 KByte/sec for each channel. The routing mechanism of each node passes each packet one channel to another without interrpting the PSI connected to that node if the destination of the packet is not that PSI. On MPSI-V1, the interpreter of KL1, actually GHC, is written in ESP using SIMPOS. Then, the execution speed of GHC will be slow, however, the parallel interpreter using communication facility can be build and evaluated. MPSI-V1 will be avilable in August, 1986.

MPSI-V2 will use 16 to 64 CPUs of PSI-II as its PEs. The connection network is almost similar to that of MPSI-V1, however, its size will be smaller using LSIs (Gate arrays) and its transfer rate will be improved. On MPSI-V2, the interpreter of KL1 will be written in microprogram so that faster execution speed can be attained. We are expecting that it can attain around 100 KLIPS if given programs do not cause suspentions often. Thus, larger scale software experiments will be possible on MPSI-V2 including the building of PIMOS. MPSI-V2 is planed to be available around the end of 1987.

4. Concluding remarks

In this paper, the effective combination of software and hardware systems on PIM is emphasized. To promote the research and development of parallel software technoigy, the intimate cooperation of software and hardware people is essential. The primary role of the hardware people is to povide parallel hardware environments which can be attractive tools for the software people. In the FGCS project, we are building the multi-PSI system for providing this tool. We also intend that PIM-I will be a next version of the tools. Prallel software research has so many unknown problems that these tools have to be improved repeatedly in many years. Our current effort may be regarded as starting the first step of this repetition toward the realization of fully parallel inference machines.

Reference

[1] H. Ishibasi, et al: SIMPOS: Sequential Inference Machine Programming and Operationg System,— Its User Interface—, to apper in FJCC'86, Nov. 1986.

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

[3] K. Ueda: Gurded Horn Clauses, Lecture Notes in Computer Science 221, Spring-Verlag, 1986.

[4] T. Chikayama: Load Balancing in Very Large Scaled Multi-Processor Systems, Proc. of fourth Japanese and Swedish Workshop on Fifth Generation Computer Systems, Jul. 1986., also to appear ICOT TR.

[5] E. Shapiro: A Subset of Concurrent Prolog and Its interpreter, ICOT TR-003, 1983.

[6] K. Clark and S. Gregory: PARLOG: Parallel Programming in Logic, Research report DOC 84/4. Dept. of Computing, Imperial College, London.

[7] D.H.D. Warren: An Abstract Prolog Instruction Set, Tech. Note 309. AI Center, SRI International, 1983.

[8] S. Uchida: Sequential Infernce Machine: SIM - Progress Report, Proc. of FGCS'84, Nov. 1984.

[9] T. Chikayama: Unique features of ESP, Proc. of FGCS, Nov. 1084.

[10] R. Nakazaki, et al: Design of A High-speed Prolog Machine (HPM), Proc. of the 12th ISCA, pp.191-197, Jun. 1985.

[11] K. Nakajima, et al: Evaluation of PSI micro-interpreter, Proc. of COMPCON'86, Mar. 1986.