PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Summary of the Parallel Inference Machine and its Basic Software

Shunichi Uchida

Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
uchida@icot.or.jp

Abstract

This paper aims at a concise introduction to the PIM and its basic software, including the overall framework of the project. Now an FGCS prototype system is under development. Its core is called a parallel inference system which includes a parallel inference machine, PIM, and its operating system, PIMOS. The PIM includes five hardware modules containing about 1,000 element processors in total. On the parallel inference system, there is a knowledge base management system (KBMS). The PIMOS and KBMS make a software layer called a basic software of the prototype system. These systems are already being run on the PIM. On these systems, a higher-level software layer is being developed. It is called a knowledge programming software. This is to be used as a tool for more powerful inference and knowledge processing. It contains language processors for constraint logic programming languages, parallel theorem provers and natural language processing systems. Several experimental application programs are also being developed for both general evaluation of the PIM and the exploration of new application fields for knowledge processing. These achievements with the PIM and its basic software easily surpass the research targets set up at the beginning of the project.

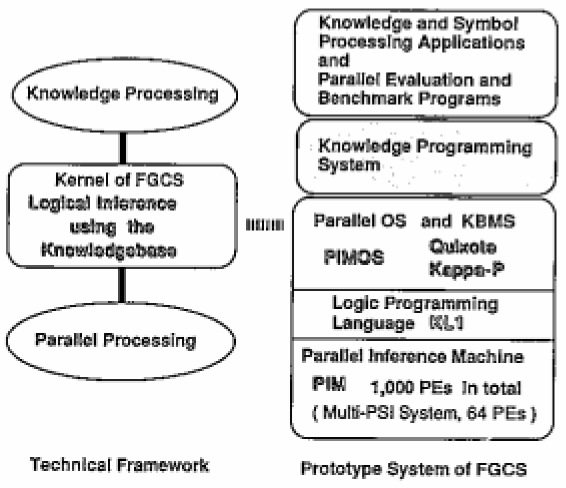

Knowledge Processing
Kernel of FGCS
Logical Inference using the Knowledgebase
Parallel Processing
Technical Framework
Knowledge and Symbol Processing Applications and Parallel Evaluation and Benchmark Programs
Knowledge Programming System
Parallel OS and KBMS
PIMOS
Quixote
Kappa-P
Logic Programming Language KL1
Parallel Inference Machine
PIM 1,000 PEs in total
(Multi-PSI System, 64 PEs)
Prototype System of FGCS

Figure 1: Framework of FGCS Project

1 Introduction

Since the fifth generation computer systems project (FGCS) was started in June, 1982, 10 years have passed, and the project is approaching its goal. This project assumed that "logic" was the theoretical backbone of future knowledge information processing, and adapted logic programming as the kernel programming language of fifth generation computer systems. In addition to the adaptation of logic programming, highly parallel processing for symbolic computation was considered indispensable for implementing practical knowledge information processing systems. Thus, the project aimed to create a new computer technology combining knowledge processing with parallel processing using logic programming.

Now an FGCS prototype system is under development. This system integrates the major research achievements of these 10 years so that they can be evaluated and demonstrated. Its core is called **a parallel inference system** which includes a parallel inference machine, PIM, and its operating system, PIMOS. The PIM includes five hardware modules containing about 1,000 element processors in total. It also includes a language processor for a parallel logic language, KL1.

On the parallel inference system, there is a **knowledge base management system (KBMS)**. The KBMS includes a database management system (DBMS), Kappa-P, as its lower layer. The KBMS provides a knowledge representation language, Quixote, based on the deductive (and) object-oriented database. The PIMOS and KBMS make a software layer called a **basic software** of the prototype system. These systems are already being run on the PIM. The PIM and basic software are now being used as a new research platform for building experimental parallel application programs. They are the most complete of their kind in the world.

On this platform, a higher-level software layer is being developed. This is to be used as a tool for more powerful inference and knowledge processing. It contains language processors for constraint logic programming languages, parallel theorem provers, natural language processing systems, and so on. These software systems all include the most advanced knowledge processing techniques, and are at the leading edge of advanced software science.

Several experimental application programs are also being developed for both general evaluation of the PIM and the exploration of new application fields for knowledge processing. These programs include a legal reasoning system, genetic information processing systems, and VLSI CAD systems. They are now operating on the parallel inference system, and indicate that parallel processing of knowledge processing applications is very effective in shortening processing time and in widening the scope of applications. However, they also indicate that more research should be made into parallel algorithms and load balancing methods for symbol and knowledge processing. These achievements with the PIM and its basic software easily surpass the research targets set up at the beginning of the project.

This paper aims at a concise introduction to the PIM and its basic software, including the overall framework of the project. This project is the first Japanese national project that aimed at making a contribution to world computer science and the promotion of international collaboration. We have published our research achievements wherever possible, and distributed various programs from time to time. Through these activities, we have also been given much advice and help which was very valuable in helping us to attain our research targets. Thus, our achievements in the project are also the results of our collaboration with world researchers on logic programming, parallel processing and many related fields.

2 Research Targets and Plan

2.1 Scope of R & D

The general target of the project is the development of a new computer technology for knowledge information processing.

Having "mathematical logic" as its theoretical backbone, various research and development themes were established on software and hardware technologies focusing on knowledge and symbol processing. These themes are grouped into the following three categories:

2.1.1 Parallel inference system

The core portion of the project was the research and development of the parallel inference system which contains the PIM, a KL1 language processor, and the PIMOS. To make the goal of the project clear, a FGCS prototype system was considered a major target. This was to be build by integrating many experimental hardware and software components developed around logic programming.

The prototype system was defined as a parallel inference system which is intended to have about 1,000 element processors and attain more than 100M LIPS (Logical Inference Per Second) as its execution speed. It was also intended to have a parallel operating system, PIMOS, as part of the basic software which provides us with an efficient parallel programming environment in which we can easily develop various parallel application programs for symbol and knowledge processing, and run them efficiently. Thus, this is regarded as the development of a super computer for symbol and knowledge processing.

It was intended that overall research and development activities would be concentrated so that the major research results could be integrated into a final prototype system, step by step, over the timespan allotted to the project.

2.1.2 KBMS and knowledge programming software

Themes in this category aimed to develop a basic software technology and theory for knowledge processing.

- Knowledge representation and knowledge base management
- High-level problem solving and inference software
- Natural language processing software

These research themes were intended to create new theories and software technologies based on mathematical logic to describe various **knowledge fragments** which are parts of **"natural" knowledge bases** produced in our social systems. We also intended to store them in a computer system as components of **"artificial" knowledge bases** so that they can be used to build various intelligent systems.

To describe the knowledge fragments, a knowledge representation language has to be provided. It can be regarded as a very high-level programming language executed by a sophisticated inference mechanism which is much cleverer than the parallel inference system. Natural language processing research is intended to cover

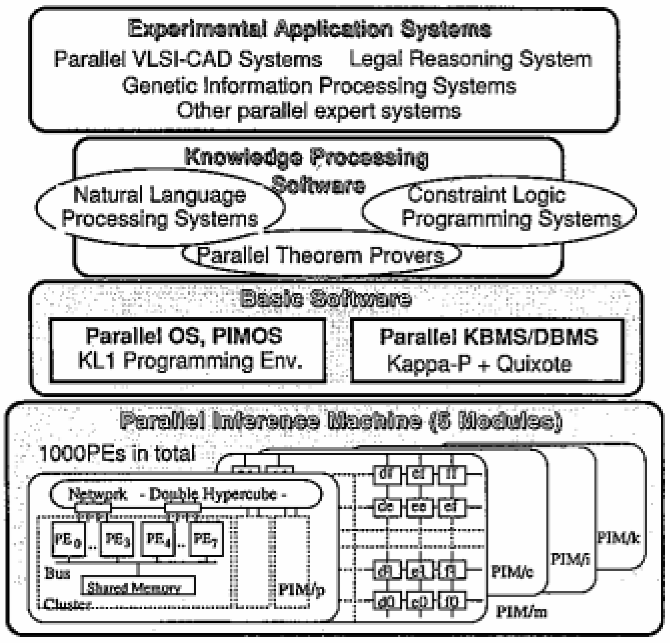

Experimental Application Systems
Parallel VLSI-CAD Systems
Legal Reasoning System
Genetic Information Processing Systems
Other parallel expert systems
Knowledge Processing Software
Natural Language Processing Systems
Constraint Logic Programming Systems
Parallel Theorem Provers
Basic Software
Parallel OS, PIMOS
KL1 Programming Env.
Parallel KBMS/DBMS
Kappa-P + Quixote
Parallel Inference Machine (5 Modules)
1000PEs in total
Network - Double Hypercube -
PE0 .. PE3 PE4 .. PE7
Bus
Shared Memory
Cluster
PIM/p
PIM/c
PIM/i
PIM/k
PIM/m

Figure 2: Organization of Prototype System

research on knowledge representation methods and such inference mechanisms, in addition to research on easy-to-use man-machine interface functions. Experimental software building for some of these research themes was done on the sequential inference machines because the level of research was so basic that computational power was not the major problem.

2.1.3 Benchmarking and evaluation systems

- Benchmarking software for the parallel inference system
- Experimental parallel application software

To carry out research on an element technology in computer science, it is essential that an experimental software system is built. Typical example problems can then be used to evaluate theories or methods invented in the progress of the research.

To establish general methods and technologies for knowledge processing, experimental systems should be developed for typical problems which need to process knowledge fragments as sets of rules and facts.

These problems can be taken from engineering systems, including machine design and the diagnosis of machine malfunction, or from social systems such as medical care, government services, and company management.

Generally, the exploitation of computer technology for knowledge processing is far behind that for scientific calculation. Recent expert systems and machine translation systems are examples of the most advanced knowledge processing systems. However, the numbers of rules and facts in their knowledge bases are several hundreds on average.

This scale of knowledge base may not be large enough to evaluate the maximum power of parallel inference system having about 1,000 element processors. Thus, research and development on large-scale application systems is necessary not only for knowledge processing research but also for the evaluation of the parallel inference system. Such application systems should be widely looked for in many new fields.

The scope of research and development in this project is very wide, however, the parallel inference system is central to the whole project. It is a very clear research target. Software research and development should also cover diverse areas in recent software technology. However, it has "logic" as the common backbone.

It was also intended that major research achievements should be integrated into one prototype system. This has made it possible for us to organize all of our research and development in a coherent way. At the beginning of the project, only the parallel inference machine was defined as a target which was described very clearly. The other research targets described above were not planned at the beginning of the project. They have been added in the middle of the intermediate stage or at the final stage.

2.2 Overall R & D plan

After three years of study and discussions on determining our major research fields and targets, the final research and development plan was determined at the end of fiscal 1981 with the budget for the first fiscal year.

At that time, practical logic programming languages had begun to be used in Europe mainly for natural language processing. The feasibility and potential of logic languages had not been recognized by many computer scientists. Thus, there was some concern that the level of language was too high to describe an operating system, and that the overhead of executing logic programs might be too large to use it for practical applications. This implies that research on logic programming was in its infancy.

Research on parallel architectures linked with high-level languages was also in its infancy. Research on dataflow architectures was the most advanced at that time. Some dataflow architecture was thought to have the potential for knowledge and symbol processing. However, its feasibility for practical applications had not yet been evaluated.

Most of the element technologies necessary to build the core of the parallel inference system were still in their infancy. We then tried to define a detailed research plan step by step for the 10-year project period. We divided the 10-year period into three stages, and defined the research to be done in each stage as follows:

- **Initial stage (3 years) :**
  -Research on potential element technologies
  -Development of research tools
- **Intermediate stage (4 years) :**
  -First selection of major element technologies for final targets
  -Experimental building of medium-scale systems
- **Final stage (3 years) :**
  -Second selection of major element technologies for final targets
  -Experimental building of a final full-scale system

At the beginning of the project, we made a detailed research and development plan only for the initial stage. We decided to make detailed plans for the intermediate and final stages at the end of the stage before, so that the plans would reflect the achievements of the previous stage. The research budget and manpower were to be decided depending on the achievements. It was likely that the project would effectively be terminated at the end of the initial stage or the intermediate stage.

3 Inference System in the Initial Stage

3.1 Personal Sequential Inference Machine (PSI-I)

To actually build the parallel inference system, especially a productive parallel programming environment which is now provided by PIMOS, we needed to develop various element technologies step by step to obtain hardware and software components. On the way toward this development, the most promising methods and technologies had to be selected from among many alternatives, followed by appropriate evaluation processes. To make this selection reliable and successful, we tried to build experimental systems which were as practical as possible.

In the initial stage, to evaluate the descriptive power and execution speed of logic languages, a personal sequential machine, PSI, was developed. This was a logic programming workstation. This development was also aimed at obtaining a common research tool for software development. The PSI was intended to attain an execution speed similar to DEC10 Prolog running on a DEC20 system, which was the fastest logic programming system in the world.

To begin with, a PSI machine language, **KL0**, was designed based on Prolog. Then a hardware system was designed for the KL0. We employed tag architecture for the hardware system. Then we designed a system description language, **ESP**, which is a logic language having a class and inheritance mechanisms to make program modules efficiently.[Chikayama 1984] ESP was used not only to write the operating system for PSI, which is named **SIMPOS**, but also to write many experimental software systems for knowledge processing research.

The development of the PSI machine and SIMPOS was successful. We were impressed by the very high software productivity of the logic language. The execution speed of the PSI was about 35K LIPS and exceeded its target. However, we realized that we could improve its architecture by using the optimization capability of a compiler more effectively. We produced about 100 PSI machines to distribute as a common research tool. This version of the PSI is called PSI-I.

In conjunction with the development of PSI-I and SIMPOS, research on parallel logic languages was actively pursued. In those days, pioneering efforts were being made on parallel logic languages such as PARLOG and Concurrent Prolog. [Clark and Gregory 1984], [Shapiro 1983] We learned much from this pioneering research, and aimed to obtain a simpler language more suited for a machine language for a parallel inference machine. Near the end of the initial stage, a new parallel logic language, **GHC** was designed. [Ueda 1986]

Table 1: Development of Inference Systems

	Sequential Inference Tech.	Parallel Inference Tech.
'82-'84 Initial Stage 前期	Sequential Logic Programming Languages, KL0 and ESP Sequential Inference Machine, PSI-I and SIMPOS, 35KLIPS for KL0	Parallel Logic Programming Languages GHC and KL1
'85-'88 Inter-mediate Stage 中期	New model of PSI, PSI-II, 330KLIPS for KL0	Experimental Model of PIM, Multi-PSI System, 5MLIPS / 64PEs for KL1 Parallel OS, PIMOS and Small Application Programs
'89-'92 Final Stage 後期	New model of PSI, PSI-III (PSI-UX), 1.4MLIPS for KL0	Prototype of FGCS, PIM, 1000 PEs total, 200MLIPS / 512PEs for KL1

3.2 Effect of PSI development on the research plan

The experience gained in the development of PSI-I and SIMPOS heavily affected the planning of the intermediate stage.

3.2.1 Efficiency in program production

One of the important questions related to logic language was the feasibility of writing an operating system which needs to describe fine detailed control mechanisms. Another was its applicability to writing large-scale programs. SIMPOS development gave us answers to these questions. The SIMPOS has a multi-window-based user interface, and consists of more than 100,000 ESP program lines. It was completed by a team of about 20 software researchers and engineers over about two years. Most of the software engineers were not familiar with logic languages at that time.

We found that logic languages have much higher productivity and maintainability than conventional von Neumann languages. This was obvious enough to convince us to describe a parallel operating system also in a logic language.

3.2.2 Execution performance

The PSI-I hardware and firmware attained about 35K LIPS. This execution speed was sufficient for most knowledge processing applications. The PSI had an 80 MB main memory. It was a very big memory compared to mainframe computers at that time. We found that this large memory and fast execution speed made a logic language a practical and highly productive tool for software prototyping.

The implementation of the PSI-I hardware required 11 printed circuit boards. As the amount of hardware became clear, we established that we could obtain an element processor for a parallel machine if we used VLSI chips for implementation.

For the KL0 language processor which was implemented in the firmware, we estimated that better optimization of object code made by the compiler would greatly improve execution speed. (Later, this optimization was made by introducing of the "WAM" code.[Warren 1983])

The PSI-I and SIMPOS proved that logic languages are a very practical and productive vehicle for complex knowledge processing applications.

4 Inference Systems in the Intermediate Stage

4.1 A parallel inference system

4.1.1 Conceptual design of KL1 and PIMOS

The most important target in the intermediate stage was a parallel implementation of a KL1 language processor, and the development of a parallel operating system, PIMOS.

The full version of GHC, was still too complex for the machine implementation. A simpler version, FGHC, was designed.[Chikayama and Kimura 1985] Finally, a practical parallel logic language, KL1, was designed based on FGHC.

The KL1 is a parallel language classified as an

AND-parallel logic programming language. Its language processor includes an automatic memory management mechanism and a dataflow process synchronization mechanism. These mechanisms were considered essential for writing and compiling large parallel programs. The first problem was whether they could be implemented efficiently. The second problem was what kind of firmware and hardware support would be possible and effective.

In addition to problems in implementing the KL1 language processor, the design of PIMOS created several important problems. The role of PIMOS is different from that of conventional operating systems. PIMOS does not need to do primary process scheduling and memory management because these tasks are performed by the language processor. It still has to perform resource management for main memory and element processors, and control the execution of user programs. However, a much more difficult role was added. It must allow a user to divide a job into parallel processable processes and distribute them to many element processors. Processor loads must be well balanced to attain better execution performance. In knowledge and symbol processing applications, the dynamic structure of a program is not regular. It is difficult to estimate the dynamic program structure. It was desirable that PIMOS could offer some support for efficient job division and load balancing problems.

These problems in the language processor and the operating system were very new, and had not been studied as practical software problems. To solve these problems, we realized that we must have appropriate parallel hardware as a platform to carry out practical software experiments using a trial and error.

4.1.2 PSI-II and Multi-PSI system

In conjunction with the development of KL1 and PIMOS, we needed to extend our research and develop new theories and software technologies for knowledge processing using logic programming. This research and development demanded improvement of PSI-I machines in such aspects as performance, memory size, cabinet size, disk capacity, and network connection.

We decided to develop a smaller and higher-performance model of PSI, to be called **PSI-II**. This was intended to provide a better workstation for use as a common tool and also to obtain an element processor for the parallel hardware to be used as a platform for parallel software development. This hardware was called a **multi-PSI system**. It was regarded as a small-scale experimental version of the PIM. As many PSI-II machines were produced, we anticipated having very stable element processors for the multi-PSI system.

The PSI-II used VLSI gate array chips for its CPU. The size of the cabinet was about one sixth that of PSI-I. Its execution speed was 330K LIPS, about 10 times faster than that of PSI-I. This improvement was attained mainly through employment of the better compiler optimization technique and improvement of its machine architecture. The main memory size was also expanded to 320 MB so that prototyping of large applications could be done quickly.

In the intermediate stage, many experimental systems were built on PSI-I and PSI-II systems for knowledge processing research. These included small-to-medium scale expert systems, a natural language discourse understanding system, constraint logic programming systems, a database management system, and so on. These systems were all implemented in the ESP language using about 300 PSI-II machines distributed to the researchers as their personal tools.

The development of the multi-PSI system was completed in the spring of 1988. It consists of 64 element processors which are connected by an 8 by 8 mesh network. One element processor is contained in three printed circuit boards. Eight element processors are contained in one cabinet. Each element processor has an 80 MB main memory. Thus, a multi-PSI was to have about 5GB memories in total. This hardware was very stable, as we had expected. We produced 6 multi-PSI systems and distributed them to main research sites.

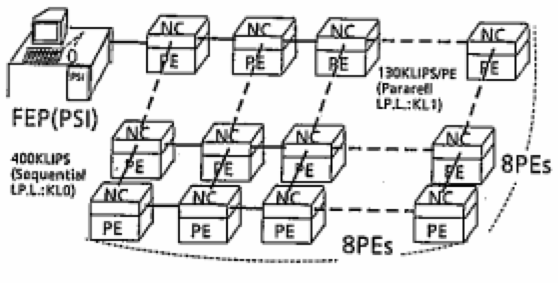
4.1.3 KL1 language processor and PIMOS

This was the first trial implementation of a distributed language processor of a parallel logic language, and a parallel operating system on real parallel hardware, used as a practical tool for parallel knowledge processing applications.

The KL1 distributed language processor was an integration of various complex functional modules such as a distributed garbage collector for loosely-coupled memories. The automatic process synchronization mechanism based on the dataflow model was also difficult to implement over the distributed element processors. Parts of these mechanisms had to be implemented combined with some PIMOS functions such as a dynamic on-demand loader for object program codes. Other important functions related to the implementation of the language processor were support functions like system debugging, system diagnostic, and system maintenance functions.

In addition to these functions for the KL1 language processor, many PIMOS functions for resource management and execution control had to be designed and implemented step by step, with repeated partial module building and evaluation.

This partial module building and evaluation was done for core parts of the KL1 language processor and PIMOS, using not only KL1 but also ESP and C languages. An appropriate balance between the functions of the language processor and the functions of PIMOS was considered. The language processor was implemented in a PSI-II firmware for the first time. It worked as a pseudo parallel simulator of KL1, and was used as a PIMOS


FEP(PSI)
PSI
NC
PE
130KLIPS/PE (Pararell LP.L.:KL1)
400KLIPS (Sequential LP.L.:KL0)
8PEs
8PEs

- Machine language: KL1-b
- Max. 64PEs and two FEPs (PSI-II) connected to LAN
- Architecture of PE:
  - Microprogram control (64 bits/word)
  - Machine cycle: 200ns, Reg.file: 64W
  - Cache: 4 KW, set associative/write-back
  - Data width: 40 bits/word
  - Memory capacity: 16MW (80MB)
- Network:
  - 2-dimensional mesh
  - 5MB/s x 2 directions/ch with 2 FIFO buffers/ch
  - Packet routing control function

Figure 3: Multi-PSI System: Main features and Appearance

development tool. It was eventually extended and transported to the multi-PSI system.

In the development of PIMOS, the first partial module building was done using the C language in a Unix environment. This system is a tiny subset of the KL1 language processor and PIMOS, and is called the PIMOS Development Support System (PDSS). It is now distributed and used for educational purposes. The first version of PIMOS was released on the PSI-II with the KL1 firmware language processor. This is called a **pseudo multi-PSI system.** It is currently used as a personal programming environment for KL1 programs.

With the KL1 language processor fully implemented in firmware, one element processor or a PSI-II attained about 150 KLIPS for a KL1 program. It is interesting to compare this speed with that for a sequential ESP program. As a PSI-II attains about 300 KLIPS for a sequential ESP program, the overhead for KL1 caused by automatic process synchronization halves the execution speed. This overhead is compensated for by efficient parallel processing. A full-scale multi-PSI system of 64 element processors could attain 5 - 10 MLIPS. This speed was considered sufficient for the building of experimental software for symbol and knowledge processing applications. On this system, simple benchmarking programs and applications such as puzzle programs, a natural language parser and a Go-game program were quickly developed. These programs and the multi-PSI system was demonstrated in FGCS'88.[Uchida et al. 1988] These proved that KL1 and PIMOS could be used as a new platform for parallel software research.

4.2 Overall design of the parallel inference system

4.2.1 Background of the design

The first question related to the design of the parallel inference system was what kind of functions must be provided for modeling and programming complex problems, and for making them run on large-scale parallel hardware.

When we started this project, research on parallel processing still tended to focus on hardware problems. The major research and development interest was in SIMD or MIMD type machines applied for picture processing or large-scale scientific calculations. Those applications were programmed in Fortran or C. Control of parallel execution of those programs, such as job division and load balancing, was performed by built-in programs or prepared subroutine libraries, and could not be done by ordinary users.

Those machines excluded most of the applications which include irregular computations and require general parallel programming languages and environments. This tendency still continues. Among these parallel machines, some dataflow machines were exceptional and had the potential to have functional languages and their general parallel programming environment.

We were confident that a general parallel programming language and environment is indispensable for writing parallel programs for large-scale symbol and knowledge processing applications, and that they must provide such functions as follows:

1. An automatic memory management mechanism for distributed memories (parallel garbage collector)
2. An automatic process synchronization mechanism based on a dataflow scheme
3. Various support mechanisms for attaining the best job division and load balancing.

The first two are to be embedded in the language processor. The last is to be provided in a parallel operating system. All of these answer the question of how to write parallel programs and map them on parallel machines.

This mapping could be made fully automatic if we limited our applications to very regular calculations and processing. However, for the applications we intend, the mapping process, which includes job division and load-balancing, should be done by programmers using the functions of the language processor and operating system.

4.2.2 A general parallel programming environment

Above mechanisms for mapping should be implemented in the following three layers:

1. A parallel hardware system consisting of element processors and inter-connection network (PIM hardware)
2. A parallel language processor consisting of run-time routines, built-in functions, compilers and so on (KL1 language processor)
3. A parallel operating system including a programming environment (PIMOS)

At the beginning of the intermediate stage, we tried to determine the roles of the hardware, the language processor and the operating system. This was really the start of development.

One idea was to aim at hardware with many functions and using high density VLSI technology, as described in early papers on dataflow machine research. It was a very challenging approach. However, we thought it too risky because changes to the logic circuits in VLSI chips would have a long turn-around time even if the rapid advance of VLSI technology was taken into account. Furthermore, we thought it would be difficult to run hundreds of sophisticated element processors for a few days to a few weeks without any hardware faults.

Implementation of the language processor and the operating system was thought to be very difficult too. As there were no prior examples, we could not make any reliable quantitative estimation of the overhead caused by these software systems. This implementation was therefore considered risky too.

Finally, we decided not to make an element processor too complex , so that our hardware engineers could provide the software researchers with a large-scale hardware platform stable enough to make the largest-scale software experiments in the world.

However, we tried to add cost-effective hardware support for KL1 to the element processor, in order to attain a higher execution speed. We employed tag architecture to support the automatic memory management mechanism as well as faster execution of KL1 programs. The automatic synchronization mechanism was to be implemented in firmware. The supports for job division and load balancing were implemented partially by the firmware as primitives of the KL1 language, but they were chiefly implemented by the operating system. In a programming environment of the operating system, we hoped to provide a semi-automatic load balancing mechanism as an ultimate research goal.

PIMOS and KL1 hide from users most of the architectural details of the element processors and network system of PIM hardware. A parallel program is modeled and programmed depending on a parallel model of an application problem and algorithms designed by a programmer. The programmer has great freedom in dividing programs because a KL1 program is basically constructed from very fine-grain processes.

As a second step, the programmer can decide the grouping of fine-grain processes in order to obtain an appropriate granularity as divided jobs, and then specify how to dispatch them to element processors using a special notation called "pragma". This two step approach in parallel programming makes it easy and productive.

We decided to implement the memory management mechanism and the synchronization mechanism mainly in the firmware. The job division and load balancing mechanism was to be implemented in the software. We decided not to implement uncertain mechanisms in the hardware.

The role of the hardware system was to provide a stable platform with enough element processors, execution speed, memory capacity, number of disks and so on. The demands made on the capacity of a cache and a main memory were much larger than those of a general purpose microprocessor of that time. The employment of tag architecture contributed to the simple implementation of the memory management mechanism and also increased the speed of KL1 program execution.

5 R & D in the final stage

5.1 Planning of the final stage

At the end of the intermediate stage, an experimental medium-scale parallel inference system consisting of the multi-PSI system, the KL1 language processor, and PIMOS was successfully completed. On this system, several small application programs were developed and run efficiently in parallel. This proved that symbol and knowledge processing problems had sufficient parallelism and could be written in KL1 efficiently. This success enabled us to enter the final stage.

Based on research achievements and newly developed tools produced in the intermediate stage, we made a detailed plan for the final stage. One general target was to make a big jump from the hardware and software technologies for the multi-PSI system to the ones for the PIM, with hundreds of element processors. Another general target was to make a challenge for parallel processing of large and complex knowledge processing applications which had never been tackled anywhere in the world, using KL1 and the PIM.

Through the research and development directed to these targets, we expected that a better parallel programming methodology would be established for logic programming. Furthermore, the development of large and complex application programs would not only encourage us to create new methods of building more intelligent systems systematically but could also be used as practical benchmarking programs for the parallel inference system. We intended to develop new techniques and methodologies.

1. Efficient parallel software technology
   (a) Parallel modeling and programming techniques
   -Parallel programming paradigms
   -Parallel algorithms

   (b) Efficient mapping techniques of parallel processes to parallel processors
   -Dynamic load balancing techniques
   -Performance debugging support

2. New methodologies to build intelligent systems using the power of the parallel inference system
   (a) Development of a higher-level reasoning or inference engine and higher-level programming languages
   (b) Methodologies for knowledge representation and knowledge base management (methodology for knowledge programming)

The research and development themes in the final stage were set up as follows:

1. PIM hardware development

   We intended to build several models with different architectures so that we could compare mapping problems between the architectures and program models. The number of element processors for all the modules was planned about 1,000.

2. The KL1 language processor for the PIM modules

   We planned to develop new KL1 language processors which took the architectural differences on the PIM modules into account.

3. Improvement and extension of PIMOS

   We intended to develop an object-oriented language, AYA, over KL1, a parallel file system, and extended performance debugging tools for its programming environment.

4. Parallel DBMS and KBMS

   We planned to develop a parallel and distributed database management system, using several disk drives connected to PIM element processors, was intended to attain high throughput and consequently a high information retrieval speed. As we had already developed a data base management system, Kappa-II, which employed a nested relational model on the PSI machine, we decided to implement a parallel version of Kappa-II. However, we redesiged its implementation, employing the distributed database model and using KL1. This parallel version is called Kappa-P. We plan to develop a knowledge base management system on the Kappa-P. This would be based on the deductive object-oriented DB, having a knowledge representation language, Quixote.

5. Research on knowledge programming software

   We intended to continue various basic research activities to develop new theories, methodologies and tools for building knowledge processing application systems. These activities were grouped together as research on knowledge programming software.

   This included research themes such as a parallel constraint logic programming language, mathematical systems including theorem provers, natural language processing systems such as a grammar design system, and an intelligent sentence generation system for man-machine interfacing.

6. Benchmarking and experimental parallel application systems

   To evaluate the parallel inference system and the various tools and methodologies developed in the above themes, we decided to make more effort to explore new applications of parallel knowledge processing. We began research into a legal expert system, a genetic information processing systems and so on.

5.2 R & D results in the final stage

The actual research activities into the themes described above differed according to characteristics. In the development of the parallel inference system, we focused on the integration of PIM hardware and some software components. In our research on knowledge programming software, we continued basic research and experimental software building to create new theories and develop parallel software technologies for the future.

5.2.1 PIM hardware and KL1 language processor

A role of the PIM hardware was to provide software researchers with an advanced platform which would allow large-scale software development for knowledge processing.

Another role was to obtain various evaluation data in the architecture and hardware structure of the element processors and network systems. In particular, we wanted to analyze the performance of large-scale parallel programs on various architectures (machine instruction sets) and hardware structures, so that hardware engineers could design more powerful and cost-effective parallel hardware in the future.

In the conceptual design of the PIM hardware, we realized that there were many alternative designs for the architecture of an element processor and the structure of a network system. For the architecture of an element processor, we could choose between a CISC type instruction set implemented in firmware and a RISC type instruction set. On the interconnection network, there were several opinions, including a two dimensional mesh network like the multi-PSI, a cross-bar switch, and a common bus and coherent cache.

To design the best hardware, we needed to find out the mapping relationships between program behavior and the hardware architectures and structures. We had to establish criteria for the design of the parallel hardware, reflecting the algorithms and execution structures of application programs.

To gather the basic data we needed to obtain this design criteria, we tried to categorize our design choices into five groups and build five PIM modules. The main features of these five modules are listed in Table 2. The number of element processor required for each module was determined depending on the main purpose of the module. Large modules have 256 to 512 element processors, and were intended to be used for software experiments. Small modules have 16 or 20 element processors and were built for architectural experiments and evaluation.

All of these modules were designed to support KL1 and PIMOS, so that software researchérs could run one program on the different modules and compare and analyze the behaviors of parallel program execution.

A PIM/m module employed architecture similar to the multi-PSI system. Thus, its KL1 language processor could be developed by simply modifying and extending that of the multi-PSI system. For other modules, namely PIM/p, PIM/c, PIM/k, and PIM/i, the KL1 language processor had to be newly developed because all of these modules have a cluster structure. In a cluster, four to eight element processors were tightly coupled by a shared memory and a common bus with coherent caches. While communication between element processors is done through the common bus and shared memory, communication between clusters is done via a packet switching network. These four PIM modules have different machine instruction sets.

We intended to avoid the duplication of development work for the KL1 language processor. We used the KL1-C language to write PIMOS and the usual application programs. A KL1-C program is compiled into the KL1-B language, which is similar to the "WAM" as shown in Figure 5. We defined an additional layer between the KL1-B language and the real machine instruction. This layer is called the virtual hardware layer. It has a virtual machine instruction set called "PSL". The specification of the KL1-B interpreter is described in PSL. This specification is semi-automatically converted to a real interpreter or runtime routines dedicated to each PIM modules. The specification in PSL is called a virtual PIM processor (the VPIM processor for short) and is common to four PIM modules.

PIM/p, PIM/m and PIM/c are intended to be used for large software experiments; the other modules were intended for architectural evaluations. We plan to produce a PIM/p with 512 element processors, and a PIM/m with 384 element processors. Now, at the beginning of March 1992, a PIM/m of 256 processors has just started to run a couple of benchmarking programs.

We aimed at a processing speed of more than 100 MLIPS for the PIM modules. The PIM/m with 256 processors will attain more than 100 MLIPS as its peak performance. However, for a practical application program, this speed may be much reduced, depending on the characteristics of the application program and the programming technique. To obtain better performance, we must attempt to augment the effect of compiler optimization and to implement a better load balancing scheme. We plan to run various benchmarking programs and experimental application programs to evaluate the gain and loss of implemented hardware and software functions.

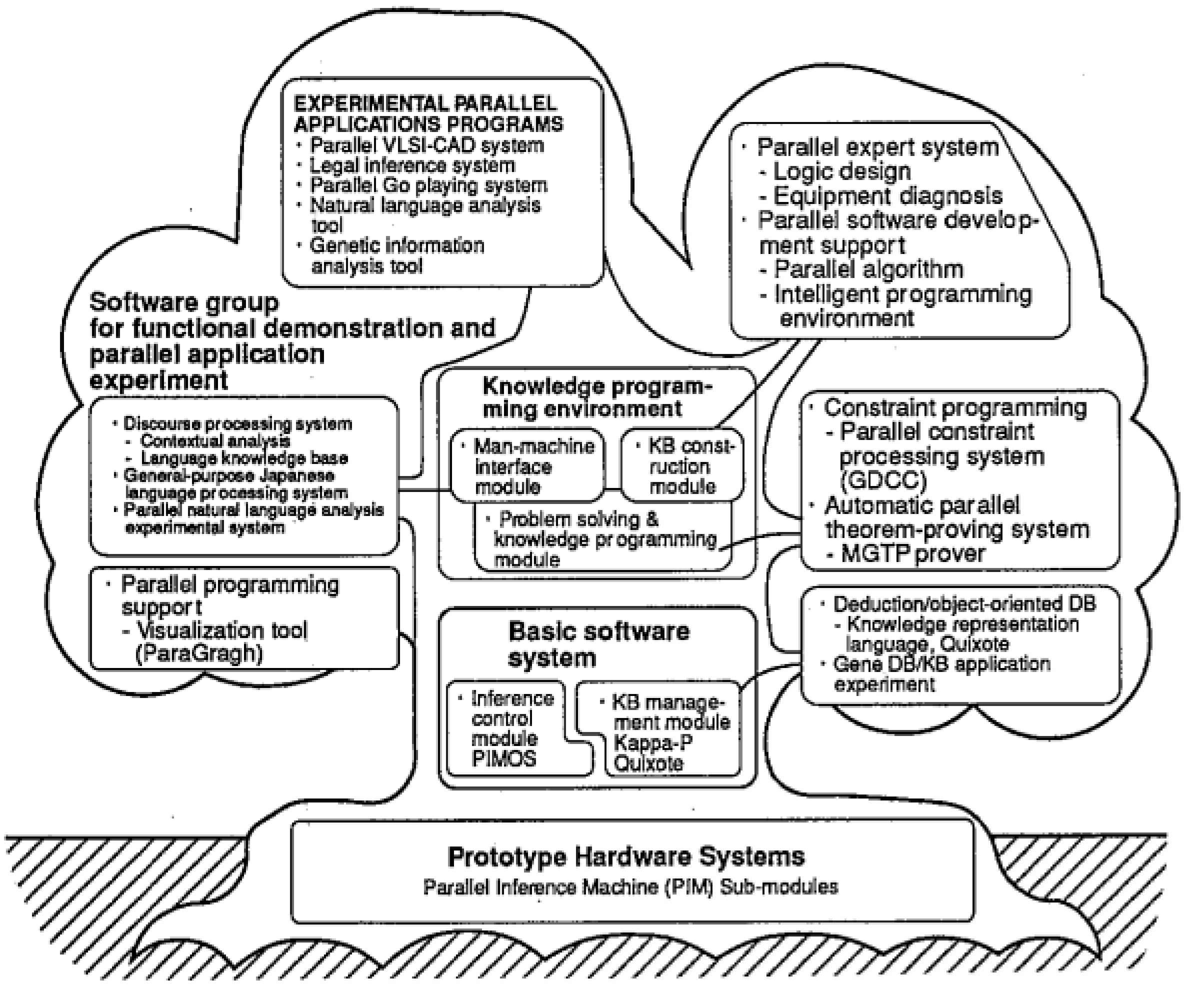

EXPERIMENTAL PARALLEL APPLICATIONS PROGRAMS
· Parallel VLSI-CAD system
· Legal inference system
· Parallel Go playing system
· Natural language analysis tool
· Genetic information analysis tool
· Parallel expert system
- Logic design
- Equipment diagnosis
· Parallel software development support
- Parallel algorithm
- Intelligent programming environment
Software group for functional demonstration and parallel application experiment
· Discourse processing system
- Contextual analysis
- Language knowledge base
· General-purpose Japanese language processing system
· Parallel natural language analysis experimental system
· Parallel programming support
- Visualization tool (ParaGragh)
Knowledge programming environment
· Man-machine interface module
· KB construction module
· Problem solving & knowledge programming module
Basic software system
· Inference control module PIMOS
· KB management module Kappa-P Quixote
· Constraint programming
- Parallel constraint processing system (GDCC)
· Automatic parallel theorem-proving system
- MGTP prover
· Deduction/object-oriented DB
- Knowledge representation language, Quixote
· Gene DB/KB application experiment
Prototype Hardware Systems
Parallel Inference Machine (PIM) Sub-modules

Figure 4: Research Themes in the Final Stage

Table 2: Features of PIM modules

Item	PIM/p	PIM/c	PIM/m	PIM/i	PIM/k
Machine instructions	RISC-type + macro instructions	Horizontal microinstructions	Horizontal microinstructions	RISC-type	RISC-type
Target cycle time	60 nsec	65 nsec	50 nsec	100 nsec	100 nsec
LSI devices	Standard cell	Gate array	Cell base	Standard cell	Custom
Process Technology (line width)	0.96 μm	0.8 μm	0.8 μm	1.2 μm	1.2 μm
Machine configuration	Multicluster connections (8 PEs linked to a shared memory) in a hypercube network	Multicluster connections (8 PEs + CC linked to a shared memory) in a crossbar network	Two-dimensional mesh network connections	Shared memory connections through a parallel cache	Two-level parallel cache connections
Number of PEs connected	512 PEs	256 PEs	256 PEs	16 PEs	16 PEs

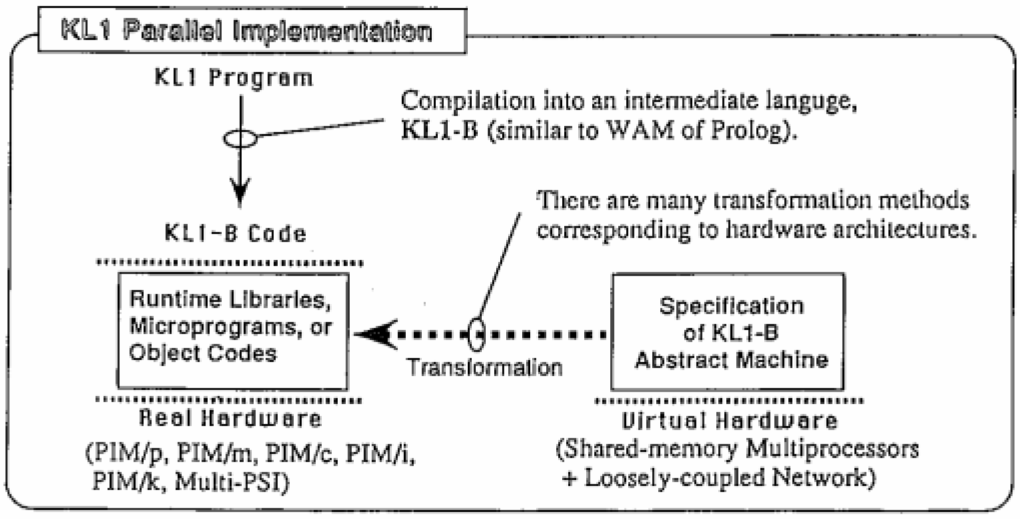
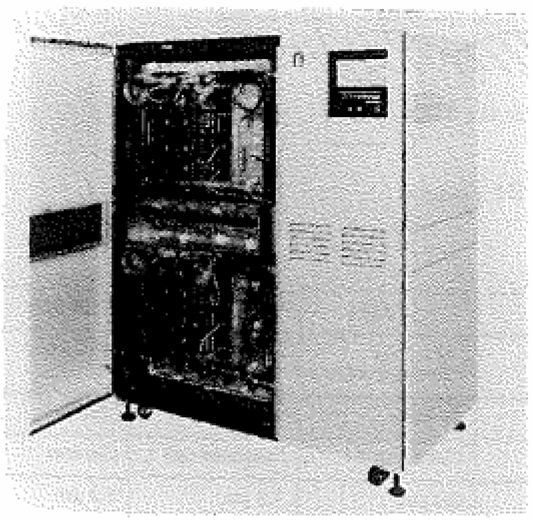
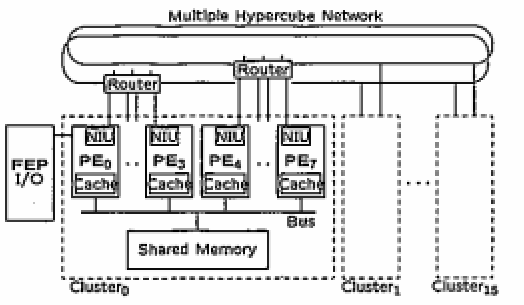

KL1 Parallel Implementation
KL1 Program
Compilation into an intermediate languge, KL1-B (similar to WAM of Prolog).
KL1-B Code
There are many transformation methods corresponding to hardware architectures.
Runtime Libraries, Microprograms, or Object Codes
Transformation
Specification of KL1-B Abstract Machine
Real Hardware
(PIM/p, PIM/m, PIM/c, PIM/i, PIM/k, Multi-PSI)
Virtual Hardware
(Shared-memory Multiprocessors + Loosely-coupled Network)

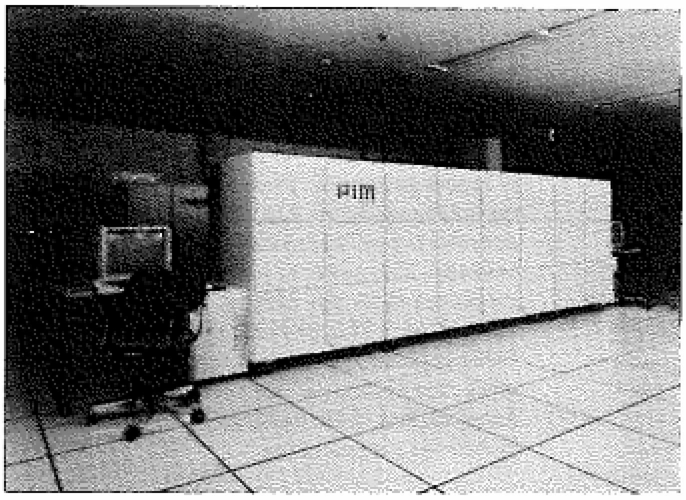
Figure 5: KL1 Language Processor and VPIM

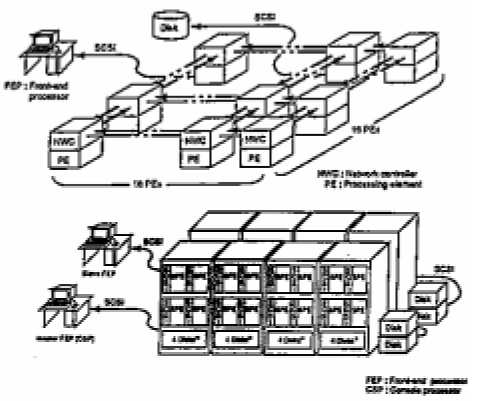




- Machine language: KL1-b
- Architecture of PE and cluster
  - RISC + HLIC(Microprogrammed)
  - Machine cycle: 60ns, Reg.file: 40bits x 32W
  - 4 stage pipeline for RISC inst.
  - Internal Inst. Mem: 50 bits x 8 KW
  - Cache: 64 KB, 256 column, 4 sets, 32B/block
  - Protocol: Write-back, Invalidation
  - Data width: 40 bits/word
  - Shared Memory capacity: 256 MB
- Max. 512 PEs, 8 PE/cluster and 4 clusters/cabinet
- Network:
  - Double hyper-cube (Max 6 dimensions)
  - Max. 20MB/sec in each link

Figure 6: PIM model P: Main Features and Appearance of a Cabinet





- Machine language: KL1-b
- Architecture of PE:
  - Microprogram control (64 bits/word x 32 KW)
  - Data width: 40 bits/word
  - Machine cycle: 60ns, Reg.file: 40 bits x 64W
  - 5 stage pipeline
  - Cache: 1 KW for Inst., 4 KW for Data
  - Memory capacity: 16MW x 40 bits (80 MB)
- Max. 256 PEs, 32 PE/cabinet
- Network:
  - 2-dimensional mesh
  - 4.2MB/s x 2 directions/ch

Figure 7: PIM model M: Main Features and Appearance of four Cabinets

5.2.2 Development of PIMOS

PIMOS was intended to be a standard parallel operating system for large-scale parallel machines used in symbol and knowledge processing. It was designed as an independent, self-contained operating system with a programming environment suitable for KL1. Its functions for resource management and execution control of user programs were designed as independent from the architectural details of the PIM hardware. They were implemented based on an almost completely non-centralized management scheme so that the design could be applied to a parallel machine with one million element processors.[Chikayama 1992]

PIMOS is completely written in KL1. Its management and control mechanisms are implemented using a "meta-call" primitive of KL1. The KL1 language processor has embedded an automatic memory management mechanism and a dataflow synchronization mechanism. The management and control mechanisms are then implemented over these two mechanisms.

The resource management function is used to manage the memory resources and processor resources allocated to user processes and input and output devices. The program execution control function is used to start and stop user processes, control the order of execution following priorities given to them, and protect system programs from user program bugs like the usual sequential operating systems.

PIMOS supports multiple users, accesses via network and so on. It also has an efficient KL1 programming environment. This environment has some new tools for debugging parallel programs such as visualization programs which show a programmer the status of load balancing in graphical forms, and other monitoring and measurement programs.

5.2.3 Knowledge base management system

The knowledge base management system consists of two layers. The lower layer is a parallel database management system, Kappa-P. Kappa-P is a database management system based on a nested relational model. It is more flexible than the usual relational database management system in processing data of irregular sizes and structures, such as natural language dictionaries and biological databases.

The upper layer is a knowledge base management system based on a deductive object-oriented database. [Yokota and Nishio 1989] This provides us with a knowledge representation language, Quixote. [Yokota and Yasukawa 1992] These upper and lower layers are written in KL1 and are now operational on PIMOS.

The development of the database layer, Kappa, was started at the beginning of the intermediate stage.

Kappa aimed to manage the "natural databases" accumulated in society, such as natural language dictionaries. It employed a nested relational model so that it could easily handle data sets with irregular record sizes and nested structures. Kappa is suitable not only for natural language dictionaries but also for DNA databases, rule databases such as legal data, contract conditions, and other "natural databases" produced in our social systems.

The first and second versions of Kappa were developed on a PSI machine using the ESP language. The second version was completed at the end of the intermediate stage, and was called Kappa-II.[Yokota et al. 1988]

In the final stage, a parallel and distributed implementation of Kappa was begun. It is written in KL1 and is called Kappa-P. Kappa-P is intended to use large PIM main memories for implementing the main memory database scheme, and to obtain very high throughput rate for disk input and output by using many disks connected in parallel to element processors.

In conjunction with the development of Kappa-II and Kappa-P, research on a knowledge representation language and a knowledge base management system was conducted. After repeated experiments in design and implementation, a deductive object-oriented database was employed in this research.

At this point the design of the knowledge representation language, Quixote, was completed. Its language processor, which is the knowledge base management system, is under development. This language processor is being built over Kappa-P. Using Quixote, construction of a knowledge base can then be made continuously from a simple database. This will start with the accumulation of passive fact data, then gradually add active rule data, and will finally become a complete knowledge base.

The Quixote and Kappa-P system is a new knowledge base management system which has a high-level knowledge representation language and the parallel and distributed database management system as the base of the language processor. The first versions of Kappa-P and Quixote are now almost complete. It is interesting to see how this big system operates and how much its overhead will be.

5.2.4 Knowledge programming software

This software consists of various experimental programs and tools built in theoretical research and development into some element technologies for knowledge processing. Most of these programs and tools are written in KL1. These could therefore be regarded as application programs for the parallel inference system.

1. Constraint logic programming system

   In the final stage, a parallel constraint logic programming language, GDCC, is being developed. This language is a high-level logic language which has a constraint solver as a part of its language processor. The language processor is implemented in KL1 and is intended to use parallel processing to make its execution time faster. The GDCC is evaluated by experimental application programs such as a program for designing a simple handling robot.[Aiba and Hasegawa 1992]

2. Theorem proving and program transformation

   A model generation theorem prover, MGTP, is being implemented in KL1. For this application, the optimization of load balancing has been made successfully. The power of parallel processing is almost proportional to the number of element processors being used. This prover is being used as a rule-based reasoner for a legal reasoning system. It enables this system to use knowledge representation based on first order logic, and to contribute to easy knowledge programming.

3. Natural language processing

   Software tools and linguistic data bases are being developed for use in implementing natural language interfaces. The tools integrated into a library called a Language Tool Box (LTB). The LTB includes natural language parsers, a sentence generators, and the linguistic databases and dictionaries including syntactic rules and so on.

5.2.5 Benchmarking and experimental parallel application software

This software includes benchmarking programs for the parallel inference system, and experimental parallel application programs which were built for developing parallel programming methodology, knowledge representation techniques, higher-level inference mechanisms and so on.

In the final stage, we extended the application area to include larger-scale symbol and knowledge processing applications such as genetic information processing and legal expert systems. This was in addition to engineering applications such as VLSI-CAD systems and diagnostic systems for electronic equipment. [Nitta 1992]

1. VLSI CAD programs

   Several VLSI CAD programs are being developed for use in logic simulation, routing, and placement. This system is aimed at developing various parallel algorithms and load balancing methods. As there are sequential programs which have similar functions to these programs, we can compare the performance of the PIM against that of conventional machines.

2. Genetic information processing programs

   Sequence alignment programs for proteins and a protein folding simulation program are being developed. Research on an integrated database for biological data is also being made using Kappa.

3. A legal reasoning system

   This system infers possible judgments on a crime using legal rules and past cases histories. It uses the parallel theorem prover, MGTP, as a core of the rule-based reasoner. This system is making full use of important research results of this project, namely, the PIM, PIMOS, MGTP and high-level inference and knowledge representation techniques.

4. A Go game playing system

   The search space of a Go game is too large to apply any exhaustive search method. For a human player, there are many text books to show typical position sequences of putting stones which is called "Joseki" patterns. This system has some of the Joseki patterns and some heuristic rules as its knowledge base to win the game against a human player. It aims to attain 5 to 10 "kyuu" level.

The applications we have described all employ symbol and knowledge processing. The parallel programs have been programmed in KL1 in a short time. Particularly for the CAD and sequence alignment programs, the processing speed has improved almost proportionally to the number of element processors.

However, as we can see in the Go playing system, which is a very sophisticated program, the power of the parallel inference system can not always increase its intelligence effectively. This implies that we cannot effectively transcribe "natural" knowledge bases written in text books on Go into data or rules in "artificial" knowledge base of the system which would make the system " clever". We need to make more effort to find out a better program structure and better algorithms to make full use of the merit of parallel processing.

6 Evaluation of the parallel inference system

6.1 General purpose parallel programming environment

The practical problems in symbol and knowledge processing applications have been written efficiently in KL1, and solved quickly using a PIM which has several hundred element processors. Productivity of parallel software using in KL1 has been proved to be much higher than in any conventional language. This high productivity is apparently a result of using the automatic memory management mechanism and the automatic dataflow synchronization mechanism.

Our method of specifying job division and load balancing has been evaluated and proved successful. KL1 programming takes a two-step approach. In the first step, a programmer writes a program concentrating only on the program algorithms and a model. When the program is completed, the programmer adds the specifications for job division and load balancing using a notation called "pragma" as the second step. This separation makes the programming work simple and productive.

The specification of the KL1 language has been evaluated as practical and adequate for researchers. However, we realize that application programmers need a simpler and higher-level KL1 language specification which is a subset of KL1. In the future, several application-oriented KL1 language specifications should be provided, just as the von Neumann language set has a variety of different languages such as Fotran, Pascal and Cobol.

6.2 Evaluation of KL1 and PIMOS

The functions of PIMOS, some of which are implemented as KL1 functions, have been proved to be effective for running and debugging user programs on parallel hardware. The resource management and execution mechanisms in particular work as we had expected. For instance, priority control of user processes permits programmers to use about 4,000 priority levels and enables them to write various search algorithms and speculative computations very easily. We are convinced that the KL1 and PIMOS will be the best practical example for general purpose parallel operating systems in the future.

6.3 Evaluation of hardware support for language functions

In designing of the PIM hardware and the KL1 language processor, we thought it more important to provide a usable and stable platform which has a sufficient number of element processor for parallel software experiments than to build many dedicated functions into the element processor. Only the dedicated hardware support built in the element processor was tag architecture. Instead, we added more support for the interconnection between element processors such as message routing hardware and a coherent cache chip.

We did not embed complex hardware support, such as a matching store of a dataflow machine, or a content-addressable memory. We thought it risky because an implementation of the complex hardware would take a long turn around time even by a very advanced VLSI technology. We also considered that we should create a new optimization technique for a compiler dedicated to the embedded complex hardware support, and that this would not easy too.

The completion of PIM hardware is now one year behind the original schedule, mainly because we had many unexpected problems in the design of the random logic circuits, and in submicron chip fabrication. If we had employed a more complex design for the element processor, the PIM hardware would have been further from completion.

6.3.1 Comparison of PIM hardware with commercially available technology

Rapid advances have been made in RISC processors recently. Furthermore, a few MIMD parallel machines which use a RISC processor as their element processor have started to appear in the market. When we began to design the PIM element processor, the performances of both RISC and CISC processors were as low as a few MIPS. At that time, a dedicated processor with tag architecture could attain a better performance. However, now some RISC processors have attained more than 50 MIPS. It is interesting to evaluate these RISC processors for KL1 program execution speed.

We usually compare the execution speed of a PIM element processor to that of a general-purpose microprocessor, regarding 1 LIPS as approximately equivalent to 100 IPS. This means that a 500 KLIPS PIM element processor should be comparable to a 50 MIPS microprocessor. However, the characteristics of KL1 program execution are very different from those of the usual benchmark programs for general-purpose microprocessors.

The locality of memory access patterns for practical KL1 programs is lower than for standard programs. As the length of the object codes for a RISC instruction set has to be longer than a CISC or dedicated instruction set processors, the cache miss ratio will be greater. Then, simple comparison with the PIM element processor and some recent RISC chips using announced peak performance is not meaningful. Thus, the practical implementation of the KL1 language processor on a typical RISC processor is necessary.

Most of the MIMD machines currently on the market lack a general parallel programming environment. The porting of the KL1 language processor may allow them to employ new scientific applications as well as symbol and knowledge processing applications.

In the future processor design, we believe that a general purpose microprocessor should have tag architecture support as a part of its standard functions.

6.3.2 Evaluation of high-level programming overhead

Parallel programming in KL1 is very productive, especially for large-scale and complex problems. The control of job division and load balancing works well for hundreds of element processors. No conventional language is so productive. However, if we compare the processing speed of a KL1 program with that of a conventional language program with similar functions within a single element processor, we find that the KL1 overhead is not so small. This is a common trade-off problem between high-level programming and low-level programming.

One straightforward method of compensating is to provide a simple subroutine call mechanism to link C language programs to KL1 programs. Another method is to improve the optimization techniques of compilers. This method is more elegant than the first. Further research on optimization technique should be undertaken.

7 Conclusion

It is obvious that a general-purpose parallel programming language and environment is indispensable for solving practical problems of knowledge and symbol processing. The straightforward extension of conventional von Neumann languages will not allow the use of hundreds of element processors except for regular scientific calculations.

We anticipated the difficulties in efficient implementation of the automatic memory management and synchronization mechanisms. However, this has been now achieved. The productivity and maintainability of KL1 is much higher than we expected. This more than compensates for the overhead in high-level language programming.

Several experimental parallel application programs on the parallel inference system have proved that most large-scale knowledge processing applications contain potential parallelism. However, to make full use of this parallelism, we need to have more parallel algorithms and paradigms to actually program the applications.

The research and development targets of this FGCS project have been achieved, especially as regards the parallel inference system. We plan to distribute the KL1 language processor and PIMOS as free software or public domain software, expecting that they will be ported to many MIMD machines, and will provide a research platform for future knowledge processing technology.

Acknowledgment

The development of the FGCS prototype system was conducted jointly by many people at ICOT, cooperating manufacturers, and many researchers in many countries. The author would like to express my gratitude to all the people who have given us much advise and help for more than 10 years.

References

[Uchida 1987] S. Uchida. "Inference Machines in FGCS Project", TR 278, ICOT, 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 project", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Nov.28-Dec.2, 1988.

[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, Japan, November 1988.

[Taki 1992] K. Taki, "Parallel Inference Machine, PIM", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Jul.1-5, 1992.

[Chikayama 1984] T. Chikayama, "Unique Features of ESP", In Proc. Int. Conf. on Fifth Generation Computer Systems 1984, ICOT, 1984, pp. 292-298.

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

[Clark adn Gregory 1983] Keith L. Clark and Steve Gregory, "Parlog: A parallel logic programming language", Research Report TR-83-5, Imperial College, March 1983.

[Clark and Gregory 1984] K. L. Clark and S. Gregory, "Notes on Systems Programming in PARLOG", In Proc. Int. Conf. on Fifth Generation Computer Systems 1984, ICOT, 1984, pp. 299-306.

[Shapiro 1983] E. Y. Shapiro, "A subset of Concurrent Prolog and Its Interpreter", TR 003, ICOT, 1987.

[Ueda 1986] K. Ueda. Guarded Horn Clauses, "In Logic Programming", '85, E. Wada (ed.), Lecture Notes in Computer Science 221, Springer-Verlag, 1986, pp.168-179.

[Ueda 1986] K. Ueda, "Introduction to Guarded Horn Clauses", TR 209, ICOT, 1986.

[Chikayama and Kimura 1985] T. Chikayama and Y. Kimura, "Multiple Reference Management in Flat GHC", In Proc. Fourth Int. Conf. on Logic Programming, MIT Press, 1987, pp. 276-293.

[Chikayama el al. 1988] T. Chikayama, H. Sato and T. Miyazaki, "Overview of the Parallel Inference Machine Operating System (PIMOS)", In Proc. Int. Conf. on Fifth Generation Computer Systems 1988, ICOT, 1988, pp. 230-251.

[Chikayama 1992] T. Chikayama, "Operating System PIMOS and Kernel Language KL1", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Jul.1-5, 1992.

[Uchida et al. 1988] S. Uchida,"The Research and Development of Natural Language Processing Systems in the Intermediate Stage of the FGCS Project", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Nov.28-Dec.2, 1988.

[Yokota et al. 1988] K. Yokota, M. Kawamura, and A. Kanaegami, "Overview of the Knowledge Base Management System (KAPPA)", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Nov.28-Dec.2, 1988.

[Yokota and Nishio 1989] K. Yokota and S. Nishio, "Towards Integration of Deductive Databases and Object-Oriented Databases–A Limited Survey", Proc. Advanced Database System Symposium, Kyoto, Dec., 1989.

[Yokota and Yasukawa 1992] K.Yokota and H. Yasukawa, "Towards an Integrated Knowledge-Base Management System", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Jul.1-5, 1992.

[Aiba and Hasegawa 1992] A. Aiba and R. Hasegawa, "Constraint Logic Programming System", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Jul.1-5, 1992.

[Nitta 1992] K. Nitta, K. Taki, and N. Ichiyoshi, "Development of Parallel Application Programs of the Parallel Inference Machine", Proc. Int. Conf. on Fifth Generation Computer Systems, Tokyo, Jul.1-5, 1992.