Computing Methods in Applied Sciences and Engineering, VI
R. Glowiński and J.-L. Lions (Editors)
Elsevier Science Publishers B.V. (North-Holland)

THE FIFTH GENERATION COMPUTER SYSTEMS AND ITS COMPUTER ARCHITECTURE

Hidehiko Tanaka

Department of Electrical Engineering
The University of Tokyo
Hongo, Bunkyo-ku, Tokyo 113
JAPAN

A project called Fifth Generation Computer Systems (FGCSs) is currently underway in Japan. The objective of this project is to develop the basic technology that is required to realize the FGCSs which will be used in the 1990s. These systems are oriented towards knowledge information processing system, the basic elements of which are the problem-solving and inference systems and the knowledge base system. A 10-year research period has been set up, and this commenced on June 1, 1982. This paper presents the general view of the FGCS, the architectural consideration, and the description of a few architectural projects.

1. INTRODUCTION

The progress of computer technology is striking nowadays. For example, we now have single-chip computers whose processing power far exceeds that of the First Generation computers. We have also various kinds of high-level languages, operating systems, and database systems. As a result, we can make programs for almost any kind of applications, provided that their algorithms can be described explicitly. This means that computers can take the place of people in many areas because of their high-speed processing ability and their large memory capability. However, there are still some application fields in which computers have not been able to demonstrate their power satisfactorily; these are pattern recognition, natural language translation, summarizing capability of sentences, learning capability of past experiences, global decision making through various kinds of information, and so on.

On the other hand, we are also facing a serious problem, namely the 'software crisis'. This arises because of the gap between the logic level required by applications and the one offered by high-level languages. As the application field expands more, the gap becomes broader and broader. This means that computers should support a higher logic level which is at the same time friendly and familiar for humans. We call such computer systems Fifth Generation Computer Systems (FGCSs) that are expected to solve these problems. In other words, the FGCSs are Knowledge Information Processing Systems (KIPSs); they have knowledge bases and are able to infer from the knowledge and solve problems as humans do. Only when computers gain such capability, they can be truly useful and tide us over the software crisis. To realize such high-level computers, various kinds of technologies are required such as knowledge representation, inference operation, knowledge acquisition, very efficient parallel execution mechanism, VLSI technology and so on. These technologies are not completed yet, but the time is now ripe to start a project to develop Fifth Generation computers which are based on artificial intelligence.

In Japan, a few study groups were formed in 1979 made up of university professors, and researchers of national laboratories and industries. In the course of three

to lead this research and development was founded in June 1982 in Tokyo and funded by the Ministry of International Trade and Industry.

Section 2 of this paper summarizes the overall view of Fifth Generation Computer Systems. Section 3 analyzes the processing model of logic programming and shows the models of logic programming oriented computer architecture. Section 4 reports the present status of computer architecture research projects which are underway at and around the Institute of New Generation Computer Technology (ICOT). Section 5 concludes this paper and shows some of the future works.

2. IMAGES OF THE FIFTH GENERATION COMPUTER SYSTEMS [2]

The FGCSs are oriented to knowledge information processing that is based on the innovative theories and technologies which offer the intelligent dialogue functions and the inference mechanism for knowledge bases that will be required in the 1990s. The functions of FGCSs can be divided into three kinds as follows:

1. Problem-solving and inference function.
2. Knowledge base management function.
3. Intelligent interface function.

The problem-solving and inference function corresponds to the basic four arithmetic operations (plus, minus, multiply, divide) and the control functions needed to perform arithmetic calculations. The system which realizes this function will be organized by the hardware inference mechanism, the control mechanism of parallel processing, the logic programs made up on these mechanisms, and the software system for higher-order predicate calculus. A language which includes the language PROLOG will be used as the interface between software and hardware. The knowledge-base management function is an innovative database management function which is enhanced by the processing function of semantic information. The relational data model is selected as the basic data model, which matches to predicate calculus very well. The software of knowledge-base management will be implemented over the hardware function which processes the relational algebra with high speed.

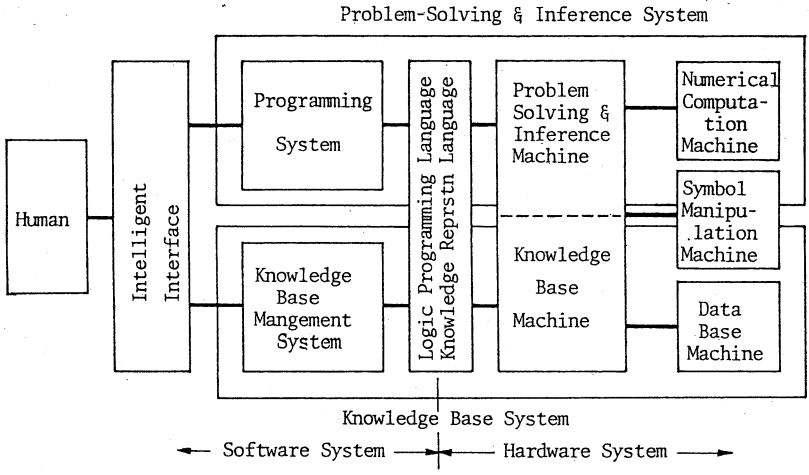


Fig.1 Image of the Fifth Generation Computer System

Intelligent interface function, which corresponds to the input/output function of conventional computers at present, will consist of a collection of systems which process, recognize or synthesize many kinds of input/output media such as characters, voice, figures, and images. It is very important to offer some friendly

interfaces for humans to exchange information with FGCSs. These functions are realized through corresponding software and hardware systems. The conceptual image of FGCSs is shown in figure 1. The right upper half of the system corresponds to the problem-solving and inference function, and the right lower half to the knowledge-base function. The left area corresponds to the intelligent interface function, which depends heavily on the previous two functions as shown in the figure. This figure shows that computers will closely approach the human system through a remarkable improvement of the logical level of the hardware system and by placing the modeling system between the hardware system and the human system.

In the knowledge-base system, three kinds of knowledge are stored: knowledge about problem domain; knowledge regarding languages, images etc. which are used by the intelligent interface function; and knowledge about the machine system and knowledge representation.

When some problem is presented by the application system through some end-user language that uses voice, figures and images, this problem is analyzed and recognized by using knowledge about the language and the images, and is transformed into intermediate specifications which are given to the programming system. The problem is understood here using the knowledge about the problem domain, and the processing specification is made up as the result. This specification is transformed into a program and optimized through referencing the knowledge about the machine system and knowledge representation. The program, written in a logic programming language, is processed by the problem-solving and inference machine and the knowledge-base machine. We think that such hardware as the symbol manipulation machine, scientific calculation machine and database machine can be regarded as machines which undertake to subcontract from the problem-solving and inference machine and the knowledge-base machine. As all of the computer hardware up to the Fourth Generation play the role of the subcontractor, we can say that the functions of hardware and software of FGCSs make the semantic gap between human and machine very small.

The inference machine which processes logic programming languages such as PROLOG is equipped with a high-power inference mechanism of hardware and a data-flow oriented control mechanism for parallel processing. The innovative von Neumann mechanism will be provided to accept the conventional programs effectively. The abstract data type support mechanism is required from the viewpoint of module programming and data security as well.

The knowledge-base machine will be realized by the combination of the relational database mechanism and the parallel processing machine of relational algebra operations, and be integrated with the inference machine in future.

These machines, of which machine language is a common kernel language for FGCSs, can be connected to each other to organize a distributed processing system. This kernel language interface the software system with the hardware system. All software systems are represented by this kernel language which the hardware system executes directly. For the implementation of this hardware, VLSI technology is indispensable as the number of gates required will be very large.

3. FIFTH GENERATION COMPUTER ARCHITECTURE

3.1. GENERAL VIEW

The major functions supported by the fifth generation computer hardware are the problem solving and inference function and knowledge base function. These functions are implemented by the inference machine and knowledge base machine, and provided through a language interface which will be developed on the basis of logic programming and knowledge representation. Though these two machines are expected to be fused into a single machine in future, we treat these machines individually at present so that the research of architecture goes at ease.

The roles of architecture support for the fifth generation computer systems can be summarized as follows.

(1) To shorten the semantic gap through supporting directly the high level operations.
(2) To get high speed processing capability, as the processing complexity will expand more and more, and the needs of high speed processing will be developed more as the application field emerges.
(3) To make it feasible to handle enormous amount of and growing knowledge data.

3.2. INFERENCE MACHINE ARCHITECTURE

A. INFERENCE PROCESS

Assume a prolog program as follows.

```
sister-of(X,Y) :- female(X),parent(X,M,F),parent(Y,M,F).   (1)
parent(alice,albert,victoria).                              (2)
parent(edward,albert,victoria).                             (3)
female(alice).                                              (4)
female(victoria).                                           (5)
```

When a question "?- sister-of(X,edward). (Who is the sister of edward?)" arises, the process to get the answer is as follows.

(a) (1) matches to the question, then,
sister-of(X,edward) :- female(X),parent(X,M,F),parent(edward,M,F).
(b) A new goal is generated.
?- female(X),parent(X,M,F),parent(edward,M,F).
(c) A literal "female(X)" is tested in the program.
(d) It is found that there are 2 candidates for X from (4) and (5).
X = alice, X = victoria.
(e) X = alice is applied to the goal.
?- parent(alice,M,F),parent(edward,M,F).
(f) The first literal "parent(alice,M,F)" is tested, and found to match when variable M and F is set to be albert and victoria, respectively.

Generally speaking, these are 2 kinds of inference. One is deductive inference and another is inductive inference. The process above is an example of deductive inference. As the deductive inference matches well to the programming, it can be set as the basic operation of inference machine. The inductive inference is expected to play an important role when knowledge acquisition is considered. Accordingly, it will be taken into account for the architecture design as the research of knowledge base

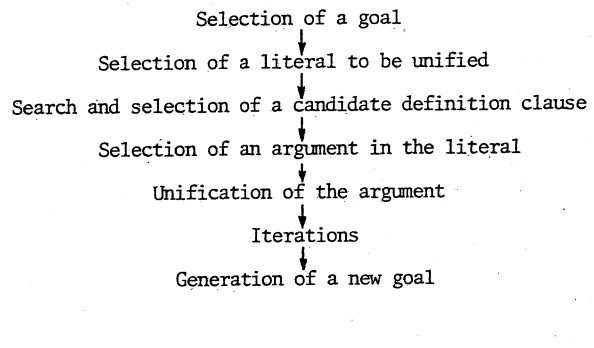


Fig.2 The procedure of inference process

progresses. So far as the deductive inference of first order predicate logic is concerned, the details of inference process can be written as a procedure of fig. 2. The procedure of fig. 2 corresponds to the machine language interpretation of conventional computers by microprogram. When we see the process from the viewpoint of logic, it can be regarded as a search process on a AND-OR tree like fig. 3. That is, a goal (?- p,q.) is expressed as an AND condition (p AND q). To satisfy a part (for example, p) of the AND condition, a few OR conditions can be found (p at node 3, or p at node 4). To satisfy each of the OR conditions, an AND condition (for example, r AND s) is required by a definition clause (for example, p :- r,s.), and so on.

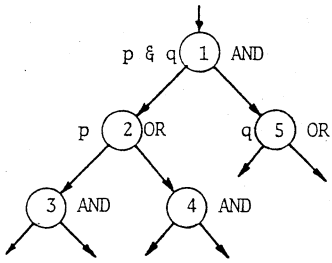


Fig.3 AND-OR tree

B. GOAL SEARCH ALGORITHMS

In logic programming based on the predicate logic, unification is the key procedure for inference. The unification is composed of a selection of a definition clause which matche to the goal and the correspondence among arguments. When a goal succeeds, the set of correspondence among arguments defines values of variables, which can be used as answer to the question. In order to make the inference process efficient, many kinds of schemes can be proposed. They can be grouped into four kinds as follows.

(1) Parallel search.
(2) Optimization of search sequence.
(3) Usage of designation by programmer.
(4) Elimination of recomputation.

PARALLEL SEARCH

On conventional computers, the inference process i.e. tree search is executed sequentially. The tree is traversed node by node. Accordingly, when a goal is failed at a node, the downward search operation is stopped there, and returned to a upper node to test the other alternatives. This return operation is called backtrack. Parallel search is a straightforward approach to speed up the search procedure. Examining the inference process of fig.2, we can find tree kinds of parallelism. First is the OR parallelism, which comes from the fact that there are several definition clauses of which head literal matches to the goal. Second is the AND parallelism among literals of a goal. Third is the intra-literal parallelism which

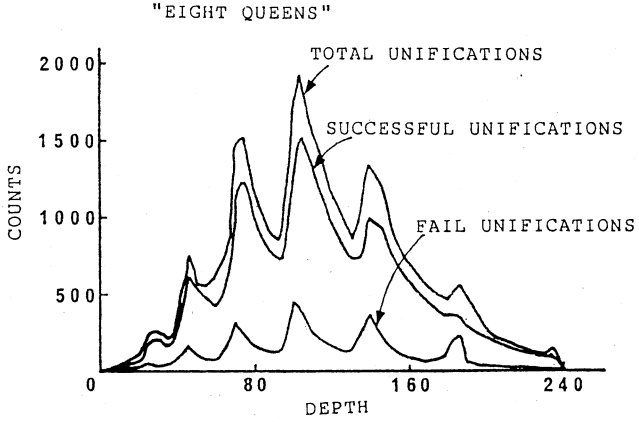


Fig.4 An example of the number of parallel goals.

performs the unification of argument in parallel with every argument. Fig. 4 is a measurement of OR parallelism. The sample program is eight queens. The horizontal axis denotes the depth of the tree. The vertical axis is the number of OR nodes at the depth. This shows that we can have a lot of parallelism easily and that the number of parallel nodes grows exponentially. Accordingly, we can expect enough parallelism through the OR parallelism. Regarding AND parallelism, the sharing of variables among literals makes the problem complicated, though AND parallelism without shared variables can be handled easily. The intra-literal parallelism can be used effectively for the design of unification executer (i.e. unifier). We can gain the speed up factor of the number of arguments through placing many unifiers of single argument in parallel.

OPTIMIZATION OF SEARCH SEQUENCES

When we want to get an answer as fast as possible, we can make the search process efficient by searching the most possible candidate first. Most popular algorithms are the depth-first one and the breadth-first one. The depth- first search selects the child node first and the breadth-first search the brother node first. Besides these, we can optimize the search sequence in terms of clauses, literals and arguments. Table 1 is an example of time measurements by changing the literal selection algorithm.

Table 1. Total number of unifications of 2 sample programs

	program JE	program EJ
scheme1	145	27
scheme2	36	27
scheme3	192	34

,where program JE(EJ) is a small program of Japanese to English (vice versa) translation.

USAGE OF DESIGNATION BY PROGRAMMER

As a programmer knows best about the behavior of his program, we can expect the efficient execution through making programmer designate the control of his program execution by introducing a few syntax into the programming languages. Examples are as follows.

a. Execution order : sequential OR, parallel OR, sequential AND, parallel AND

b. search region : To restrict the search region by removing unnecessary subtrees when some conditions met. This examples are guard[3], remote-cut[4] and success-throw[4].

c. Input/Output direction of variables. When a goal has shared variables, we face to the ambiguity which appearance of the variable defines the value. This ambiguity makes the AND parallelism difficult. Accordingly, if the ambiguity is removed with the help of programmer, it makes the AND parallel operation very easy. An example is the read-only-annotation of concurrent Prolog[3].

d. Lazy evaluation : By restricting the number of repetitive usages of some clauses, we can protect against the occurrence of unnecessary tasks.

ELIMINATION OF RECOMPUTATION

There arises a lot of unifications and interim goals during the search process. However, there may be many duplicated goals and same kind of unifications among these. We can expect to make the search efficient by eliminating these unnecessary duplicated processing. Some memory mechanism is needed to implement this operation. Regarding the duplicated goals, memorizing some middle goals as if

they are predefined theorems will improve the inference speed much. As this memorizing process changes the original program (add new knowledge), it is closely related to the learning mechanism, and further research is required. One example of this scheme is the proposal of applicative cache [7] by Keller.

There may happen the duplicate processing in a goal. For example, assume a definition of Fibonacci number as follows.

```
f(N,V):-f(N-1,V1),f(N-2,V2),add(V1,V2,V).
f(0,0).
f(1,1).
```

When a goal ?-f(6,V). is given, the first unification gives a next goal.

```
?-f(5,V1),f(4,V2),add(V1,V2,V).
```

Then, the second unification gives,

```
?-f(4,V1'),f(3,V2'),add(V1',V2',V1),f(4,V2),add(V1,V2,V).
```

In this goal, there arises literals f(4,V1') and f(4,V2). If we know that the function f is a unique function, we can remove one of them. Therefore, the goal can be changed as follows.

```
?-f(4,V2),f(3,V2'),add(V2,V2',V1),add(V1,V2,V).
```

While the original Fibonacci number calculation requires exponential order of processing, the above calculation needs linear order of processing. These elimination scheme may give substantial improvement for the inference processing.

C. PROCESSING MODELS

There are several processing models for the implementation of parallel inference machine as follows.

(1) Multi-process model.
(2) Data-flow model.
(3) Functional model.
(4) Logic model.

The models from 1 to 3 are such models that logic programs are translated into some intermediate structures which are different from model by model. On the other hand, the model 4 is a direct execution model of logic programs.

MULTI PROCESS MODEL[8]

This is a model which uses the conventional multi-processes structure for the implementation of inference processing. Processes which communicate each other are created and destroyed dynamically. Usually, each node of AND-OR tree is implemented by a process. This model is a very natural application of multi-processes structure, and can be implemented easily on conventional multi-processor computers.

DATA FLOW MODEL

Logic programs are translated into data flow graphs which are executed directory by a data flow machine. Data driven structure, which controls the start of each operation through the availability of required data, can be thought to be the basic principle of parallel processing. There are 2 research themes. One is to show the feasibility of data flow architecture of high degrees of parallelism. The other is to find out the algorithm which maps the logic programs to the data flow graphs efficiently. This algorithm is the key point of this model, which determines the parallelism of logic program execution. To implement OR parallelism on data flow model, all OR node processes are driven in parallel. The only results of processes whose unification succeed are merged into a stream and returned to the upper AND process. The sequence of this stream data is non deterministic. Regarding AND parallelism, the consistency check among the solutions is required generally. However, this may incur a lot of overhead especially in the case that the size of solution space is large. Accordingly, the pipeline schemes which executes the processing of goal literals with pipeline fashion are suggested for the AND parallelism. This is shown to be feasible through using the stream concept stated above by the PIM-D machine of ICOT.[13]

FUNCTIONAL MODEL

In this model, logic programs are translated into an intermediate form which is suitable for the execution of some functional programming languages. Examples are ALICE [9] of Imperial College of London and M3L [10] of Toulouse Univ. This model focalizes on the input/output relation of each operation. The output of function is determined by the input only. This means that the function has no side effect. This characteristics is fit for preserving the locality of each process, so that the parallel execution becomes easy. The parallelism of this model comes from the parallel evaluation of an expression. A program is represented by a graph whose each node expresses a function. When some function is reducible using rewrite rules, the rewrite operation is performed independently of others. To implement logic programs, some new rewrite rules should be added considering the variable instantiation of unifications.

LOGIC MODEL

This model simulates the behavior of logic program execution directory by hardware. Therefore, this one corresponds to the high level language machine of predicate calculus. The abstract image of this model is shown in fig. 5. In the goal pool, a lot of alternative goals are stored. Each goal data is transferred to unify-processor and performed unification. The result generats new goals and is returned to the goal pool. In this case, there are several alternatives for the implementation. For example, we can select either all data of a goal or a part data of a goal as the transfer unit between unify processors and goal pool. Though all goals are logically independent, they can be implemented by data-sharing for reasons of efficiency. OR parallelism generates several goals from a goal. Intra literal parallelism can be used to speed up the unification in the unify processor. Examples of this model is PIE [11],[12] of university of Tokyo and PIM-R [6] of ICOT. Fig. 4 is a simulation data of PIE, which shows a lot of parallelism. The machines of this model try to realize this kind of parallelism directory. As the behavior of this machine directly reflects the behavior of logic program, it is very easy to understand the machine behavior.

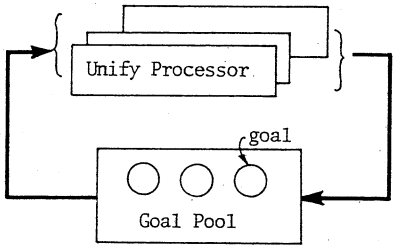


Fig.5 The abstract image of logic model

D. CONSIDERATION OF PARALLEL MACHINE IMPLEMENTATION

Followings are the consideration of parallel machine implementation by taking the logic model machine of fig. 5 as an example. The same kind of consideration can be applied to the other models as well. Assume that all of the unify processors are equipped with definition memories which store all the definition clauses. Accordingly, access to the definition memory doesn't make any influence to the behavior of major processing loop, that is the loop between unify processors and goal pool. For the detail design of this machine, 6 design points can be distinguished as follows.

(1) Transfer data unit between unify processor and goal pool.
(2) Definition memory...Shared or dedicated.
(3) Goal data...Shared or copied.
(4) Structure memory...Separated or integrated.
(5) Cache.
(6) Pipeline control.

TRANSFER DATA UNIT

Alternatives are all data of a goal and a part data of a goal. When all data of a goal are used as the unit, the behavior of machine becomes very simple. Each unify processor fetches a goal, preforms one unification and returns the new generated goals into the goal pool. All unify processor can operate independently.

On the other hand, the volume of the unit may be very large. This may reduce the speed of one round of unification. By an initial measurement result, the volume of the goal is about a few hundred bytes. When a part data of a goal is used as the unit, the transfer data between unify processor and goal pool can be reduced to only the one required. So the data traffic may be reduced. However, the access to the goal pool becomes random rather than burst, and the control can be complicated more.

DEFINITION MEMORY

The definition clauses should be accessible from all unify processors. There are 2 alternatives for the definition memory implementation. One is to use a shared memory among unify processors. The other is to equip all unify processors with their own local definition memory. To evaluate these schemes, think the model of fig. 6. Each processor has a local memory and an access path to the shared memory. Assume that one cycle of processing (unification) needs m memory accesses, within which $m\rho$ accesses are the shared memory accesses. When the total throughput of this system is H cycles/sec, the throughput restraints of memories can be expressed as

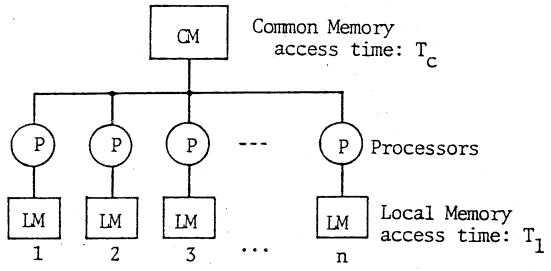


Fig.6 Shared memory model

$$\frac{H}{n} m\rho \cdot n \cdot T_c < 1 , \quad \text{and} \tag{6}$$

$$\frac{H}{n} m(1-\rho) \cdot T_1 < 1 , \tag{7}$$

where, n is the number of processors, and Tc and Tp are the access time of shared (common) memory and of local memory, respectively. From these constraints, the maximum throughput Hmax is, by changing the ρ as the parameter,

$$H_{max} = \frac{1}{m \cdot \rho_0 \cdot T_c} , \tag{8}$$

$$\text{where } \rho_0 = \frac{T_1}{nT_c + T_1} \tag{9}$$

From the equation (9), the effective degree of parallelism ne is given as follows.

$$n_e = \frac{1-\rho}{\rho} \cdot \frac{T_1}{T_c} \tag{10}$$

This means that it is useless to have more processors than ne when the shared memory access probability is given. If we require more than 100 of parallelisms, the following expression should be satisfied.

$$100 \ll \frac{1-\rho}{\rho} \frac{T_1}{T_c} \tag{11}$$

that is,

$$\rho \ll \frac{1}{100 \left(T_c / T_1 \right) + 1} \tag{12}$$

When $T_c/T_p = 2$, ρ must be less than 0.005. Therefore, shared access should be less than 0.1%, if we require the parallelism of more than a few hundreds. This means that fairly large amount of local memory should be provided as the

definition memory of each processor.

GOAL DATA STORAGE
While several new goals are generated from a goal by OR parallelism, the difference among data of these new goals comes from the difference among definition clauses which are the candidates of unification between the head literal of the old goal and definition clauses. The rest data are the same. Usually, a goal data includes many environment data such as variable instantiation. Accordingly, changed data by a unification is rather less than the unchanged one. This means that shared data scheme among goals can be very efficient. However, sharing may introduces dependency among goals, which is harmful for the highly parallel operation and makes the operation complicated. Some compromised schemes are needed, which preserve logical independence while some data are shared physically.

SEPARATION OF STRUCTURE DATA
Logic programs need to process large structure data. At that time, if all data of the structure should be always transferred from goal memory to unify processor, the overhead of transmission cannot be ignored. Instead, the operation code could be transferred to the structure, where the operation is executed. This means that it can improve the throughput of the system to provide a structure memory which is optimized for the structure manipulation and structure storage.

USAGE OF CACHE
Since cache system (buffer memory) is very effective to reduce the memory access time, it is very natural to use cache for the implementation of inference machine which is rather memory speed necked system. The memories which can be equipped with cache are the definition memory and goal memory. The definition memory can be regarded as a kind of cache for the knowledge base. Before starting some job, all necessary definition clauses are assumed to be loaded into all definition memories, in the inference machine discussed previously. The interface between definition memory and knowledge base is one of the knowledge base machine interface. This will be clarified as the research of knowledge base progresses. One more role of the definition memory is to store the intermediate results as a kind of theorems. A simple way to do this is to store all intermediate results into the capacity bounded cache. By applying "least recently used (LRU) algorithm", results of high usage-probability would be maintained in the cache. Regarding goal pool, the goal buffer memory in the unify processor acts like cache when all data of a goal is used as the transfer unit between unify processors and goal pool.

PIPELINE CONTROL
In one cycle of unification, a goal is fetched by a unify processor, the unification is executed, new goals are generated and they are returned into the goal memory. This sequence could be controlled with pipeline fashion as fig.7.

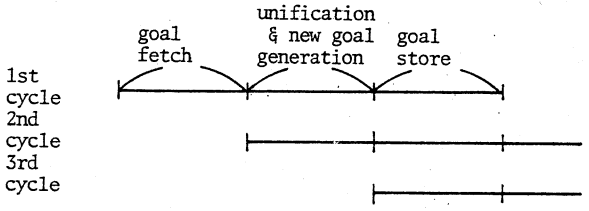


Fig.7 Pipeline control of inference machine

3.3. KNOWLEDGE BASE MACHINE ARCHITECTURE
A. THE CONCEPT OF KNOWLEDGE BASE
Knowledge information processing systems handle all information as knowledge. This includes simple data such as country population, rules such as mathematical theorems, programs, and common knowledge. From the operation point of view, KIPS can be seen as inference systems. On the other hand, KIPS can be seen as

knowledge base systems from the data point of view. As stated previously, the future FGCS will be a fused version of inference machine and knowledge base machines. However, at present, they should be investigated separately as their characteristics differ much from each other. Knowledge has many sided views. One aspect is rules and facts. The other aspect is permanent or temporary. Though mathematical theory can be regarded as permanent, knowledges expressed by some application programs are rather temporal (i.e. only within the scope of the program).

B. THE ROLES OF KNOWLEDGE BASE MACHINE

The machine which supports knowledge base is the knowledge base machine. The roles of knowledge base machine are,

(1) knowledge manipulation,
(2) knowledge storage, and
(3) knowledge acquisition.

KNOWLEDGE MANIPULATION

This includes the search operation for some stored facts or rules, and the search operation for such knowledge that are gotten as the results of the combination of stored facts and rules. The latter one is to get the results from inference operations, and corresponds to inference itself. Though any inference operation can be executed on the inference machine stated previously, there are such knowledge that are fit better for set operation like relational algebra. An example is relational data. Though relational data can be expressed by predicate calculus languages such as PROLOG, its manipulation could be better performed on some relational data base machine. This means that some part of inference should be done on knowledge base machine.

KNOWLEDGE STORAGE

Knowledge base machine should store a large amount of knowledge which will grow time by time. The same kind of functions as data base are required to maintain knowledge. Furthermore, some innovative background operation will be needed to detect inconsistency in the knowledge and to improve the efficiency of storage and of manipulation.

KNOWLEDGE ACQUISITION

Knowledge acquisition is to make up new knowledge, to test inconsistency of the knowledge world after adding the new one, and to remove unnecessary redundant knowledge. Up to this time, knowledge acquisition is done in such systems as expert systems with the help of human. Though it will be very hard to do this with full automatic, this function will be most important in future. To make up new knowledge, induction inference will be needed. Deductive inference function will be used for the inconsistency check.

C. MACHINE ARCHITECTURE

The basic architecture for the knowledge base machine is the relational data base machine. The relational data base machine will be added by knowledge operation mechanism and evolve toward the knowledge base machine. At the beginning, only facts data will be stored in the relational data base machine as they matches well to the relational data, and rules will be transferred to inference machine when they are used for execution. Typical interface to relational data base machine is relational algebra or relational calculus. As relational algebra is directly fit for the implementation of actions, it will be better to use relational algebra as the basic interface. As the relational calculus which can be mapped to relational algebra is a kind of predicate calculus, relational data base machines seem to match very well to predicate calculus languages such as PROLOG.

4. COMPUTER ARCHITECTURE RESEARCH PROJECTS

4.1. RESEARCH ORGANIZATIONS AND PROJECTS

Fifth generation computer systems project started in April 1982 with the foundation of ICOT. In June 1982, a research center inaugurated at ICOT as the kernel part for research and development of the Fifth Generation Computer Systems. In addition, the advisory groups : the Project Promotion Committee and 5 working groups, were formed to advice and encourage the activities of ICOT. These groups are organized by many researchers and experts of the organizations such as University of Tokyo, Electro Technical Laboratory of MITI, Electrical Communication Laboratory of NTT and so on.

Many research projects are going on in parallel. Among those, the computer architecture related projects are summarized as follows.

1. Sequential inference machine (SIM) project
2. Parallel inference machine (PIM) project
3. Knowledge base machine (KBM) project

The sequential Inference Machine (SIM) is a pilot model for the efficient development of software for the Fifth Generation Computer Systems. The architecture of SIM is basically the one of conventional computers and tuned for the logic programming. The object of Parallel Inference Machine project is to clarify the architecture which supports the parallel execution of inference operations. Knowledge Base Machine project is the research of such architecture that supports the execution of basic knowledge base operations. This project includes a few items. One is the research and development of relational data base machine. Second is the research of parallel knowledge operations which provide speedy knowledge accumulation, retrieval and updating, data conversion, etc. As very large scale integration technology is one of the key technology for architecture design, a few VLSI related projects are going on, such as VLSI-CAD development, and experimental fabrication of a few machine components by VLSI. The projects stated above are going at ICOT or a few organization closely related to ICOT. However, besides these, many projects which are related to the knowledge information processing systems are going on at other organizations than ICOT, as well. Followings are the introduction of computer architecture related projects which are being carried out mainly at ICOT.

4.2. SEQUENTIAL INFERENCE MACHINE PROJECTS [14]

SIM is a pilot model for software development. The kernel language version Ø, KLØ, plays the role of interface between SIM hardware and software. That is, the KLØ can be regarded as the machine language of SIM. KLØ is a predicate calculus language which includes and extends PROLOG. Each predicate in the kernel language is converted to the internal representation format of the SIM, and then directly interpreted and executed by microprogram.

The basic configuration of personal SIM (PSI) is shown in fig. 8. The data path units includes a few buses, some registers and an ALU. Some registers are specialized for resolution and unification. Resolution is a stack operation which

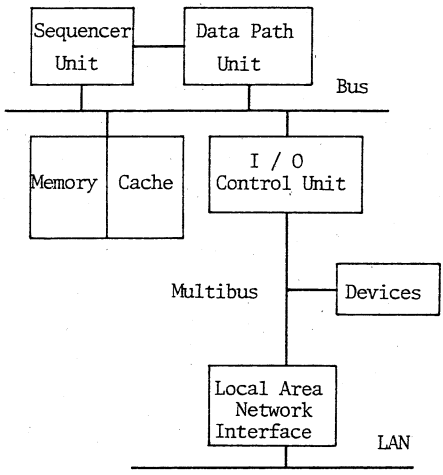


Fig. 8 The configuration of the Sequential Inference Machine-PSI

accompanies the call/return of a clause. Unification performs the data comparison with the tag check. The memory module consists of a cache and address translation mechanism, and memory board. SIM doesn't support virtual memory. The memory board will accommodate 16MW. One word is 40 bits, of which format is shown in fig. 9. The tag part of 6 bits represents data types such as

symbolic atom, integer number, real number, vector, string, variable, undefined, reference, built in code, code, control mark, etc.

SIM provides a bit-map display and pointing device, a multibus (IEEE-796) for standard input/output, and a local area network interface. The instruction expression of SIM is shown in fig. 10. This expression represents a clause in KL0, and consists of a clause header part, an argument part of the head, and two or more body-goal parts. A body-goal may contain a user defined predicate (pointer and arguments), or a built-in predicate. KL0 has four stacks : local, global, control and fail stacks. KL0 runs using these stacks and a heap area where instruction codes are stored.

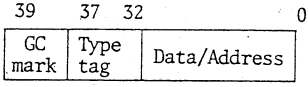
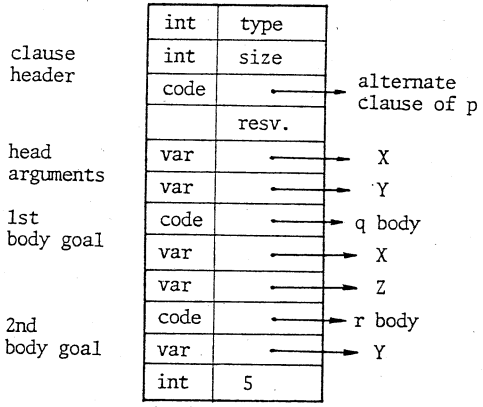


Fig.9 Word format



, where p(X,Y) :- q(X,Z), r(Y,5).

Fig.10 Clause representation of p.

4.3. PARALLEL INFERENCE MACHINE PROJECT

PIM project aims to clarify and to show the feasibility of the architecture which supports the parallel execution of inference operations. The research of a few processing models for parallel inference machine is being carried out in parallel. Followings are the brief description of 2 machines, data-flow model PIM and logic model PIM, among them.

A. DATA FLOW MODEL PIM

Among 3 types of parallelism (OR, AND, argument parallelism), OR parallelism and argument parallelism are realized on this machine. The argument parallelism is achieved by decomposing a unification process for a goal literal into several unification operations for each argument. The decomposition is represented by a data flow graph, through which efficient parallel processing is realized. Regarding AND parallelism, AND pipeline processing is realized instead of it as it may introduce great complexity.

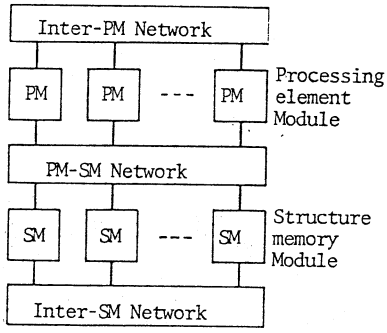


Fig.11 The configuration image of data-flow Parallel Inference Machine

The structure image of this machine is shown in fig. 11. [13] The machine is made of processing modules (PMs), structure memory modules (SMs), and three networks. PM which executes the unification processes, consists of an instruction control module and an execution module. These 2 modules form a circular pipeline structure. The instruction control module detects the firable operations of which operands are ready to be executed, and generates the instruction packets. The execution module receives the packets, interprets the instructions, and executes the corresponding functions.

The SM stores, manages and manipulates structured data such as list, vector and stream. To avoid the access conflicts on some specific SM, structures are distributed among the SMs. Each memory cell is provided with reference counter of 8 bits a read-only tag and a garbage tag of one bit so as to allow structure sharing and to control garbage collection. There are 3 networks, Inter-PM network, PM-SM network, and Inter-SM network. These networks support asynchronous packet communications. A compiler and a simulator are being developed regarding the software of this machine. The compiler compiles programs in KLØ to data flow graphs.

B. LOGIC MODEL PIM

The parallel inference machines of this model directly execute the inference operations. As shown in fig. 5, many goals are stored in a goal pool. Each unify processor fetches a goal at a time, performs unification with definition clauses and generates several new goals if the unifications succeed. Project PIE (parallel inference engine [11]) of this type is based on OR parallelism, and goals are independent each other. As each goal includes all the necessary information from the start of the inference to the processing step, goals may include a lot of same information each other. However, this doesn't always mean that each goal should use separate memory area. It is enough for each goal to be regarded as independent from logical point of view. On the other hand, you may fear at the first glance that the size of goal grows exponentially as the inference steps elapse. However, it is shown from fig. 12 that the size doesn't grow so much. The example program of fig. 12 is eight queens, which is the same program for fig. 4. (As this characteristics depend on the programs, we need more measurement results, of course.)

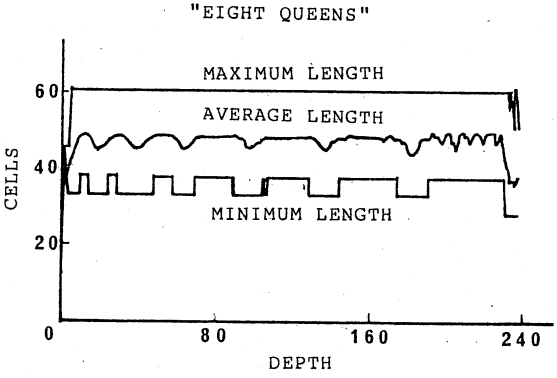


Fig.12 The change of average size of goals.

The configuration image of PIE is shown in fig. 13. Each unify processor is served by an individual definition memory, which stores the same copy of definition clauses needed for the processing. The loading of definition clauses is controlled by a system manager. Memory modules act as a goal pool on the whole. At the initial stage of research, this memory modules were assumed to memorize each goal physically independently due to its simplicity. However, data sharing schemes are being investigated now. By this scheme, each cell of memory modules is equipped with reference counter, and makes up the structure memory modules. Activity controllers control the search strategy of solutions. The examples are the "guard" operation to select only one success solution at an OR parallel execution, and "remote-cut" operation to discard some goals which turned out to be

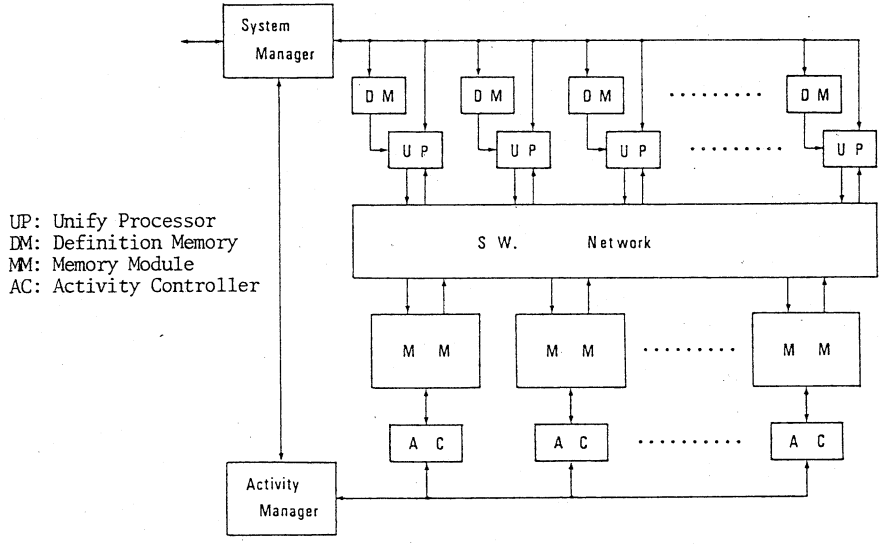


Fig.13 The configuration of Parallel Inference Engine-PIE

unneeded any more. For this purpose, each activity controller keeps the inference tree information in its memory. Following this tree, activity controllers send and receive commands each other, and control the activity of the system through cutting the tree connections or setting the status of goals "wait". The Activity Manager is the central controller of this system, while the activity controller is the local one.

For the investigation of PIE, we wrote a few software simulators such as,

1. Simulators for OR parallelism measurement,
2. Time simulator of PIE model, and
3. Structure memory simulator.

Besides these software simulator, we are building a hardware simulator and a unify processor board. The hardware simulator which will be made of a few work stations can simulate the parallel activity of PIE model. The unify processor board which will be implemented by TTL ICs is an experimental product for unification hardware.

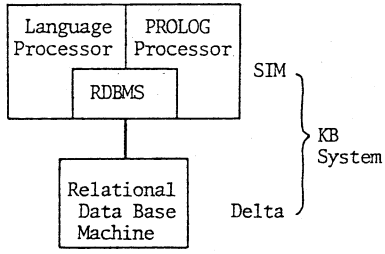


Fig.14 The experimental system of Knowledge Base System

4.4. RELATIONAL DATA BASE MACHINE PROJECT [5]

At the initial stage of FGCS project, the knowledge base system hardware is compose of a relational data base machine and a sequential inference machine. The software module for knowledge base management which runs on the sequential inference machine transforms the queries for external

data base into relational algebra commands, accesses to the relational data base machine, and returns the result information to the originator of queries. The architectural basic design data for the knowledge base subsystem will be gathered through these experiences. This configuration is shown in fig. 14. The relational data base machine for this experiment is called "Delta". Fig. 15 shows the internal organization of Delta. Control processor (CP) analyzes command trees given by SIM, breakdowns them into subcommands for hierarchical memory and relational data base engine, and manages the dictionary/directory. Relational data base engine (RDBE) is a hardware to execute retrieval subcommands in relational algebra. RDBE sorts the input attributes, and input the output into a relational operation unit. As Delta can be provided several RDBEs connected through a network, it can process the command tree in parallel.

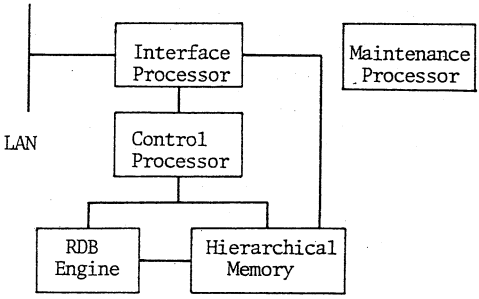


Fig.15 The internal organization of "Delta".

Hierarchical memory (HM) is a physical storage device of data. It retrieves data with high speed, and transfers the requested data to the RDBE, CP and IP. It also stores incoming data, insert, delete and modifies the stored data. The hardware components of (HM) are the working memory of high speed RAM, the silicon disk which is made of medium speed semiconductor memory to simulate a high speed disk memory, and the group of moving head disks. These 3 components form a memory hierarchy, and realize the high speed secondary storage of large volume. The working memory contains also the directory/dictionary information. The maintenance processor is a kind of supervisor processor. The roles are the bootstrapping of Delta, diagnosis of failures, recovery from the failure, loading control of existing data base, and data gathering for system evaluation.

Besides this hardware design and implementation, a software simulator is developed, whose results will be used to fix the detailed parameter and to find the system bottleneck.

5. CONCLUSION

In this paper, I summarized the fifth generation computer systems project which is now being carried out in Japan since 1982, gave consideration especially about the computer architecture for inference operations and knowledge operations, and described a few architectural projects which are going on at and around the institute of new generation computer technology.

At the end of 1984, several experimental systems are coming out such as personal sequential inference machines, parallel inference machines of data flow model and logic model, and relational data base machine. The middle term for FGCS project will begin at the fiscal year of 1985. Regarding the architectural research, the first year of the middle term will be devoted to the full scale evaluation of a few processing models. At the same time, the feasibility of parallel architecture will be evaluated in terms of the degree of parallelism. The major themes of the middle term are to built 2 subsystems, the inference machine subsystem and the knowledge base subsystem. These machines will be developed using the result of the initial term. The VLSI technology will play an important role for the

implementation of hardware.

During the initial term, the kernel language 0 (KL0) is used for the research. However this language is oriented to the conventional sequential machine. Though the second version of kernel language (KL1) is developed, which is oriented to the parallel processing and the object concept, the full scale usage of KL1 will begin at the middle term. The architectural research of the initial term is based on the subset of KL0 added by a few parallel processing feature.

[1] Proceedings of International Conference on Fifth Generation Computer Systems, Japan Information Processing Development Center, Oct.1981.

[2] Moto-oka,T. and Tanaka,H., The Fifth Generation — Progress in Japan, Supercomputer Systems Technology, Series 10, No.6, Pergamon Infotech Limited, 1982.

[3] Shapiro,E.Y., and Takeuchi, Object Oriented Programming in Concurrent Prolog, ICOT Technical Report TR-004, 1983.

[4] Chikayama,T., ESP as Preliminary Kernel Language of Fifth Generation Computers, ICOT Technical Report TR-005, 1983.

[5] Shibayama,S., Kakuta,T., Miyazaki,N., and Yokota,H., A Relational Database Machine "Delta", ICOT Technical Memorandum TM-0002, 1982.

[6] Research Report on Fifth Generation Computer Systems Project, Institute for New Generation Computer Technology, March 1983.

[7] Keller,R.M., and Sleep,M.R., Applicative Cashing: Programmer Control of Object Sharing and Lifetime in Distributed Implementations of Applicative Languages, Proc. 1981 Conference on Functional Programming Languages and Computer Architecture, 1981.

[8] Conery,J.S., and Kibler,D.F., Parallel Interpretation of Logic Programs, Proc. 1981 Conference on Functional Programming Languages and Computer Architecture, 1981.

[9] Darlington,J., and Reeve,M., Alice: A Multi-Processor Reduction Machine for the Parallel Evaluation of Applicative Languages, Proc. 1981 Conference on Functional Programming Languages and Computer Architecture, 1981.

[10] Sansonnet,J.P., Castan,M., and Percebois,C., M2L: A List-Directed Architecture, International Symposium on Computer Architecture, La Baule, May 1980.

[11] Goto,A., Aida,H., Yamazaki,A., Maruyama,T. Yuhara,M., Tanaka,H., and Moto-oka,T., On the Efficient Parallel Processing of the Highly Parallel Inference Engine — PIE, Proc. Electronic Computer Society of IECE of Japan, EC83-9, 1983.

[12] Goto,A., Aida,H., Maruyama,T. Yuhara,M., Tanaka,H., and Moto-oka,T., Proc. of The Logic Programming Conference '83, Tokyo, March 1983 (in Japanese).

[13] Ito,T., Onai,R., Masuda,Y., and Shimizu,H., Prolog Machine based on the Data Flow Mechanism, Proc. of The Logic Programming Conference '83, Tokyo, March 1983 (in Japanese).

[14] Uchida,S., Yokota,M., Yamamoto,A., Taki,K., and Nishikawa,H., Outline of the Personal Sequential Inference Machine: PSI, New Generation Computing Vol.1, No.1, 1983.