TR-425

Knowledge Base System in Logic Programming Paradigm

by
H. Itoh, H. Monoi, S. Shibayama(Toshiba),
H. Yokota(Fujitsu) and A. Konagaya(NEC)

October, 1988

ICOT
Mita Kokusai Bldg. 21F
1-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Knowledge Base System in Logic Programming Paradigm

Hidenori ITOH, Hidetoshi MONOI (ICOT)
Shigeki SHIBAYAMA (Toshiba Corp.)
Nobuyoshi MIYAZAKI (Oki Electric Industry Co. Ltd.)
Haruo YOKOTA (Fujitsu Ltd.)
Akihiko KONAGAYA (NEC Corp.)

ABSTRACT

This paper describes the current research and development status of the knowledge base subsystem being investigated in Japan's Fifth Generation Computer Systems (FGCS) project. Our aim is to realize the subsystem in the logic programming paradigm to manage large knowledge bases shared by AI application systems. In the intermediate stage of the project, several approaches are being taken to realize the knowledge base subsystem. Experimental systems are being developed in order to study the technical aspects. These systems will be integrated into the prototype of the FGCS in the final stage.

1 INTRODUCTION

The Fifth Generation Computer Systems (FGCS) project aims to develop a prototype system for a knowledge information processing system. The prototype system processes knowledge in the logic programming and parallel processing paradigms. To realize the prototype system, we have developed parallel inference subsystems and knowledge base subsystems in the intermediate stage. These subsystems are integrated into the prototype of the FGCS by the parallel logic programming language Guarded Horn Clauses (GHC) in the final stage [Itoh 88].

The knowledge base subsystem provides convenient environments in which to construct, retrieve, and manipulate large, shared knowledge bases for AI applications on the inference subsystems. The subsystem inherits most of the traditional database functions, such as access path selection and transaction control. However, knowledge base systems must have richer functions and interfaces for manipulating knowledge than traditional database systems. In other words, because the AI application programs use knowledge-representing data that has a more complex structure, the knowledge base subsystems must have high-level functions so that they can handle a large amount of knowledge and high-level interfaces between the knowledge bases and application programs.

In the initial (three-year) stage of the project, we developed a relational database experimental system *Delta* as the first step to research the knowledge base subsystem [Murakami 83] [Kakuta 85]. By doing this, we accumulated architectural experience about systems that must process large amounts of knowledge efficiently [Itoh 87]. We also developed an interface between the logic programming language Prolog and the relational database on it so that we could study the technical problems regarding their integration [Kunifuji 82] [Yokota 84] [Yokota 86a].

We are in the intermediate four-year stage, and aim to develop a prototype of the knowledge base subsystem. The subsystem can handle more complex knowledge-representing data directly and provide friendly interfaces for the knowledge processing programs based on logic programming paradigms. To develop the prototype, we have defined and developed four models of the subsystem using the sequential inference machines that were developed in the initial stage. To research the knowledge base subsystem efficiently, we employed the following approaches:

- The first approach is to extend a logic programming language that supports knowledge base functions. We have developed a practical knowledge base system with a large amount of knowledge in order to prove the effectiveness of the functions.

  The entire system is developed on the *CHI* machine with a high- performance sequential inference processor and a large-capacity memory. The memory capacity is sufficient for realizing a practical memory-based knowledge base.

- The second approach is to perform distributed knowledge base processing: the efficient retrieval and management of knowledge bases in the distributed environment. The system is developed on *PSI* machines connected by a local area network.

  In this approach, knowledge bases are realized in the context of deductive databases. We have developed software and hardware systems to manage distributed knowledge bases and to process queries.

- The third approach is to realize parallel knowledge base processing. We have developed an experimental knowledge base subsystem with multiple processing elements and a large-scale multiport memory. We have also developed the control software for the parallel processing. The experimental system is made accessible from *PSI* through a logic-based query language.

  In this approach, we adopted a relational knowledge model, an extension of the relational data model. The architecture of the experimental system follows the ideas behind database machines.

- The last approach is to research interfaces between parallel logic programming languages and knowledge bases.

  In this approach, we selected applications to study the interfaces in the parallel processing environment. We adopted the parallel logic programming language GHC and embedded knowledge-base handling functions in it.

Technologies obtained in these approaches have been integrated into the parallel knowledge base processing model.

This paper describes each system with related research topics. Section 2 describes the knowledge base system on the *CHI* machine. Section 3 describes the distributed knowledge base system using the *PSIs*. Section 4 describes the parallel knowledge base processing model. Section 5 describes the knowledge base interface system for parallel logic programming languages. Lastly, Section 6 is a summary of this paper.

2 KNOWLEDGE BASE SUBSYSTEM ON A SEQUENTIAL INFERENCE MACHINE

This section describes the high performance knowledge base system developed on the *CHI* machine [Habata 87]. We developed this system in order to investigate mechanisms for the efficient retrieval and management of knowledge bases. The novelties of the system are its practicability in terms of performance and memory capacity, and its extension of multiple name space in a multi-process environment.

2.1 Overview of the System

CHI is one of the inference machines developed in the FGCS project, designed for high performance execution of large practical logic programming programs. Figure 1 shows the *CHI* hardware configuration. The hardware consists of a high performance processor (500k LIPS for benchmark programs) and a large main memory (320 MB) connected to a front-end processor for input-output operations.

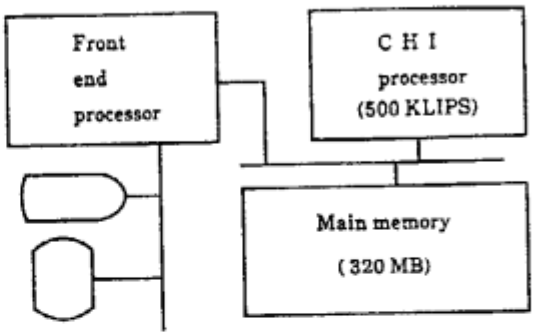

Figure 1. *CHI* hardware system configuration

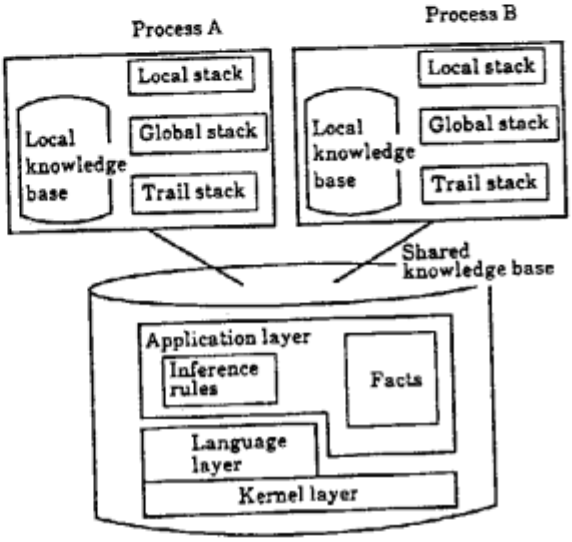

Figure 2. *CHI* software system configuration

The knowledge base system is composed of three layers: a kernel layer, a language-processing layer and an application layer (Figure 2). The kernel layer provides basic functions for multi-processing and remote input/output operations [Konagaya 87]. The language-processing layer provides a full interactive programming environment for SUPLOG [Atarashi 88], a Prolog dialect with multiple name space. The application layer provides special inference rules and facts for specific areas, such as DNA sequence matching [Doolittle 86] and machine translation systems. All processes share the knowledge base systems and execute logic programs with their own execution environment: local, global and trail stacks and a local knowledge base. From the user's point of view, *CHI* acts like a domain-oriented knowledge-base machine rather than like a Prolog machine, if application layer programs are loaded with system programs.

The high performance comes from special hardware for unification, backtracking, clause indexing and sophisticated compiler optimization [Habata 87]. To make use

of compiler optimization, we divided predicates into dynamic predicates (predicates that can change their definition dynamically) and static predicates (predicates that cannot change their definition dynamically). This division distinction is very effective because we can eliminate the overhead of predicate calling for most predicates (static ones). We also endeavored to implement high-performance dynamic predicates, since the dynamic predicates tend to form a bottleneck if they are executed by an interpreter. We introduced a "dynamic compilation" or "incremental compiling" technique that compiles a clause when asserted [Konagaya 88]. As a result, the *CHI* machine can execute dynamic predicates only three times slower than it executes static predicates.

The large memory capacity (320 MB) makes it possible to realize a memory-based knowledge base system. Knowledge base systems require a large knowledge data as well as a number of inference rules. For example, a DNA sequence matching system requires DNA data (20 million residues), and a machine translation system requires a language translation dictionary (50,000 words). From a practical point of view, large knowledge data retrieval is the most time-consuming process in the implementation of practical knowledge base systems. The memory-based knowledge base system solves this problem, since it eliminates disk access time, which occupies a large proportion of the data retrieval process in conventional computer systems.

A multiple-multiple name space has been introduced to avoid interprocess name conflict and to represent a hierarchical knowledge database. To solve the interprocess name conflict, the multiple-multiple name space facility copies name spaces when a process is created. The name space copying scheme enables processes to access name spaces independently while sharing clauses.

The hierarchical knowledge database can be obtained by the encapsulation, inheritance and shadowing mechanisms of the multiple-multiple name space. The encapsulation mechanism enables the use of the same name in a different way in the knowledge base. The inheritance mechanism provides an efficient way of defining shared clauses. The shadowing mechanism is used for solving name conflicts that occur in inheritance. The mechanism is also useful for representing non-monotonic logic.

The following sections give further details about the multiple-multiple name spaces that play an essential role in knowledge base systems.

2.2 Multiple-Multiple Name Spaces

Multiple-multiple name spaces provide an elegant way of implementing a shared knowledge database in a multiprocess environment. The shared knowledge base is very important, especially in the field of co-operative problem solving. The problems that have to be solved are

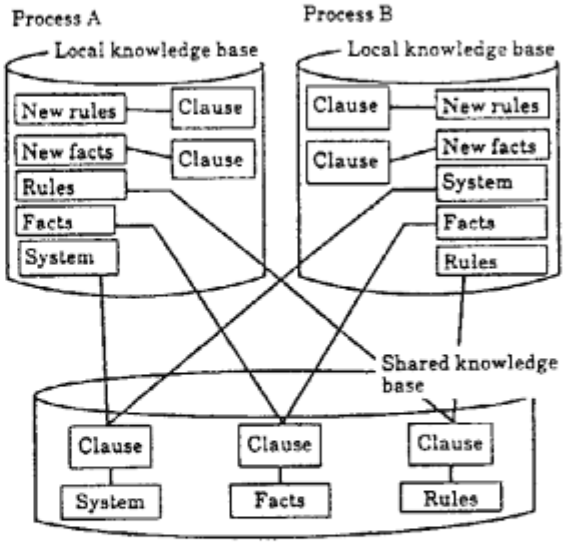

Figure 3. Multiple-multiple name space

name consistency and name conflict between processes. The inter-process name conflict results from the inherited nature of a knowledge base that permits it to update its component (clause) dynamically. A lock mechanism may save this problem, but would still leave a scheduling problem; the results of the program might change depending on the process scheduling. Our observations about knowledge bases lead us to conclude that most knowledge bases are static data. We solved the problem by dividing a knowledge base into two parts: a shared knowledge base and a local knowledge base. The shared knowledge base contains all system programs and the static knowledge. The local knowledge base contains the process' own programs and dynamic knowledge. To realize interprocess communication, we chose a mail box, a message based communication, rather than the shared knowledge base, since updating the shared knowledge base causes nondeterminacy of knowledge base access.

The inter-process name conflict may occur when processes share a name space. To solve the problem, we adopted the following name space copying scheme. In the scheme, each process copies the name tables of the shared clause database, if one has been created. The point is that the copied name tables are in the local clause database, so each process can change any name space without affecting other processes. Figure 3 shows an example of this scheme. In this example, each process has three shared-name spaces: "system", "rules" and "facts", and two local name spaces: "new_rules" and "new_facts".

Process scheduling does not affect program execution, no matter how the program changes a clause database dynamically. Local clause database can be removed when a process is terminated or killed.

2.3 Hierarchical Knowledge Base

Multiple-multiple name spaces also give us an elegant way of representing a hierarchical knowledge base that supports encapsulation, inheritance and shadowing of predicates. These facilities make it possible to represent frame-like hierarchical knowledge naturally in logic programming paradigms.

Encapsulation The encapsulation facility reduces name-conflict and increases reliability by hiding internally used predicates from outer-worlds. For example, the knowledge about Mr. Konagaya's account may be written in the following way.

```
:- in_package(konagaya).
:- export withdraw/2, deposit/2.
:- dynamic current_account/1.

   withdraw(Amount,New_balance) :-
        retract(account(Balance)),
        New_balance is Balance - Amount,
       (New_balance >= 0
            -> assert(account(New_balance));
        print("Not enough balance!"),
        assert(account(Balance))).

   deposit(Amount,New_balance) :-
        retract(account(Balance)),
        New_balance is Balance + Amount,
        assert(account(New_balance)).
```

In the above case, the predicate account/1 is used only for keeping Konagaya's current balance. So it should be hidden so that no one can access the balance directly.

Inheritance The inheritance facility enhances hierarchical knowledge representation such as frame theory [Minsky 74] and scripts. One of the great advantages of the name-based inheritance in a clause database is that we can construct both rule hierarchy and data hierarchy in the same way. That is, we can provide more flexible and powerful ways of mixing rule sets and data sets than conventional AI-tools can.

For example, a class of bird may have a general property of birds, such as that a bird has two wings, or a bird can fly. These rules can be described as follows.

```
:- in_package(bird).
:- external wings/1, canfly/1.

   wings(2).
   canfly.
```

A class of sparrow can be defined inheriting a class of bird.

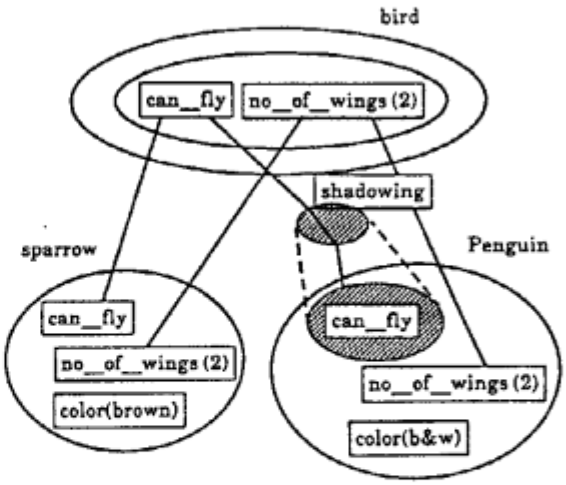

Figure 4. Inheritance in knowledge bases

```
:- in_package(sparrow,[$use(bird)]).
:- external color/1.

   color(brown).
```

Shadowing The shadowing facility makes it possible to hide some predicates so that they are not inherited from the super class. A kind of non-monotonic knowledge can be represented by using the facility. For example, a class of penguin can be defined inheriting a class of bird, but the predicate canfly/0 can be shadowed.

```
:- in_package(penguin,[$use(bird)]).
:- external color/1.
:- shadowing canfly/0.

   color(b & w).
   canfly :- fail.
```

2.4 Summary and Future Works

A clause database can be extended to a knowledge base by means of a multiple-multiple name space. The multiple-multiple name space also gives an elegant way of sharing a knowledge base among processes.

The knowledge base system can be extended to an object-oriented base by introducing a history-dependent data structure, that is, objects. In the system, a clause may be used to define constraints between objects.

3 DISTRIBUTED KNOWLEDGE BASE SUBSYSTEM

Coordination of various knowledge bases and processing knowledge bases in a distributed environment is im-

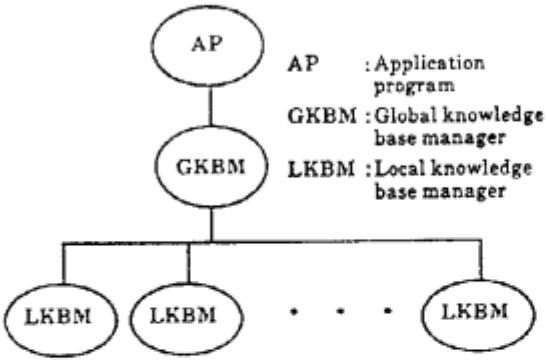

Figure 5. Logical configuration of the *PHI* system

portant for future knowledge information processing systems. One of the most fundamental issues in the study of the knowledge base is the knowledge base model as a framework. We have selected a deductive database as a fundamental platform to study knowledge bases in distributed environment. We call this system a distributed deductive database (DDDB) system.

3.1 Overview of the System

A deductive database consists of an intensional database (IDB), a set of rules, and an extensional database (EDB), a set of facts. The EDB is assumed to be much larger than the IDB. There is a well known one-to-one correspondence between a ground unit clause of the EDB and a tuple of a relational database. We have adopted a two-layered configuration: the lower layer, a relational database management system, handles the EDB and the upper layer handles the IDB.

In order to support a distributed environment, we gave the deductive database system global knowledge managers and local knowledge base managers. An experimental system, the Predicate logic based HIerarchical knowledge management (*PHI*) system, is being developed to study technical issues. In this system, one global manager and one or more local managers are dynamically assigned to each user or application program as shown in Figure 5.

The principal technical issues being investigated in the research of the DDDB system are as follows.

- Distributed query processing.
- Distributed database updating and management.
- Interface between logic programming languages and the DDDB system.
- Architecture of a dedicated processor for efficient handling of the deductive database.

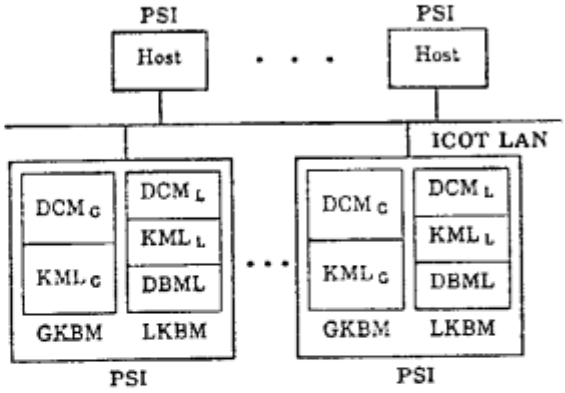

Figure 6. Physical configuration of the *PHI* system

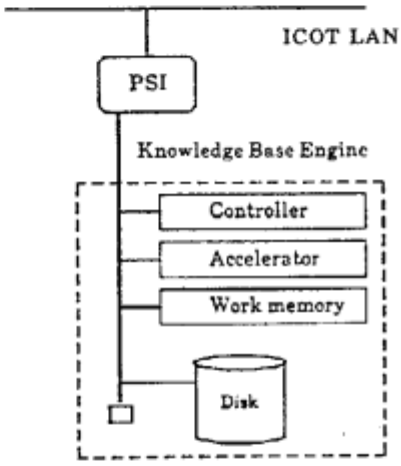

Figure 7. Knowledge base engine

The *PHI* physically consists of a number of personal sequential inference machines (*PSIs*) as shown in Figure 6. Each site has a global manager and a local manager. A dedicated processor is designed as an attached processor of a *PSI*. The processor adopts a superimposed code scheme, and has an accelerator for processing indexes based on the scheme illustrated in Figure 7. An experimental application program for software development is also being developed to investigate the functionality and performance of the system.

3.2 Distributed Deductive Database

Principal Features A DDDB consists of a deductive database distributed over a number of sites. A set of ground unit clauses (facts) having the same predicate symbol corresponds to a relation. IDB is regarded as an extension of views in relational database. A query is denoted by a goal atom or a set of clauses. The answer

is a set of ground instances of the query that are "logical consequences" of the set of clauses in the deductive database and of the sets of clauses in the query.

Principal features of the *PHI* are as follows.

- The database is a set of function-free clauses which may have negative literals in their bodies.
- Data manipulations are performed by means of a logic data language that includes extended definite clauses.
- The query processing strategy is a bottom-up strategy with query transformation and dynamic optimization.
- Concurrency control is performed by a two-phase lock method.
- Recovery is performed by a two-phase commitment method.
- Security management is provided using password and data catalogs.

Algorithms used for the last three features above are similar to those developed for traditional distributed relational databases.

The interface of the *PHI* is designed to be embedded in sequential logic programming languages such as Prolog and ESP (Self-contained Extended Prolog). The *PHI* computes the answer to a query as a set, and returns the answer piece by piece to the user program by instantiating values to variables in order to adjust to the sequential execution of the host languages. If a backtrack occurs in the user program, the system returns an alternative answer.

Distributed Query Processing Strategy In DDDB system, it is important to reduce the communication cost to transfer intermediate results by determining appropriate transfer directions. For instance, when the system joins two intermediate results, transferring the smaller one is better. There are two ways to determine transfer direction. One is a static optimization strategy that determines the directions by predicting the sizes of intermediate results before the actual processing. The other is a dynamic optimization strategy that determines the directions by comparing sizes of actual intermediate results during the processing. The *PHI* uses the dynamic optimization strategy because it is difficult to predict sizes of intermediate results for recursive queries. This decision reduces the management overhead of statistical information necessary to predict the size of intermediate results, but increases communication overhead to compare sizes of intermediate results [Takasugi 87]. The latter problem is not serious in the *PHI* because of the broadcast communication capability of *ICOT-LAN* [Taguchi 84].

Recursive Query Processing Strategy Recursive query processing strategies are classified into top-down strategies and bottom-up strategies. A top-down strategy computes the answer to a query by generating subqueries in a similar way to that of Prolog. A bottom-up strategy computes the answer by generating intermediate results from relations in the EDB. We have adopted a bottom-up strategy in the *PHI*, because a top-down strategy results in large communication overhead with frequent interactions between sites. Bottom-up strategies have two problems:

1. They compute unnecessary results because they compute all elements of the least fixpoint (least Herbrand model) of the database.
2. The iterative procedure which computes the least fixpoint involves a lot of redundant computations.

To solve the first problem, query transformation procedures are used. They transform queries to other forms that have smaller least fixpoints while preserving the equivalence of answers. To solve the second problem, a differential computation technique [Balbin 87] is used.

Query Transformations Query transformation procedures called Horn clause transformations (HCTs) are used to transform a set of clauses to an equivalent set of clauses [Miyazaki 88a] [Miyazaki 88b] [Sakama 87]. Three kinds of HCTs have been proposed for the system. They are all based on a fundamental procedure called "clause replacement". Because unnecessary information is removed from the database, the resultant database has a smaller least Herbrand model than the original database. Adding logical consequences preserves the equivalence of the transformed result for a given goal. HCTs are briefly described below.

HCT/P (HCT by Partial evaluation) :
This is a procedure that uses resolution to obtain logical consequences. It is regarded as a generalization of a procedure that substitutes the relational algebra expression of a (derived) relation for the relation symbol. It is called HCT/P because it is based on the partial evaluation technique developed for program transformation.

HCT/R (HCT by Restrictor) :
This is a procedure that uses new predicates called restrictors in order to construct clauses that are logical consequences of original clauses based on the subsumption. HCT/R results in a similar transformed database to the magic set transformation [Bancilhon 86].

HCT/S (HCT by ground Substitution) :
: This is a procedure that substitutes ground terms for variables of a clause to obtain logical consequences. This procedure is a generalization of procedures that move the constant in transitive closure operation.

3.3 Handling Negations

The *PHI* allows negative literals in bodies of clauses. This extension introduces some difficulties to the system:

- The semantics of such a database is difficult to define without some syntactical restrictions.
- Efficient query processing for such database is more difficult than for definite databases.

The *PHI* restricts the database to a "stratified" database [Apt 88]. A stratified database is a set of extended clauses that has no recursive paths involving negations. The stratified database can be partitioned into layers, and the semantics of the database are defined layer by layer from the lowest layer. The semantics of stratified databases has been extensively studied by many researchers [Apt 88] [Van Gelder 86] [Gelfond].

For instance, let us consider the following extended clause.

```
r(X,Y) :- p(X,Y),¬q(X,Z)
```

This clause has a variable, Z, which appears only in a negative literal. This Z is attached by an implicit universal quantifier according to the standard logical interpretation of clauses. It is inefficient to process this kind of clause by a bottom-up procedure, because it is necessary to check all instances of q(X,Z) or actually obtain ground instances of ¬q(X,Z). So the *PHI* handles these kinds of variables as if they are attached by existential quantifiers instead of universal quantifiers. With this convention, the above clause is equivalent to the following clauses.

```
r(X,Y)  :-  p(X,Y),¬q1(X).
q1(X)   :-  q(X,Z).
```

This convention enables us to compile negative literals to difference operations in relational algebra. It is also used in many Prolog processors.

Query evaluation methods for stratified databases have been also investigated by several researchers. As in the case of definite databases, these methods are classified into either top-down computation or bottom-up computation. As for top-down computation, several query evaluation methods for stratified databases have been recently proposed [Seki 88] [Kemp-Topor 88]. Since the usual SLDNF-resolution is obviously insufficient, these methods have incorporated some bottom-up computation features into a top-down algorithm. In [Seki 88], for example, a query evaluation method called OLDTNF-resolution has been proposed, which is based on OLDT resolution (Ordered Linear Resolution with Tabulation) [Tamaki 86], augmented with negation as failure rule. OLDTNF-resolution was shown to be sound and complete with respect to the standard model semantics for a class of stratified programs under reasonable assumptions for database applications.

The bottom-up query processing of stratified database in the *PHI* is basically same as the query processing of definite database. The *PHI* first transforms a query to an equivalent form using HCTs, and then computes the results layer by layer. However, unconditional usage of HCTs may result in unstratification. HCT/P and HCT/S can be used in stratified database without limitation, because they preserve layered structure of the database. HCT/R may transform a stratified database to an unstratified database, and it is difficult to handle unstratified database in general.

3.4 Superimposed Code Scheme for Deductive Databases

In a deductive database system that adopts a bottom-up strategy, operations such as selections, joins, set operations and set comparisons are frequently performed. The frequent usage of set operations and set comparisons is a major difference between a deductive database and a relational database. The concept of superimposed codes, which originally was proposed for text processing, possibly provides a unified approach that will realize efficient processing of both EDB and IDB [Wada 88] [Morita 88]. Superimposed code schemes have been studied for the knowledge base engine.

Superimposed Code Scheme for the EDB In relational database, indexes to attributes are used for efficient access to tuples in an EDB. If only a few attributes are frequently used in conditions of queries, the design of the indexes is easy. This is usually the case in business applications. We consider that more uniform treatment of attributes is necessary in deductive database. An index scheme based on superimposed codes is a good candidate for such a purpose. The index is obtained as follows (Figure 8).

1. The value of each key attribute is hashed to a code called a binary coded word (BCW)
2. All BCWs for a tuple are ORed together to obtain a superimposed code word (SCW)

The SCWs are much smaller than the original tuples.

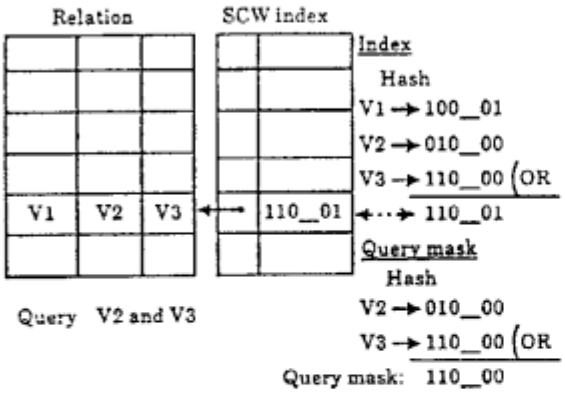

Figure 8. Example of SCW

Retrieval using this index is performed as follows.

1. The value of each key attribute in the query is hashed to obtain BCW.
2. BCWs are ORed together to obtain a query mask Q.
3. Check the SCW index if each SCW satisfies (Q 'and' $SCW = Q$). If a tuple corresponding to the index satisfies the query condition, the SCW satisfies this condition.

Set operations and set comparisons necessary to process recursive queries can also be performed with SCW indexing. The SCW indexes are used to make pairs of indexes whose corresponding pairs of tuples may be the same. Because the SCWs are much smaller than the original tuples, we can improve performance by preprocessing with the SCW index.

The advantages of the superimposed code scheme are as follows.

- The total size of the indexes is smaller than in other index schemes if there are many key attributes. In deductive databases, all attributes might be keys.
- Performance is better if more than one key attribute is specified in a query.
- Index processing can be easily performed in parallel, because the structure of the index is simple.

The disadvantages of the superimposed code scheme are as follows.

- A whole index scan is usually necessary. Although the index may be small, the index scan is still time consuming.
- Retrieval cannot be efficiently handled with range conditions.

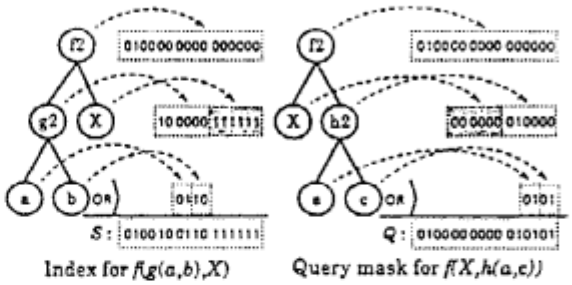

Figure 9. Example of SSCW

Dedicated hardware, a parallel processing architecture, or a combination of both can solve the first problem. Dedicated hardware is used in the experimental system for index processing.

The superimposed code scheme can be extended for structures (functions) and rules. Structures and rules can be handled by a superimposed code scheme for terms. The extended scheme uses structured superimposed code words (SSCW) an example of which is illustrated in Figure 9 [Morita 88].

4 PARALLEL KNOWLEDGE BASE SUBSYSTEM

This section describes the knowledge base system based on the parallel knowledge base model. The distributed model mentioned in the previous section assumes an environment where inference machines (*PSIs*) are connected by a local area network. In that sense it investigates a knowledge base processing scheme among the distributed processing powers. The parallel model, however, is a processing scheme to enhance the processing power of a network site.

4.1 Overview of the System

This system aims at implementing an experimental parallel knowledge base system (*Mu-X*) as the backend of the *PSI* machines. In this approach, dedicated hardware with multiple processors and a large-scale multiport shared memory is implemented.

The *Mu-X* adopted the term-relational model proposed in [Yokota 86b]. The term-relational model was used as a candidate for bridging the gap between logic programming languages and databases. The model could be considered to be a basic mechanism to implement deductive database systems. However, in this research, more attention was paid to providing primitives of term-relational model manipulation. The term relations can naturally store basic logic programming constituents (terms) and provide retrieval capabilities, based on unification, for terms. As a concrete example, a unification-based query language has been implemented [Monoi 88b]

on the model. It is based on relational calculus and interfaces *PSI* programming environment and the experimental machine. A set of classes were written in ESP [Chikayama 84] and added in the *PSI* programming environment. These classes provide methods (predicates) which interface with the user in the *PSI*'s programming environment and the *Mu-X*. The classes are activated by the method call from user programs. It forwards the message specified by the method call (typically, a "retrieve" predicate) to the *Mu-X* using network facilities for execution.

Put simply, the *Mu-X*'s role in this context is to be a backend machine for execution of the queries denoted by the retrieve predicates of ESP. Parallel processing was adopted to accelerate the retrieval. This will be described in later chapters. This experimental machine shares many research issues with parallel database machines [Shibayama 87].

4.2 Hardware Considerations

Mu-X has a shared memory multiprocessor architecture (Figure 10). There are two types of shared memories. One is conventional word-granularity shared memory for control information storage and can be regarded as an interconnection structure for multiple processing elements. The other is page-granularity conflict-free multiport page-memory for working knowledge base storage [Tanaka 84b]. The multiport page-memory consists of a set of ordinary memory banks, a switching network for interchanging the multiple ports and memory banks, port controllers attached to each port and a main controller. By cyclically interchanging the network and appropriately reading/writing the proper part of memory banks, simultaneous access from each port to arbitrary memory pages is realized. The multiport page-memory was incorporated so that several idle processing elements (PEs) could participate in the processing of a query without any memory access interference. From another point of view, the multiport page-memory can enhance the memory bandwidth to the multiple of memory banks (usually, number of ports).

The I/O bandwidth enhancement is achieved by providing a disk system to each of the PE. Term relations are horizontally partitioned and stored across the disk systems.

This architecture follows that of the knowledge base machine architecture given in [Yokota 86b] and [Morita 86]. However, simulation study of the architecture [Sakai 88] [Monoi 88a] revealed that even multiple brute-force hardware engines did not provide a performance improvement proportional to the number of PEs. This is because of the input-length dependency of the processing times. If a join processes the area of a rectangle that has sides whose lengths are the cardinalities

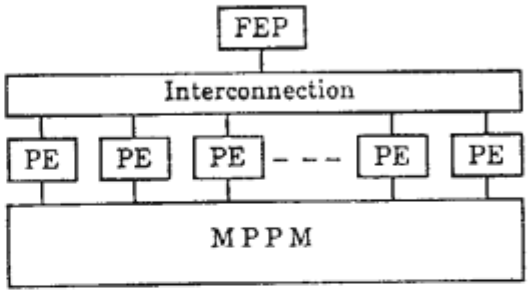

Figure 10. Hardware configuration of the parallel knowledge base system

Table 1. Hardware specifications

Number of PE	8
PE core	MC68020 at 12.5MHz
PE memory	2MB
Multiport page-memory	8 ports
	64MB with 512-byte pages
	5MB/sec/port transfer speed

of the relations, division of the area increases the total input data that must be read to be processed.

So even using a lot of engines that can process join with only the data input time will not reduce the processing time. It was also recognized that a hardware-oriented engine could only perform a limited class of operations. At the time the hardware design of this experimental machine began, it was not clear what operations should be supported by the processing element core.

For these reasons it was decided that the *Mu-X* would not incorporate hardware engines. Instead, it incorporated general-purpose microprocessors in place of the hardware engines. The effort to implement a more flexible unification engine is carried out separately. The multiport page-memory was implemented with eight ports and has a capacity of 64MB. The specification of the hardware is shown in Table 1.

4.3 Software Considerations

The software's aim in this system is to pursue parallel processing technology in the field of knowledge base processing. This aim shares much with database systems research. There are numerous researches belonging to this category, for example, *GAMMA* [DeWitt 86], *Grace* [Kitsure 82], *MPDC* [Tanaka 84a], and *MDBS* [Demurjian 86]. The characteristics of this research are as follows:

- Moderate size of experimental machine.

  Grace and *MPDC*, for example, are systems that require enormous effort to implement because of

the variety of hardware components and the complexity of the software. The *Mu-X* falls into a simpler category of parallel processing. There are two kinds of hardware components that must be programmed. One is the processing element (PE), the core of the processing, and the other is the front end processor (FEP). Since the FEP's functions are very simple, the PE is the only component that needs intensive programming.

- Incorporation of terms as the basic data representation scheme

  This system manipulates terms in much the same way that inference machines do. We not only provided an additional data type (term) but also adopted it as the basic data representation scheme in the system. For example, in the interface between *PSI* and the FEP, term representation is used to denote the query language.

- Flexibility of the software

  The system is experimental, so later modification or addition of operations is quite probable. The system software has been designed to cope with those changes.

Parallel Processing

(a) Consideration of hybrid memory systems

The parallel processing in this system is strongly influenced by the two types of memory system: a conventional shared memory and the multiport page-memory. The software is designed to make the best use of the characteristics of the memory systems.

The conventional shared memory has the following characteristics.

- The unit of access is typically a word.
- There is potential access conflict among multiple PEs.
- Access (when there is no memory access conflict) is quick, typically within a few microseconds.

The multiport page-memory is a page-based memory system activated by means of a control block (page transfer control block, PTCB for short). It has the following characteristics.

- The unit of access is a page.
- There is no access conflict among PEs (PE ports).
- Access is associated with overheads.

The overheads are of three types. The first is the overhead similar to the latency of disk access. This is the time that it takes for the asynchronous memory page access request (through the PTCB) to be recognized by the port controller that polls for the request. In this implementation, the polling interval is equal to the page transfer time, so on average there is half the page transfer time latency. The second type is the overhead of one-page transfer. This is the time that it takes for the requested page to be transferred to a buffer space. The last one is software overhead required to prepare a PTCB for the multiport page-memory. It consists of searching the multiport page-memory directory for the proper page number, assigning a destination buffer, making up a PTCB and so on. In the current implementation, four physical pages of 512 bytes constitute a logical page of 2 KB. As physical page transfer time is 100 microseconds and is the interval of request polling, one logical page transfer requires $4 \times 100 + 100/2$ or 450 microseconds on average. The software typically requires about 500 microseconds. To sum up, the transfer time for one logical page is about one millisecond. Both the hardware speed and software speed could be improved using faster technology for the former and a faster processor with cache memory for the latter.

Considering these characteristics, using the multiport page-memory as a buffer memory for the database pages was a natural choice. We also decided to place the system directory in the multiport page-memory. Initially it is stored in the disk and at startup time is loaded into the multiport page-memory so that the PEs can access the shared information quickly. The directory related to a PE is further copied in the local memory of the PE. Other control information, such as command queues, is placed in the conventional shared memory. Locking is done using the conventional shared memory by means of atomic read-modify-write instructions.

(b) Scalability consideration

The multiport page-memory is a hardware component that has a scalable property. We tried to keep the hardware's scalability within the tolerance of the conventional shared memory's bandwidth. For example, the control software is not placed on a special (centralized) control processor. Instead, any processing element can become the control processor in a unit of a transaction. When a transaction is received from a *PSI* machine, an idle PE is assigned to be the master of that transaction. The transaction master takes care of the compilation, parallel command generation, and response generation of that transaction. Parallel command execution is a task for multiple PEs (possibly including the transaction master PE). In that sense, parallel processing is applied toward (1) inter-transaction and (2) parallel command execu-

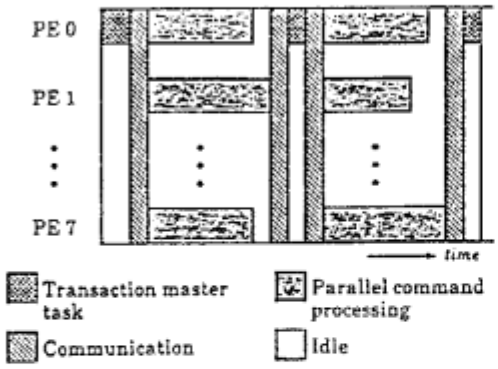

Figure 11. A parallel processing timing diagram

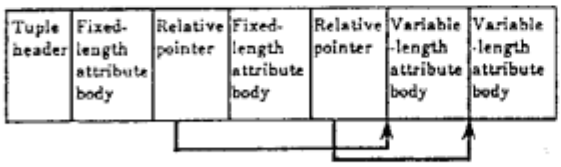

Figure 12. Representation of variable-length records

tion levels. Figure 11 shows a timing diagram of query processing where parallelism in the command execution level is realized. In this figure, PE0 is the transaction master and takes care of the master's tasks. This is a set of serialized operations performed intermittently between parallel command executions. The parallel command execution is done by idle processors as shown in Figure 11.

Term Data Type Support From software's point of view, relational knowledge base support is (1) the addition of a data type (term) and (2) the addition of a set of operations to relational database enhanced with the term data type. To do these, the basic data structure supports tagged data and variable length records, which is required because the term relational model allows variance of atomic and structured data as in Prolog. The structure of a record that supports variable-length record is shown in Figure 12.

Efficiency Consideration In database machine research, the importance of elimination of software overheads is often stressed. The software system has been designed and coded with this clearly in mind. The system owes the file system and the software development environment to the residing operating system. However, the rest of the software was made from scratch. To develop so much new software was expensive, but helped to make a specialized, compact and efficient system. For example, the control software of the PE is a single-process program and there is little overhead in switching between transac-

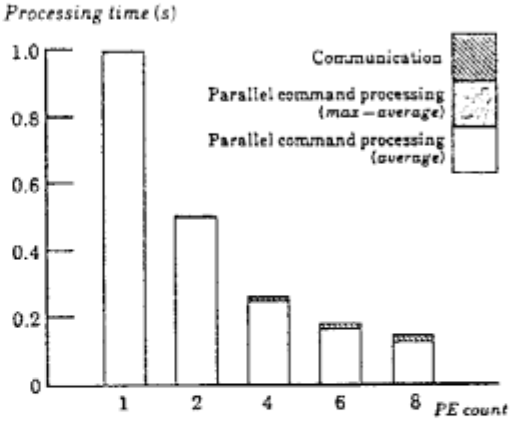

Figure 13. Performance of the selection operation

tion master tasks and parallel command execution tasks. Considering the nature of the system and preliminary evaluation results, we are convinced that this has been a good choice. We note that there are numerous decisions we took that have to undergo further evaluation.

4.4 Evaluation

So far, we have made a preliminary performance evaluation. This evaluation was to obtain the basic speed of the hardware and the efficiency of the parallel processing method, not to discover the final performance values.

The queries we took were selection and join operations. The selection query selects 111 tuples from 1600-tuple relation, the size of which is 500 KB. The join is performed between a 15 KB, 111-tuple relation, the result of the previous selection, and a 20K-byte, 215-tuple relation. A nested-loop algorithm is used. The result is 37 tuples. Note that the tuples are variable-length and, according to the parallel processing scheme, the query is processed as shown in Figure 11.

Figure 13 shows the result of the selection. The total processing time is almost identical to the time for parallel command execution. The overhead of parallel execution (in this case, communication time) is not recognized until the number of participating processors reaches six. Still the overhead is quite low. The effect of parallel processing is thus satisfactory, at least within the machine's degree of parallelism.

Figure 14 shows the result of the join. In contrast to the selection case, the total processing time of the join saturates at the processor count of six. In this case also, the effect of parallel command execution is good. However, the overhead increases as the number of processors increases. The source of overhead is the variance in the processing times of PEs. The communication time is hidden because the absolute processing time is about ten times greater than in the case of selection.

This phenomenon is clearly illustrated by comparing

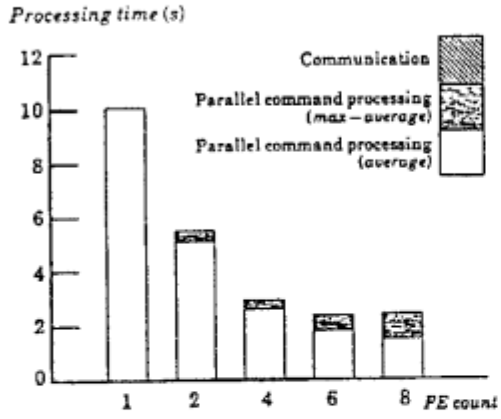

Figure 14. Performance of the join operation

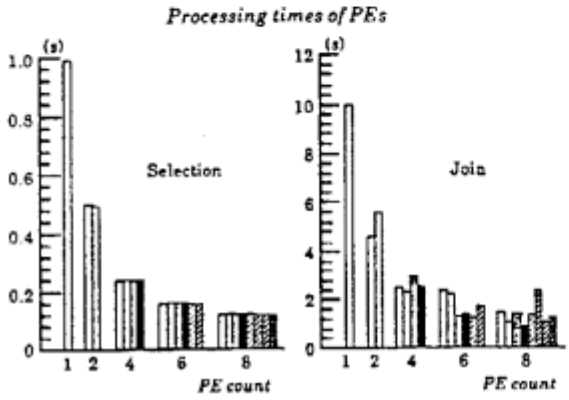

Figure 15. Comparison of processing times

the processing times of PEs in selection and join cases (Figure 15). The reason why there is variance in the join is because the size of the source relation is not large enough to be evenly shared by the PEs. The 20 KB relation (ten 2K pages) is divided by eight PEs, so two PEs have to process two pages while the remaining six only have to process one page each.

This evaluation is done using the first version of software where there are neither indexing schemes nor clustering schemes. The hashing based indexing scheme and, for join operation, bucket-wise hash-join method [Kitsure 83a] is being implemented. We leave more detailed evaluations for the future.

5 INTERFACE BETWEEN GHC AND PARALLEL KNOWLEDGE BASE SUBSYSTEM

The knowledge base subsystem should retrieve information quickly from a large amount of knowledge and treat a variety of knowledge objects uniformly. Then, it should manipulate the retrieved knowledge elements efficiently. The goal of the FGCS project is to build a knowledge information processing system using logic programming paradigms. Combining a parallel logic programming language and a dedicated system for operating a knowledge base seems to be one possible way to implement applications of FGCS project.

This section describes interfaces that combine a parallel logic programming language and a knowledge base system.

5.1 Overview of the System

Retrieval-by-unification (RBU) operations have been proposed [Yokota 86b] as the dedicated system for operating a knowledge base. RBU operations are an extension of relational database operations for manipulating the variety of knowledge objects. A knowledge element is represented by a term, a well-defined structure capable of handling variables. A knowledge base consists of sets of terms called term relations. The RBU system searches the term relations for desired terms, those unifiable with a search condition. We have implemented two extended relational algebra operations: unification restriction stream (*urs*) and unification join stream (*ujs*). Other conventional retrieval operations, such as union, projection, join, and selection, and updating operations, such as insert and delete, have also been implemented.

Guarded Horn Clauses (GHC) [Ueda 85], a parallel logic programming language with committed choice semantics, is the kernel language of the FGCS. It handles parallel processes and streams for communication among processes efficiently, but is inadequate in searching for alternative knowledge elements, since a variable of GHC can be assigned only once. GHC also has trouble handling global information such as that in knowledge bases. GHC has no appropriate means of guaranteeing the consistency of knowledge bases during parallel updating.

RBU enables GHC to process knowledge bases. RBU commands for retrieving and updating term relations are issued from parallel problem-solving systems written in GHC. A term relation is used to control consistency in parallel operation. The combination of GHC and RBU is useful in many types of knowledge information processing system for the FGCS project.

5.2 Parallel Retrieval

Now, consider production (rule-based) systems checking for feasibility of the combination of GHC and RBU. The basic concept of a production system involves applying state transition production rules from an initial state to reach a goal state that satisfies termination conditions. Several states can be generated from a single state by applying the production rules, and the state transitions make a search tree. The goal of a production system is to derive a path from the initial state to a goal state by traversing the search tree.

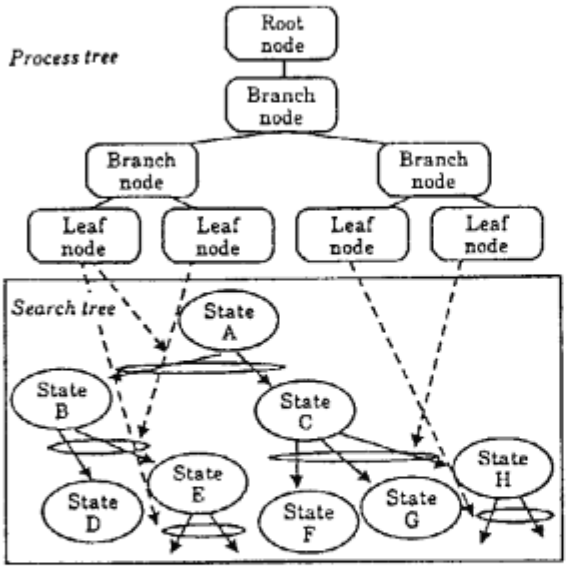

Figure 16. Process configuration and a search tree

Parallel processing is viewed as a way of reducing the large amounts of time consumed by production systems [Gupta 87]. One implementation is the parallel traversal of a search tree in which new states are generated from different states in parallel. Limits on memory and the number of processors require the use of special search strategies. The best first search [Barr 81] is one such strategy. It selects a state from a search tree using state evaluation of the current state to generate new states. The state selected has the best evaluation value in the tree at a given time. The centralized control of this strategy makes finding the best value a bottleneck, however. Control must be localized for efficient parallel processing. We propose a new search strategy called the Better First Search. The strategy looks only in a subtree of the search tree for the state that has the best evaluation value. Although this value is good, it may not be the best in the entire tree; we call it a "better" value.

We use a tree structure as the process configuration to implement the Better First Search in parallel. The tree configuration is not directly related to the search tree traversed by the production system. The three types of nodes (processes) in the process tree are the root node, leaf nodes, and other branch nodes. Productions are performed at the leaf nodes. Production priorities are controlled at the branch nodes based on their evaluation values. System control such as that of the user interface is performed at the root node. Figure 16 shows the process configuration and a search tree.

Nodes in the process tree are implemented using perpetual processes generated from recursively called GHC clauses. Process behavior is controlled by streams bound to variables in arguments in the clauses. The streams are treated as messages for the process. This configuration

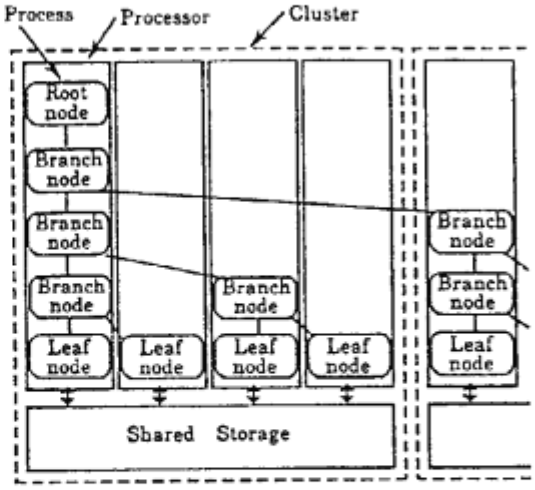

Figure 17. Implementation on the parallel model

is suitable for the parallel model of knowledge base machine mentioned in Section 4. A number of processors and shared storage compose a cluster in this machine, making it important to localize processor communications. We plan to locate each leaf process in a processor (Figure 17).

5.3 GHC Interface

A production is performed by retrieving knowledge elements from a knowledge base and updating the knowledge base based on production rules. The knowledge base is a global state for parallel production processes. GHC cannot handle global states among perpetual processes, nor effectively retrieve and update the knowledge base, even if a common stream is prepared as an argument of every clause to implement a global state in GHC. The unification implemented in GHC cannot be used to search for multiple knowledge elements, because a GHC variable can only be assigned a value once. Once bound to a knowledge element, the GHC variable's binding cannot be changed.

Connecting GHC to a dedicated system that processes knowledge bases enables a parallel production system to be built. RBU knowledge elements are terms defined in the same first-order logic as GHC, thus eliminating syntactical transformation. RBU stores a set of terms as a term relation which is used to guarantee the consistency in knowledge bases during parallel updating.

The special predicate `rbu(C)` is provided in GHC to enable the use of RBU. Commands for retrieving and updating knowledge bases are bound to the stream argument `C`.

For example:

```
C = [urs(tr1,[1],p(a,$(1)),[1],X),
```

$$\texttt{ujs(tr1,[2],tr2,[1],[3],Y),}\cdots].$$

The first command sentence, `urs( tr1, [1], p(a,$(1)), [1], X)`, dictates a search of the first attribute of the term relation tr1 for terms unifiable with the condition `p(a,$(1))`, yielding the derivation of the first attribute as a result. Results are returned as a stream bound to the variable X in the command sentence:

$$\texttt{X} = [\texttt{p(a,g(\$(2))),p(a,g(b)),}\cdots].$$

The second command sentence, `ujs( tr1, [2], tr2, [1], [3], Y)`, is used to derive the third attribute of a result relation generated by a unification join operation which searches the second attribute of `tr1` and the first attribute of `tr2` for unifiable terms. Results are returned bound to the variable Y.

$$\texttt{Y} = [\texttt{q(\$(10),c)},\cdots]$$

The special function symbol `$` is used to indicate a variable in command sentences and in results. GHC variables cannot be used for knowledge retrieval, so other symbols are needed to indicate variables for retrieval. These variables are bound to knowledge elements in RBU, but unbound in GHC. This corresponds to unbound variables appearing in a template predicate of the `setof` predicate in Prolog systems.

5.4 Implementation of RBU

Different approaches have been proposed to improve retrieval speed. One approach was to use dedicated hardware: for example, a unification engine was proposed by [Morita 86] [Yokota 86b]. [Ohmori 87] proposed a hash vector for indexing clauses. Superimposed code words for terms and a dedicated engine for manipulating the words were proposed by [Wada 88]. We use indexing that retrieves a set of terms by unification and backtracking. Retrieved terms resemble each other somewhat because they are unifiable with the search condition. For efficient backtracking, these terms must be located near an index. The trie is a type of tree structure that shares identical elements [Knuth 73] and meets this requirement. Figure 18 gives an example of a trie for a set of terms.

The costs of unification are proportional to the count of comparisons between components of the object terms. A trie reduces the number of comparisons when unification is performed. For example, consider what happens when the set of terms in Figure 18 is searched for terms that can be unified with the condition `p(f(a,b),h(c))`. Using the trie structure, the component p is compared only once, whereas four comparisons are necessary if the trie structure is not used. Using the trie structure, 10 comparisons are needed to search for all terms unifiable with the condition; 18 comparisons are needed if the trie structure is not used.

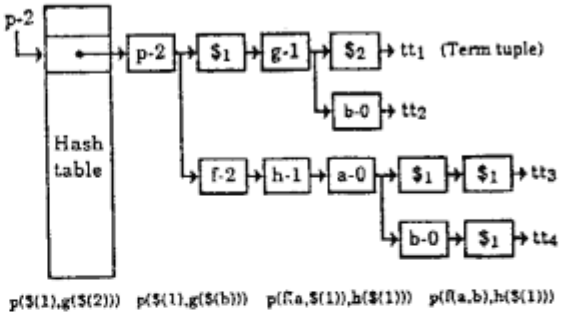

Figure 18. Tuple index with hashing and trie structure

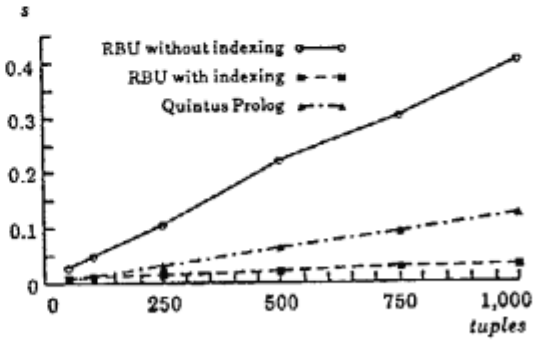

Figure 19. Comparison of search speeds

A hash table is used before the trie structure when storing many types of terms in a term relation (Figure 18). The first components of terms are used as hash entries. The trie structure is combined with hash collision resolution.

We compared the search and updating speeds of the RBU prototype with those of the Quintus-Prolog interpreter. Prolog compilers do not support assert and retrace predicates, (they cannot update knowledge bases), so the compiler has not been examined. Figure 19 compares the search speeds of the Prolog interpreter and *urs* with and without indexing. The *urs* without indexing is about four times slower than the Prolog clause search. This search time increases with tuple count in both Prolog and *urs* without indexing. However, the search time of *urs* with indexing scarcely increases regardless of the number of tuples. For 1000 tuples, it is about one-fourth of the time that a Prolog clause search would take. This is a result of the indexing.

Figure 20 compares the tuple insertion speeds of the two systems. Tuple insertion using RBU takes only about one-sixth the time of a Prolog *consult* operation. The overhead for making an index for a term relation is about one tenth of the insertion time.

6 CONCLUSION

In this paper, we have described the current status of research and development concerning the knowledge base

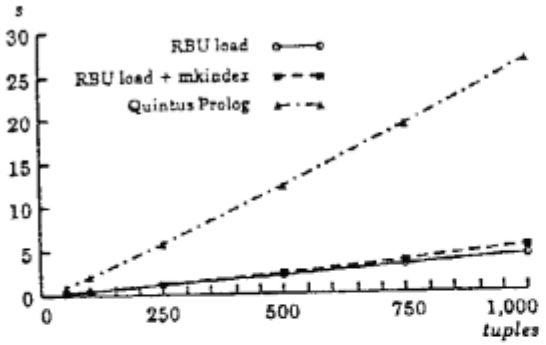

Figure 20. Insert speed comparison

subsystem in FGCS project. In the intermediate stage, we have investigated and experimented on the following four knowledge base mechanisms required for constructing the prototype of the FGCS.

(1) The knowledge base system developed on the *CHI* machine.

The knowledge base system on the *CHI* machine provides a very high performance knowledge-retrieval mechanism, a practical memory-based knowledge database, and a hierarchical clause database for a multi-process environment. In the system, multiple-multiple name spaces play an essential role in avoiding interprocess name conflicts and in hierarchical knowledge representation. The system will be a good vehicle for the next knowledge base research project.

(2) The distributed knowledge base system based on deductive databases.

A distributed deductive database system has been developed. It uses *PSI* machines connected by *ICOT-LAN*. The query processing strategy of the system is based on a bottom-up approach combined with query transformation procedures. A dynamic optimization method is used to process distributed queries. Dedicated hardware for processing indices has also been designed based on a superimposed code scheme for efficient knowledge base processing.

(3) The parallel knowledge base system.

The total system with the experimental hardware and knowledge base management software has been developed. The system can manipulate sets of terms efficiently in parallel. The hardware configuration proved useful for knowledge base purposes. The system connects to *PSI* machines, and a powerful unification-based query language has been developed as an interface.

(4) The knowledge base interface system for parallel logic programming languages.

We proposed to introduce a parallel logic programming language interface into a dedicated knowledge base system. We considered a parallel production system to check the feasibility of the combination of RBU and GHC. Parallel processes for the production system are implemented by perpetual processes written in GHC. Each process issues RBU commands for retrieving knowledge. We also outlined the concept for interfacing RBU with GHC using streams, and evaluated the search and updating speed of our RBU prototype.

The various kinds of technology developed in this stage will be incorporated into the FGCS prototype.

ACKNOWLEDGMENT

We would like to express our gratitude to the other members of the third laboratory of the ICOT Research Center. Each system described in this paper has been developed with the close co-operation of manufacturers. Thanks goes also to the manufacturers' people who were engaged in the implementations. We are indebted to the members of the KBM Working Group for their fruitful discussions.

References

[Apt 88] Apt, K.R., Blair, H.A. and Walker, A., "Toward A Theory of Declarative Knowledge", Minker (ed.), in *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, 1988

[Atarashi 88] Atarashi, A., Yanagida, S. and Konagaya, A., "SUPLOG Reference Manual", 1988 (In Japanese)

[Balbin 87] Balbin, I. and Ramamohanarao, K., "A Generalization of the Differential Approach to Recursive Query Evaluation", J. Logic Programming, Vol.4 No.3, 1987

[Bancilhon 86] Bancilhon, F., Maier, D., Sagiv, Y. and Ullman, J.D., "Magic Sets and Other Strange Ways to Implement Logic Programs" 5th ACM PODS, 1986

[Barr 81] Barr, A. and Feigenbaum, E. A., in *The Handbook of Artificial Intelligence*, 1, William Kaufmann, Inc. 1981

[Chikayama 84] Chikayama, T., "Unique Features of ESP", in *Proc. Int. Conf. Fifth Generation Computer Systems*, pp.292-298, 1984

[Demurjian 86] Demurjian, S.A. and Hsiao D.K., "A Multibackend Database System for Performance Gains, Capacity Growth and Hardware Upgrade", in *Proc. Int. Conf. on Data Engineering*, pp.542-554, 1986

[DeWitt 86] DeWitt, D.J., Gerber, R.H., Graefe, G., Heytens, M.L., Kumar, K.B. and Muralikrishna, M., "GAMMA - A High Performance Dataflow Database Machine", in *Proc. 12th Int. Conf. Very Large Databases*, pp.228-237, 1986

[Doolittle 86] Doolittle, R. F., "Of Urfs and Orfs, A Primer on How to Analyze Derived Amino Acid Sequences", University Science Books, Mill Valley, CA, 1986

[Gelfond] Gelfond, M. and Przymusinska, H. and Przymusinski, T., "On the Relationship between Circumscription and Negation as Failure", to appear in *Journal of Artificial Intelligence*

[Goto 87] Goto, A., "Parallel Inference Machine Research in FGCS Project", in *Proc. of the US-Japan AI Symposium 87*, pp. 21-36, 1987

[Gupta 87] Gupta, A., in *Parallelism in Production Systems*, Morgan Kaufmann Publishers, Inc.,1987

[Habata 87] Habata, S., Nakazaki, R., Konagaya, A., Atarashi, A. and Umemura, M., "Co-operative High Performance Sequential Inference Machine: CHI", in *Proc. ICCD'87*, New York, 1987

[Itoh 87] Itoh, H., Sakama, C. and Mitomo, Y., "Parallel Control Techniques for Dedicated Relational Database Engines", in *Proc. 3rd Int. Conf. Data Engineering*, pp.208-215, 1987

[Itoh 88] Itoh, H., Takewaki, T. and Yokota, H., "Knowledge Base Machine Based in Parallel Kernel Language", in eds. Kitsuregawa and Tanaka, in *Database Machines and Knowledge Base Machines*, Kluwer Academic Publishers, 1988

[Kakuta 85] Kakuta, T., Miyazaki, N., Shibayama, S., Yokota, H. and Murakami, K., "The Design and Implementation of Relational Database Machine Delta", in *Proc. Int. Workshop on Database machines '85*, 1985

[Kemp-Topor 88] Kemp, B.D. and Topor, W.R., "Completeness of a Top-down Query Evaluation Procedure for Stratified Databases", Dept. of Computer Science, Univ. of Melbourne, Technical Report, 1988, also in *Proc. 5th Int. Conf. and Symp. on Logic Programming*

[Kitsure 82] Kitsuregawa, M., Tanaka, M. and Motooka, T., "Relational Algebra Machine GRACE", *Lecture Notes in Computer Science*, Springer-Werlag, pp.191-214, 1982

[Kitsure 83a] Kitsuregawa, M.,Tanaka, M. and Motooka, T., "Application of Hash to a Data Base Machine and Its Architecture", in *New Generation Computing*, OHMSHA, 1, 1983

[Knuth 73] Knuth, D. E., "The Art of Computer Programming", 3, Sorting and Searching, Addison-Wesley, 1973

[Konagaya 87] Konagaya, A., Nakazaki, R. and Umemura, M., "A Co-operative Programming Environment for a Back-end Type Sequential Inference Machine CHI", in *Proc. Int. Workshop on Parallel Algorithms and Architectures*, East Germany, pp.25-30, 1987

[Konagaya 88] Konagaya, A., "Implementation and Evaluation of a Fast Prolog Interpreter", in IPS Japan SIG-SYM 46-4, 1988 (in Japanese)

[Kunifuji 82] Kunifuji, S. and Yokota, H., "Prolog and Relational Database for Fifth Generation Computer Systems", in *Proc. Workshop on Logical Bases for Data Bases*, Gallaire, et al.(eds.), ONERA-CERT, 1982

[Minsky 74] Minsky, M., "A Framework for Representing Knowledge", MIT AI Memo No.306, 1974

[Miyazaki 88a] Miyazaki, N., Haniuda, H. and Itoh, H., "Horn Clause Transformation: An Application of Partial Evaluation to Deductive Databases", in *Trans. IPSJ*, Vol.29, No.1, 1988 (in Japanese)

[Miyazaki 88b] Miyazaki, N., Haniuda, H., Yokota, K. and Itoh, H., "Query Transformations in Deductive Databases", ICOT-TR 377, 1988

[Monoi 88a] Monoi, H., Morita, Y., Itoh, H., Sakai, H. and Shibayama, S., "Parallel Control Technique and Performance of an MPPM Knowledge Base Machine Architecture", in *Proc. 4th Int. Conf. Data Engineering*, pp.210-217, 1988

[Monoi 88b] Monoi, H., Morita, Y., Itoh, H., Takewaki, T., Sakai, H. and Shibayama, S., "Unification-Based Query Language for Relational Knowledge Bases and its Parallel Execution", in *Proc. Int. Conf. Fifth Generation Computer Systems*, 1988

[Morita 86] Morita, Y., Yokota, H., Nishida, K. and Itoh, H., "Retrieval-By-Unification Operation on a Relational Knowledge Base", in *Proc. of 12th Int. Conf. on Very Large Databases*, pp. 52-59, 1986

[Morita 88] Morita, Y., Itoh, H. and Nakase, A., "An Indexing Scheme for Terms using Structural Superimposed Code Words", ICOT TR-383, 1988

[Murakami 83] Murakami, K., Kakuta, T., Miyazaki, N., Shibayama, S. and Yokota, H., "Relational Database Machine: First Step to a Knowledge Base Machine", in *Proc. 10th int. symp. Computer Architecture*, pp.423-426, 1983

[Ohmori 87] Ohmori, T. and Tanaka, H. "An Algebraic Deductive Database Managing a Mass of Rule Clauses", in *Proc. of 5th Int. Workshop on Database Machines*, pp. 291-304, 1987

[Sakai 88] Sakai, H., Shibayama, S., Monoi, H., Morita, Y. and Itoh, H., "A Simulation Study of a Knowledge Base Machine Architecture", in *Database Machines and Knowledge Base Machines*, Kluwer Academic Publishers, pp.585-598, 1988

[Sakama 87] Sakama, C. and Itoh, H., "Partial Evaluation of Queries in Deductive Databases", Workshop on Partial Evaluation and Mixed Computation, 1987

[Seki 88] Seki, H. and Itoh, H., "A Query Evaluation Method for Stratified Programs under the Extended CWA", ICOT Technical Report TR-337, 1988, also in *Proc. 5th Int. Conf. and Symp. Logic Programming*

[Shibayama 87] Shibayama, S., Sakai, H., Monoi, H., Morita, Y. and Itoh, H., "*Mu-X*: An Experimental Knowledge Base Machine with Unification-Based Retrieval Capability", in *Proc. France-Japan Artificial Intelligence and Computer Science Symposium 87*, pp.343-357, 1987

[Taguchi 84] Taguchi, A., Miyazaki, N., Yamamoto, A., Kitakami, H., Kaneko, K. and Murakami, K., "INI: Internal Network in the ICOT Programming Laboratory and its Future", in *Proc. of 7th ICCC*, 1984

[Takasugi 87] Takasugi, T., Haniuda, H., Miyazaki, N. and Itoh, H., "Distributed Query Processing in KBMS PHI", in *IPS Japan SIG-MDP*, 34-9, 1987 (in Japanese)

[Tamaki 86] Tamaki, H. and Sato, T., "OLD Resolution with Tabulation", in *Proc. of 3rd ICLP*, 1986

[Tanaka 84a] Tanaka, Y., "MPDC: Massive Parallel Architecture for Very Large Databases", in *Proc. Int. Conf. Fifth Generation Computer Systems*, pp.113-137, 1984

[Tanaka 84b] Tanaka, Y., "A Multiport Page-Memory Architecture and A Multiport Disk-Cache System", in *New Generation Computing*, OHMSHA, 2, pp.241-260, 1984

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

[Van Gelder 86] Van Gelder, A., "Negation as Failure Using Tight Derivations for General Logic Programs", in *Proc. 1986 Symp. on Logic Programming*, IEEE Computer Society, pp. 127-138, 1986, also to appear in *Journal of Logic Programming*

[Wada 88] Wada, M., Morita, Y., Yamazaki, H., Yamashita, S., Miyazaki, N. and Itoh, H., "A Superimposed Code Scheme for Deductive Databases", in eds. Kitsuregawa and Tanaka, in *Database Machines and Knowledge Base Machines*, Kluwer Academic Publishers, 1988

[Yokota 84] Yokota, H., Kunifuji, S., Kakuta, T., Miyazaki, N., Shibayama, S. and Murakami, K., "An Enhanced Inference Mechanism for Generating Relational Algebra Queries", in *Proc. 3rd ACM SIGACT-SIGMOD Symp. Principles of Database Systems*, pp.229-238, 1984

[Yokota 86a] Yokota, H., Sakai, K. and Itoh, H., "Deductive Database System Based on Unit Resolution", in *Proc. 2nd Int. Conf. Data Engineering*, pp.228-235, 1986

[Yokota 86b] Yokota, H. and Itoh, H., "A Model and an Architecture for a Relational Knowledge Base", in *Proc. 13th Int. Symp. Computer Architecture*, pp.2-9, 1986