TR-311

# An Experimental Knowledge Base Machine with Unification-Based Retrieval Capability

by
S. Shibayama, H. Sakai, H. Monoi
Y. Morita & H. Itoh

October, 1987



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

# An Experimental Knowledge Base Machine with Unification-Based Retrieval Capability

Shigeki Shibayama, Hiroshi Sakai (Toshiba Corporation)
Hidetoshi Monoi, Yukihiro Morita, Hidenori Itoh (ICOT)

Abstract

This paper describes how one of ICOT KBM research projects has progressed to reach experimental knowledge base machine implementation. To enhance the expressiveness of the relational database model,a term relational model was introduced and a machine architecture for manipulation of that model was proposed. A simulation study on the architecture was carried out and the selection of a parallel control strategy was found to be important for effective use of the multiport page-memory and unification engines,which are hardware components of the architecture. The experimental hardware for this architecture is then described concerning its objectives, hardware configuration,and control software. The future plans and points of concern for the evaluation of the architecture concludes the paper.

## 1. Introduction

As computer isage increase its categories of knowledge information processing, it becomes more and more important to extract, store and manage human knowledge in computer manipulable form. One such example is the expert system approach, which stores the knowledge in the form of production rules.

From another point of view, databases are considered as sources of knowledge when data retrieval is done. In JapanUs Fifth Generation Computer Systems project, we have been engaged in the study of knowledge base machines as an enhancement of database machines/systems. The relational model was selected as a first step toward a knowledge base model [Murakami 83]. As the other (and main) aim of the project is to prove that the logic programmming can be a unifying principle of future computing systems, it is assumed that the knowledge base machine have good affinity to the logic programming languages and systems.

The term-relational model [YokotaH 86] is proposed as a knowledge base model. It is characterized by the incorporation of terms as an object type and definition of a set of operations, based on unification, on term objects. Other efforts which have similar objectives to us are, for example, given in [Ait-Kaci 85], [Bancilhon 86], [Tsur 86], [YokotaK 87]. We think this model is appropriate for our knowledge base model because (1) it can represent complex objects in a natural way, (2) it can perform a breadth-first deduction and (3) it has an affinity with the logic programming environment in that it is common to retrieve terms with unifiability.

Though the last reason is somewhat proper to our standpoint, it is applicable to other programming environments by considering unification as a pattern-matching criteria.

In section 2, the expressiveness of a term relational model is described and some additional operators are introduced. In section 3, simulation results on an architecture of a knowledge base machine is described and in section 4 an experimental knowledge base machine based on the simulated architecture is described. Section 5 is the conclusion and future plans.

## 2. Relational Knowledge Base

### 2.1 Term Relational Model

Knowledge base and database technologies have much in common, for example, access methods for data/knowledge items, management of data with secondary storage, parallel processing strategy, access control and so on. We thought that it is a good idea to start with the relational database systems, where the relational model [Codd 70] being the basis of a knowledge base model. However, as our research progresses, it is recognized that some enhancements are necessary to the relational model to be used as a knowledge base model. Because we usually want to make an object in a knowledge base self-descriptive, we often come up with a structured data object. For example, a natural way of representing parent relationship is to use a predicate and write "parent(son-of-a-person, person)". If we want to represent unknown (at the time of query) information, we could use a variable and write "parent(son-of-a-person, X)" where X is a variable and represents the parent of "son-of-a-person". It is awkward, but not impossible, to store such objects in a relational database. Syntactically, they are "terms" follows:

(1) atomic symbols and variables are terms
(2) if "f" is an n-place functor symbol and x1 to xn are terms, then f(x1,x2,...,xn) is a term
(3) Terms are generated only applying above rules

Terms could be stored in a relation whose name corresponds to the predicate symbol. The arguments of the predicate are stored in character strings in corresponding attributes. However, there are several disadvantages in this scheme. First, this scheme cannot store variables with its original semantics. A reltion attribute may contain a character string starting with an uppercase letter (denoting variables), but that is not recognized as a variable by the RDBMS. Second, there is no way of storing a dynamically-generated data structure which is not defined at the time of the database design. As it is common and easy in manipulating a knowledge base to generate a new term at execution time, this is a significant disadvantage. (The list structure is the typical case; the length of a list structure changes frequently.) Third, it is impossible to retrieve an object using complex pattern-matching.

To overcome these disadvantages, a new knowledge base model was proposed [YokotaH 86]. This model permits terms as objects contained in relations. There are several features obtained by introducing terms as a data object. Some of those are:

(1) Complex objects, either statically or dynamically defined, can be represented.
(2) Nested relations can be represented.
(3) A kind of inference (breadth-first input resolution) can be performed

Each of the features are explaned below. ( The syntax of the examples follows that of Dec10 Prolog. )

(1) As an example of a complex object, the following term shows the configuration of a simple computer.

```
"computer
  ([cpu([system_bus, internal_bus]),
    cache([system_bus, internal_bus]),
    main_memory([system_bus]),
    io_channel([system_bus, io_bus]),
   disk_controller([io_bus]),
   display_controller([io_bus]),
   disk_drive([io_bus]),
   diplay([io_bus])])"
```

This term represents a computer consisting of a cpu which is connected to the system bus and the internal bus, a cache memory which is connected to the system bus and the internal bus, a main memory which is connnected to the system bus,

and so on. We could write "cpu(68020, 12.5MHz, [system_bus, internal_bus])", "cache(64KB[system_bus,internal_bus])" and so on for explaining the details of the components.

As the nature of the information of this kind, the structure of each data is varied, and the addition of a computer component (mouse, for example) results in the generation of a new term as shown below.

```
"computer
   ([cpu([system_bus, internal_bus]),
     cache([system_bus, internal_bus]),
     main_memory([system_bus]),
     io_channel([system_bus, io_bus]),
     disk_controller([io_bus]),
     display_controller([io_bus]),
     disk_drive([io_bus]),
     diplay([io_bus]),
     mouse([io_bus])])"
```

(2) Nested relations are considered to be a special case of complex objects; the schema of the inner relation is predefined. An example of an inner relation of a nested relation is as follows:

```
"career([programmer(tokyo,1978,1982),
 chief_programmer(tokyo,1983,1985),
 systems_analyst(osaka,1986,present)])"
```

This term means that a person worked from 1978 to 1982 as a programmer at Tokyo office, promoted and worked as a chief programmer from 1983 to 1985 there, then became a systems_analyst from 1986 to present with a move of office from Tokyo to Osaka. If the person becomes a manager in 1988, the above career term will be:

```
"career([programmer(tokyo,1978,1982),
   chief_programmer(tokyo,1983,1985),
   systems_analyst(osaka,1986,1987),
   manager(osaka,1988,present)])"
```

This corresponds to the insertion of a tuple into the inner relation in the "career" attribute of the primary relation.

To effectively retrieve data items represented by terms, extended query qualifications are required. In the "computer" example, if we want to know whether a floppy disk drive is included in the system configuration, a query like below is essential.

```
?-arg_member(floppy_disk_drive(_),
                computer).
```

The meaning of this query is to check whether the first argument of the built-in predicate arg_member (namely floppy_disk_drive(_) ) is a member of the arguments of the functor at the second argument (computer).

A Prolog-like program to evaluate the above query might look like:

```
arg_member(X,Y) :-
     get_arg(Y,Arg), member(X,Arg).
```

where "get_arg" is a special (extralogical) built-in predicate to get the argument of the functor unified to Y, and "member" is the usual membership check predicate. Both of the functions these predicates provide are beyond the scope of normal relational database system. This indicates that it is needed to incorporate extended operations into the term relational model to make use of the power of term representation.

The most important operation primitive is unification. As we set up an interface that is highly logic programming oriented (described in subchapter 4.2), unification is a natural choice as a pattern-matching criterion. *Unification-join* and *unification-restriction* are defined by extending the usual qualification condition of equality to unifiability. The retrieval operations using unification are collectively called the *Retrieval-by-Unification (RBU)* operations. RBU operations are the basic mechanisms to perform the input resolution described below.

```
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
parent(smith,clark).
parent(clark,turner).
        ...
```

| [ancestor(X,Y) \| S] | [parent(X,Y) \| S] |
|---|---|
| [ancestor(X,Y) \| S] | [parent(X,Y),ancestor(Z,Y) \| S] |
| [parent(smith,clark) \| S] | S |
| [parent(clark,turner) \| S] | S |
| ... | ... |

Figure 1. Example of Term Relation

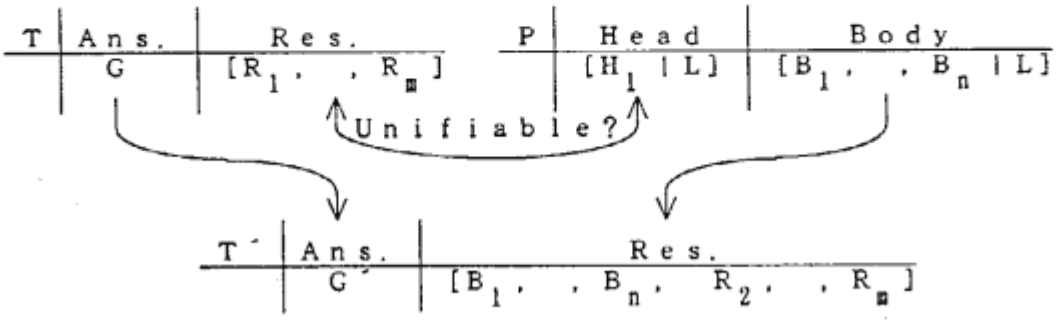



**Figure 2. Unification-join for One Step of Input Resolution**

(3) By permitting terms as data objects, Horn clauses (both facts and rules) can be stored as shown in Figure 1. By using a special list structure and unification-join, (somewhat tricky) input resolutions can be performed.

The process of a resolution is as follows. First, the query is unification-restricted to the first attribute (the head part of the Horn clauses) of the permanent term relation. The variable substitution is applied to the second attribute (body part) of the relation. In the term relational model, the scope of the variable is within a tuple, and the literal representation of variables is of no importance. A variable name is used only for specifying the identity of the variable from the other variables. (The term parent(X,Y) is identical to parent(P,Q) and so on). These correspond to the same rules with Prolog clauses (programs).

Second, the body of the result of the first unification-restriction (a source temporary term relation) is unification-joined to the head of the permanent term relation. (The same permanent term relation used in the first unification-restriction.) The resultant temporary relation consists of the head part of the source temporary relation and body part of the permanent relation. (Figure 2). By using the newly obtained temporary relation as the next source relation, this unification-join is repeated until no new results are obtained. One iteration of the unification-join corresponds to one step of resolution (reduction). As these operations are based on set operations, the (intermediate) results are obtained in sets, holding every resolvent at each resolution step.

### 2.2 An Architecture for Manipulating Term Relations

An architecture for performing the above resolution is proposed [Yokota 86], [Morita 86] (Figure 3). The *MPPM*, which stands for the *multiport page-memory*, is a memory system permitting conflict-free page access from multiple ports [Tanaka 84]. (The MPPM will be described in chapter 4.) There are multiple *unification engines (UEs)* connected to ports of the

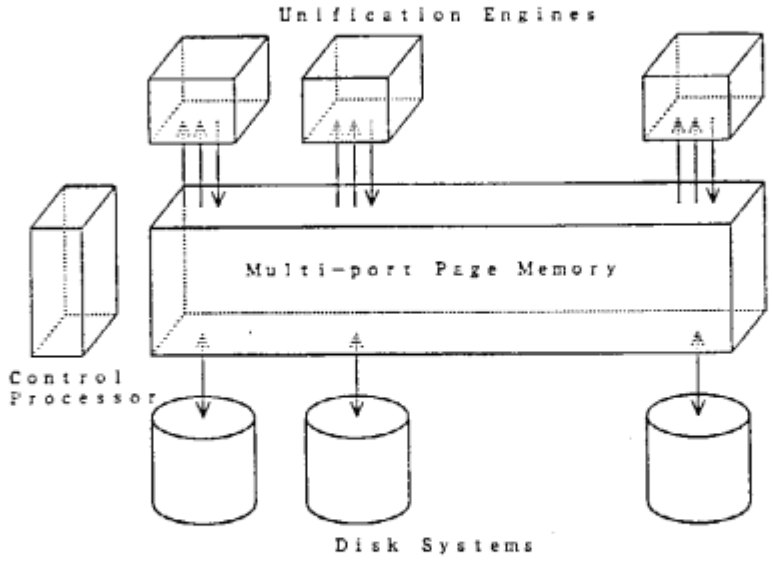
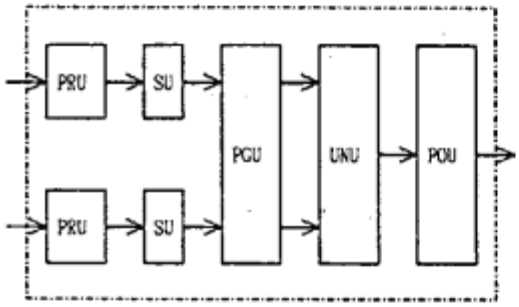



**Figure 3. System Configuration of a KBM**




PRU :Processing unit
SU :Sort unit
PGU :Pair generation unit
UNU :Unification unit
POU :PostProcessing unit

**Figure 4. Unification Engine Configuration.**

MPPM. UEs are pieces of dedicated hardware for performing unification-join and unification-restriction operations in a pipelined fashion. Following the relational database engine (RDBE) processing scheme ([Sakai 84], [Shibayama 84]), The UE is supplied with two input relations, unifies all the unifiable pairs between the two input relations and output the resultant relation back into the MPPM. A UE has separate sorters for both input ports, sorting the terms according to the generality of terms.(Figuure 4). The sorted two input relations are then merged in the pair-generator unit and possibly unifiable term pairs are supplied to the unification unit. Full unification is performed in the unification unit. The idea is that by providing pieces of dedicated (fast) hardware for unification operation and using them in parallel with a conflict-free memory system, a high performance machine capable of performing resolution upon the stored knowledge base is achieved. (Knowledge base in this context means a collection of term relations which contain both rules and facts.) We have conducted a simulation study for estimating the performance of this architecture, which is described in Chapter 3.

### 2.3 More Operation Primitives

#### 2.3.1 Unify-Check

As we have shown, unification is one of the necessary operation primitives to manipulate term relations. Unification applies, however, the unifier (variable bindings) to the terms being unified. So, for example, when we want to retrieve rules whose head can be unified to an instantiated term, unification-restriction cannot be used. To remedy this, we propose the unify-check opera-

tion primitive. We assign the "<=>" symbol to define the unify-check operator. The unify-check only tests unifiability of the arguments and does not apply the unifier. Thus, in the example below, the rule contained in "rule" relation is retrieved and forwarded to the predicates that follow.

..., rule(X,Y), X <=> t(P,q), ...

### 2.3.2 Generality-Compare

When we want to know, for example, if there is a rule (with variables) in a term relation identical to a given rule, neither literally-equal ("=="), unification ("=") nor unify-check ("<=>") can be used. To do this, the generality of terms should be compared. The generality of term is defined as:

> Between terms t and u, t is defined more general than u if and only if there is a substitution s such that st = u.

According to above rule, f(X,Y) is more general than f(3,Z). We introduce the "more-general" operator to represent the generality as:

f(X,Y) >> f(3,Z)

Less-general operator is obvious. When a term is more general than another term and vice versa, the generality of the terms are equal. This is denoted using "generality-equal" operator as follows:

f(X,Y) <<>> f(P,Q)

According to the definition, if the generality of a two term is equal, they can be made literally identical by appropriately renaming the variables of one term. Thus when two rules (having variables) are equal in generality, the two rules are, in fact, identical.

Note that there are many cases when generality order is not applicable. For example, we cannot decide the generality between f(X,3) and f(4,Y).

These additional primitives were found when we worked on the inference machine interface, which will be described in chapter 4.

## 3. A Simulation Study of the Architecture [Sakai 87]

To evaluate the characteristics of the input resolution on the proposed architecture, we performed a set of detailed simulations.

We began this simulation study to obtain knowledge about the behavior of UEs, because we wanted, at the first place, to examine how effective the proposed UE configuration was. So, a UE simulator was made. In this simulator, the UE is simulated at a register transfer level to exactly estimate the processing time. It showed us several bottlenecks within the UE configuration. Those drawbacks were, however, overcome by fine-tuning the architecture [Morita 87]. And those improvements are reflected to the simulator. Thus we think that the configuration of the UE is close to optimalas (as far as the same hardware algorithm is used).

The next step was to carry out the system simulation, that is, we wanted to know how effective the system architecture incorporating the UEs and the MPPM was. We decided to use the UE simulator as a component of the system simulation. If we could use some analytical model to estimate the processing time of a UE, we could have done without the extremely detailed, thus time consuming, model. However, it is impossible to know, before actual execution of unification-join, how many unifiable pairs are included in the input relations and how long the result becomes. Because those are most relevent to the processing time, we had to choose the most detailed model.

We did not include the control overhead time nor disk access time in the system simulation study. This is because we wanted to know the essential characteris-

tics of the nature of the operation.

The sample problems we adopted are ancestor-finding problem and 8-queen problem. The former is a typical operation of a breadth-first search with recursively defined rules. The nature of the former problem is such that the unification performed in the search is not a very complex one and that the unifiability filtering, done in the pair-generator section, works very well. When going upward along the family tree, parents of a person are at most two independent of the total parent-child relationships. The family tree is computer-generated under reasonable assumptions of human life cycle. About 1800 facts of parent-child relationship are used.

The latter 8-queen problem is another famous sample problem. In our case, however, the representation of the problem is somewhat special. Usually, generate and test type search is adopted to solve the problem. We coded the problem specifically to solve it using only unification-join and unification-restriction. The idea is to use common variables to propagate the constraint of next queen placement. This scheme resulted in a complicated set of rules, having 25 variables in the left hand side (head), if denoted by a Prolog rule.

The aim of the system simulation was to obtain knowledge about the systems behavior when executing repeated unification-joins for an input resolution. Soon after obtaining and examining some preliminary results, it is observed that the most influential to performance is the control method to assign jobs to UEs in parallel. As the UE has the performance close to optimal and the MPPM is conflict-free, there are almost no bottlenecks in hardware resources, once a job is assigned and executed. Taking secondary storage into account, the behavior would be different. It was, however, beyond the scope of this simulation. By the reasons as mentioned, we concentrate on the parallel control methods to make use of the hardware resources.

In evaluating the control methods, we selected the following measures in the simulation:

(1) Execution time Elapsed time of executing an input resolution

(2) Page loading factor As we use the page-based memory system, source relations and temporary relations are stored in units of pages. Page loading factor varies according to the control method.

(3) Performance Stability This shows the stability of execution time over variance of the tuple size or other system parameters.

Assuming that relations are stored in page units, the basic parallel control strategy is to divide the join into sets of smaller joins (over subsets of relation pages) and collect the results. As unification-joins are repeated using the temporary relation obtained in the previous unification-join, it is necessary to dynamically assign UEs to joins over permanent relation pages and newly created temporary relation pages.

One control method is generating join requests between single pages. Suppose that the permanent relation P consists of pages p1, p2, ..., pn and the temporary relation T consists of pages t1, t2, ..., tm, join requests Jij (n >= i >= 1, m >= j >= 1) for every combinatin of i and j are generated. We call this method the *SP (Single Page at a time)* method.

The simulation showed that when performing the repeated unification-join operations, the SP method is not advantageous for high performance. (Figure 5, Figure 6)

This is because the performance (processing time) of the UE depends much upon the size of the input relations, as the time to read them cannot be avoided from processing time. The total amount of data that must be transferred increases as we divide the joins to smaller pieces. For example, if the original join is divided into k**2 pieces (each input relation is divided

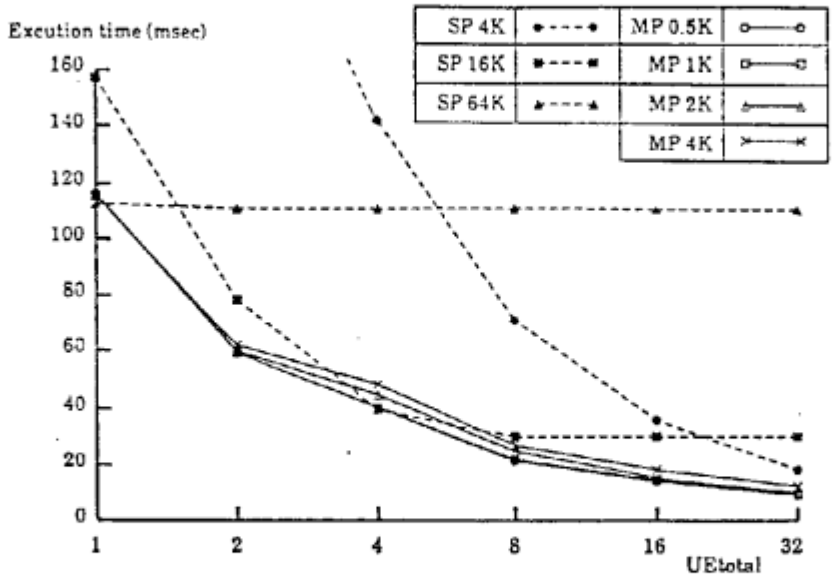



Figure 5 Relationship between *UEtotal* and Execution Time (Ancestor Problem)

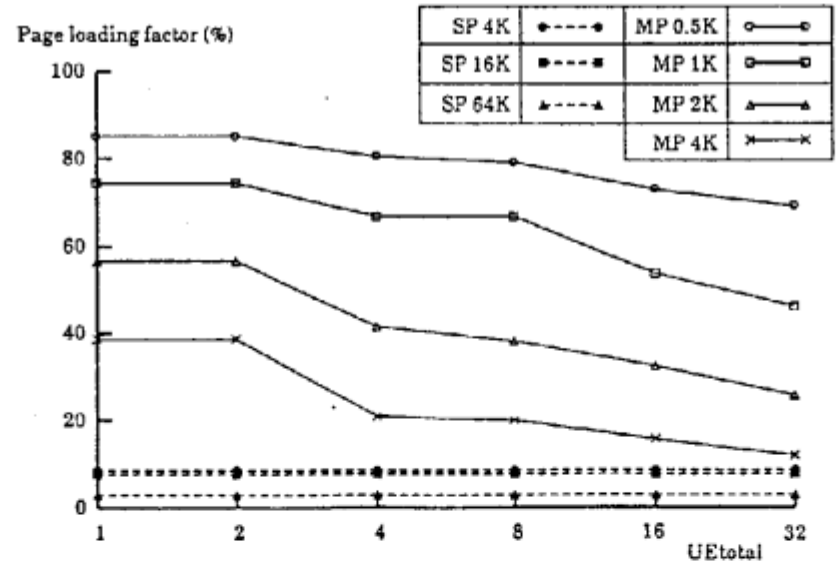



Figure 6 Relationship between *UEtotal* and Page Loading Factor (Ancestor Problem)

into k subrelations), the total amount of data to be transferred becomes k times larger. Even if there were k**2 UEs that can work in parallel, the performance improvement will be only k times.

Though the actual processing time does not depend only upon the amount of input data, this discussion is essential to the limitation of the SP method.

To improve the SP method, we adopted the *MP (Multiple pages at a time)*method. The MP method partitions relations according to their size and the number of UEs. Though joins are divided to smaller pieces as in the SP method, the size of each subre-

lation is not restricted to a single page. Thus the granule of join is larger than that of the SP method.

There are several parameters to tune the MP method. The interested reader may refer to [Sakai 87]. Oneimportant thing in this MP method is to know the degree of its parallelism. It is observed that, when generating new join requests, the requests should be about as many as the number of UEs. To generate too few join requests results in the increase of idle UEs. This is intuitively explained as follows. At first, all UEs are available and all the UEs are assigned jobs that are thought to have equal processing load. However, the actual processing load varies according to the content of the processed data, which results in the variance in processing time. At the end of the first UEUs job execution, if all the rest of the remaining job is assigned to the available UE (only one), the UE is loaded with too much of the job and the rest of the UEs, becoming free one after another, will have no jobs in the queue.

On the other hand, too many join requests makes the granules of join small as in the SP method.

Moreover, an interesting result was obtained by examining the MPPM port utilization ratio (Figure 7).

| | Ancestor | 8-queen |
|---|---|---|
| Port for TR | 4%-18% | 16%-23% |
| Port for PR | 36%-45% | 1%-14% |
| Port for output | 0.7%-2% | 7%-12% |
| Avarage | 16%-18% | 11%-13% |
| Only one port per a UE | 36%-49% | 25%-35% |

Figure 7. Port Utilization Ratio of the MPPM

The original UE occupies three MPPM ports, two for input and one for output. The port utilization ratio indicates how much of the potential data transfer capability is utilized in a resolution. The simulation showed that only a low fraction of the potential data transfer capability was used (0.7% to 40%). We modified the configuration to assign only one port to a UE. This resulted in the increase of only 30% of processing time with one third of MPPM ports. We are convinced that this configuration is practically superior to the three-ported UE.

## 4. An Experimental Knowledge Base Machine

### 4.1 Research Objectives

Although the processing times of original hardware configuration are obtained by a series of simulations, there are several impractical assumptions. Those are:

(1) there is no practical basis as to how much hardware is required to implement a UE
(2) a UE is single-purpose
(3) no control overhead is included.

For (1) and (2), we are considering more practical hardware for unification-based operations. This will be reported elsewhere. For (3), we think that control software is bound to be run, in a sense, on a general-purpose machine due to its complexity. (Here, general-purpose does not mean a von Neumann architecture.) Obviously, no data/knowledge base machine will be able to claim its high performance without an efficient control mechanism implementation. Thus the main objective of the research is to evaluate the hardware and software architecture through the experiments to design, implement and evaluate control software on a hardware prototype. The expected results would be improvements in hardware configuration, requirements to system software (operating system, in particular), require-

ments to secondary storage functionalities and proposal of an instruction set for efficient data/knowledge base control software implementation.

Another research objective is to find effective decentralized parallel control algorithms, under the given hardware resources, for the processing elements. We aim at decentralizing the control because we want to make the architecture scalable to a certain extent (say, several hundred).

For example, the MP method disclosed in the simulation study is found to be a good control strategy for assigning jobs to PEs in parallel, for it balaces well the PE usage and the amount of data transfer. Our next step is to try to implement it on the experimental knowledge base machine hardware in a most efficient way and find out any architectural hindrances.

Another objective is to investigate a good internal processing algorithm on operation primitives. Sorting algorithms are one of the best sought examples. This will be meaningful in both software and hardware implementation of the algorithm.

The last objective is to obtain experiment on the effectiveness of the interface for inference machines (described in the next subchapter). For an experiment, we will connect the experimental knowledge base machine to a PSI (Personal Sequential Inference machine) [Taki 84].

### 4.2 Inference Machine (PSI) Interface

The PSI is adopting ESP as its sole programming language. ESP is a language based on logic programming with the addition of object-oriented features [Chikayama 84]. Every ESP program is regarded as an object, providing methods as the interface to other objects. An object is either a class object or an instance (of a class) object. We decided to provide a kbm class (denoted as #kbm, where #indicates a class name), as the interface class to other application classes. The other way to establish an interface with the KBM is to add built-in interface predicates into ESP language, which requires modification of the ESP language processor and operating system (SIMPOS). Compared to the expected endeavor, this approach would not bring enough additional functionalities.

When a process in PSI wants to use KBM, it sends a message to #kbm object (calls a #kbm method). If the message is the first one from that process, an instance of kbm class is created. Each instance corresponds to a transaction, thus multiple transactions are supported. (Of course, it is the KBM that supports multiple transactions. An instance is the agent of a transaction in PSI.)

The #kbm class (instance of it, actually) provides KBM interface methods by inheriting them from the parent class. The interface is based on the relational calculus. The representative interface method is the :retrieve method. (A colon denotes a method call in ESP.) Some examples of :retrieve method call are shown below. (Some arguments of the method call are abbreviated for clarity. For example, the first argument, which must specify the callee object in ESP language, is abbreviated.)

```
:retrieve(r(X,Y,W),(f(X,Y),g(Y,W)))
```

Here, f and g are 2-attribute term relations. The second argument means that the second attribute of relation f be (unification-)joined to the first attribute of relation g. The comma between f(X,Y) and g(Y,W) is the conjunction connective. The resultant (temporary) relation is obtained as a 3-attribute temporary relation r, projecting out one of the join attributes. In this example, as in logic programming languages, unification is assumed by using the same variable name in the second argument (attribute) of relation f and in the first argument of relation g.

```
:retrieve(r(X,Y,W),
        (f(X,Y),g(U,W),Y==U))
```

This example differs slightly from the former example. In this case, relational equi-join between the second argument of relation f and the first argument of relation g is implied. The "==" operator denotes a "literally-equal" condition. If the operator is changed to "=", which indicates unifiability, this query becomes identical to the former one.

```
:retrieve(r(X,Y),(f(X,Y),X==3))
```

This query indicates a selection. Those tuples whose first attribute is 3 are selected to be a temporary relation r.

```
:retrieve(Temporary_relation, Query)
```

This is, as indicated by the above examples, the general syntax where Temporary_relation specifies the resultant temporary relation and the Query specifies, following the Prolog syntax, the query condition.

In the second chapter, more operation primitives are introduced. Below is an example of unify-check operator.

```
:retrieve(r(X,Y),
        (rule(X,Y), X <=> t(P,q)))
```

Similarly, a generality-compare operator can be used in the :retrieve method.

```
:retrieve(a(X,Y),
        (f(X,Y), X <<>> g(p,q)))
```

There are other KBM interface predicates provided as the methods of the #kbm class. Those are definition predicates, update predicates, input/output predicates and so on. A set of those predicates provides the logic programming oriented interface to applications in the inference machine PSI.

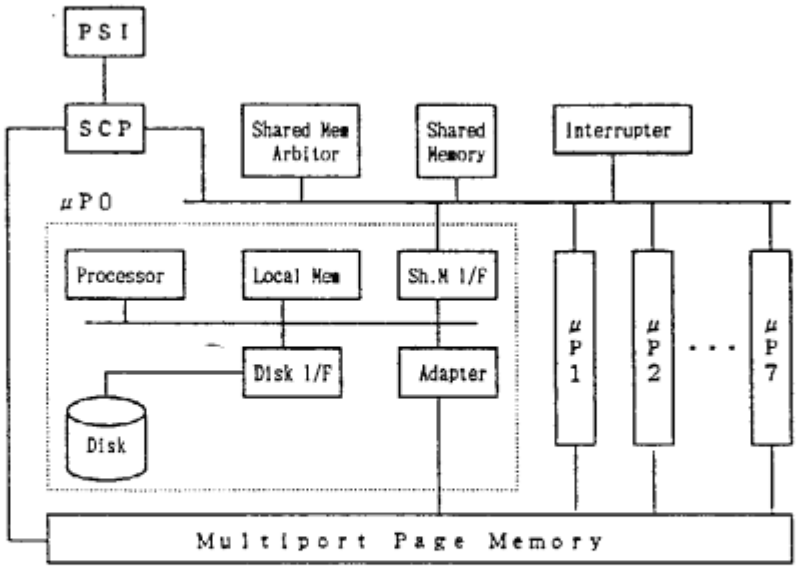



Figure 8. Configuration of Experimental KBM

### 4.3 Hardware Configuration

#### 4.3.1 General Configuration

The configuration of the experimental knowledge base machine is shown in Figure 8. It consists of an 8-port multiport page-memory (MPPM), processing elements connected to each port of the MPPM, and a control processor. The core of the processing element is the MC68020 microprocessor, operating at a 12.5MHz clock. Other PE components are a local memory (2MB), an MPPM interface, a common memory interface, and a hard disk. The control processor and PEs share a common memory (2MB) for storage of common tables and inter-processor communication purpose. As the PE is implemented by a general-purpose microprocessor, this machine bears some similarity to other multiprocessor database machines, for example, [DeWitt 86], [Kerr 82].

Each MPPM interface has a MPPM interface memory, physically implemented by dual-port RAMs. One block of the interface memory is used for PE-MPPM control information communication, and the rest of the blocks are for data buffers. The MPPM memory banks work synchronously, and once they begin to work, the one-page transfer to the buffer memory cannot be halted by a memory access conflict. This is the reason expensive dual-port RAMs are used in MPPM interface memory. The dual-port RAMs are mapped in the PE processor memory space and can be used just like normal memory area. The software is responsible for the management of the MPPM interface memory.

#### 4.3.2 MPPM

The MPPM is a memory system with multiple data read/write ports that can operate independently and simultaneously. The unit of access is a physical page (or a track in disk analolgy). The current implementation adopted a 512-byte track size. The most notable feature of MPPM is its conflict-free track access capability, that is, any port can access any track without conflict between any other ports. The cost for that is its latency time; a port cannot access a fraction of a track until the end of a complete track transfer.

To support the independent access to tracks from multiple ports, each port is provided with a channel-like intelligent controller (port controller) to interpret the track transfer request from the corresponding PE. The track transfer request is placed in the MPPM interface memory by a PE, in the form of a control block. The control block is called a *Page Transfer Control Block(PTCB)*. An MPPM port controller polls for a PTCB ready flag, which indicates that the corresponding PE has a transfer request and formed it in a PTCB. If the ready flag is set, the port controller reads in the current PTCB and sets up necessary hardware registers according to the contents of the PTCB for the port activation. The port controller is microprogrammed so that (1) it can read, interpret the PTCB and set up the port activation speedily to minimize the control overhead per port activation and (2) it can serve the complex requirements for a channel, for example, PTCB validity check and interrupt issuance at appropriate points. More important is its adaptability to the possibly alterable specification of a PTCB. A microprocessor could not fulfill the speed requirement and a hard-wired logic could not bear the complexity of its requirements.

Besides the port controller, MPPM consists of memory banks and a network to interchange the memory banks to ports. The requirements of this network are not as general as usual interconnection networks. It only has to work synchronously. If port i is, for example, connected to memory bank k at a time period, port i will be connected to memory bank k+1 mod n, where n is the number of ports, at the next time period. This cyclic network interchange is repeated n times; the total time is the track read-out time (track latency).

In the current implementation, as n is small (n is 8), the network is implemented by 3-state gates, essentially n pieces of n-to-1 selectors.

### 4.4 Control Software Overview

The control software consists of three major components. They are:

(1) control facilities

This is a collection of operating system like functions. For example, MPPM management, disk management, MPPM access method, inter-processor communication support and so on.

(2) operation modules

An operation module performs the assigned operation over one or several pages. The operations are, for example, selection and join. These modules use the above control facilities if necessary.

(3) compiler

The compiler compiles relational calculus based query into a sequence of internal commands. The internal commands are executed by the operation modules.

A query is received by the control processor and stored in the common memory. The control processor does not perform compilation nor execute control facilities. As is described in the research objective chapter, we do not centralize such control. (We notice that to call it the control processor, named for historical reasons, is thus misleading.) A free PE finds a new query in the common memory and compiles it into a sequence of internal commands in a transaction command queue. The queue is generated per transaction and placed in common memory. An idle processor polls for the transaction command queues. If there is a command in a transaction command queue, it executes the command. Commands are categorized into two types. One is the processor-specified type and the other is processor-non-specified. A processor-specified command must be executed by the specified processor. This type is necessary because the data is distributed across the disks in the PEs. If there is a need for a specific page fetch from disk, the command must go to the PE processor that has the page in its disk. A processor-non-specified type command can be executed by any processor. There is a parameter in processor-non-specified command that controls the maximum parallelism. If a processor-non-specified command is specified with n as the parameter value, the command cannot be executed by more than n processors. In other words, the command is shared by a maximum of n processors and executed in parallel. The typical case is when the command is a selection from multiple pages. Each of the free processors that can participate in the execution of the command looks at the page queue in the common memory and takes the page number specified at the top of the page queue. The processor then removes the taken page from the page queue. This page-taking is a critical section of control, so a lock using the common memory is necessary to ensure the atomicity. After taking out one page from the page queue, the processor reads in the page from MPPM and performs selection operation on the page. If the operation is complete and there still remains unprocessed pages, the processor continues the process until all the page numbers in the page queue is consumed.

## 5. Conclusion and Future Works

In this paper we described the knowledge base model and knowledge base architecture for the model first. We then described that the MP method, which controls the granurality of partial joins dynamically when the original join is executed by multiple processing elements. The MP method was simulated and shown to be

superior to the SP method as a parallel control strategy.

We then described several aspects of the experimental knowledge base machine, which is a prototype of the original architecture. An interface with the inference machine based on relational calculus was given. The hardware and software architecture of the experimental machine was described.

We are now implementing the control and knowledge base processing software on the experimental machine. First version of the integrated software will be completed next March. By examining the performance, usability and system behaviors using the integrated system, we will evaluate the following:

(1) effectiveness and limitation of the prototype architecture
(2) parallel control algorithms and implementations
(3) interface with the inference machine

The observations which have already be obtained, or will be obtained in the future are for the most part applicable not only to knowledge base machines but also to database machines.

[references]


[Ait-Kaci 85] Ait-Kaci, H., Nasr, R., "LOGIN: A Logic Programming Language with Built-In Inheritance", J. of Logic Programming, Vol. 3, 1986.

[Bancilhon 86] Bancilhon, F., Khoshafian, S, "A Calculus for Complex Objects", Proc. PODS, Cambridge, MA., 1986.

[Codd 70] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, 13, 6, 1970.

[Chikayama 84] Chikayama, T., "Unique Features of ESP", in Proc. Conference on Fifth Generation Computer Systems U84, Tokyo, 1984.

[DeWitt 86] DeWitt, D. J., et al. "GAMMA - A High Performance Dataflow Database Machine", Proc. 12th VLDB, Kyoto, 1986.

[Kerr 82] Kerr, D. S., et al. "The Implementation of a Multi-Backend Database System (MBDS): Part I - Software Engineering Strategies and Efforts Towards a Prototype MDBS", Technical Report, OSU-CISRC-TR-82-1, The Ohio State University, 1982.

[Morita 86] Morita, Y., et al. "Retrieval-by-Unification Operation on a Relational Knowledge Base", Proc. 12th VLDB, Kyoto, 1986.

[Morita 87] Morita, Y., et al., "Performance Evaluation of a Unification Engine for a Knowledge Base Machine", ICOT Technical Report TR-240, 1987.

[Murakami 83] Murakami, K., et al. "A Relational Data Base Machine: First Step to Knowledge Base Machine", Proc. 10th International Symposium on Computer Architecture, Stockholm, 1983.

[Sakai 84] Sakai, H., et al., "Design and Implementation of the Relational Database Engine", Proc. Conference on Fifth Generation Computer Systems U84, Tokyo, 1984.

[Sakai 87] Sakai, H., et al., "A Simulation Study of a Knowledge Base Machine Architecture", Proc. 5th International Workshop on Database Machines, Karuizawa, 1987.

[Shibayama 84] Shibayama, S., et al. "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor", New Generation Computing, Vol. 2, 1984.

[Taki 84] Taki, K., et al. "Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)", Proc. Conference on Fifth Generation Computer Systems U84, Tokyo, 1984.

[Tanaka 84] Tanaka, Y., "Multiport Page-Memory Architecture and a Multiport Disk-Cache System", New Generation Computing, Vol. 2, 1984.

[Tsur 86] Tsur, S., Zaniolo, C., "LDL: A

Logic-Based Data-Language", Proc. 12th VLDB, Kyoto, 1986.

[YokotaH 86] Yokota, H., Itoh, H., "A Model and Architecture for a Relational Knowledge Base", Proc. 13th International Symposium on Computer Architecture, Tokyo, 1986.

[YokotaK 87] Yokota, K., "Deductine Approach for Nested Relations.", ICOT Technical Report , to appear.