ICOT Technical Report: TR-384

TR-384

The Design and Implementation of a Knowledge Base Machine Mu-X

by
H. Sakai, S. Shibayama, A. Nakase(Toshiba)
H. Monoi, Y. Morita and H. Itoh

May, 1988

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

THE DESIGN AND IMPLEMENTATION OF A KNOWLEDGE BASE MACHINE Mu-X

Hiroshi Sakai, Shigeki Shibayama, Akihiko Nakase
Toshiba R & D Center
Hidetoshi Monoi, Yukihiro Morita, Hidenori Itoh
ICOT Research Center

ABSTRACT

Within Japan's Fifth Generation Computer Systems Project, a knowledge base machine named Mu-X is being developed. Although the term "knowledge base" seems to have various meanings, Mu-X is designed as an advanced database machine with the features typical to the fifth generation computer systems.

This paper shows the design considerations of the system architecture, especially of the management software on the parallel architecture. Mu-X is designed as a software-oriented multiprocessor system with hybrid shared memory systems. The management software aims at (1) effective use of the hardware architecture, (2) minimizing software overhead, and (3) parallel processing for high throughput. These design principles are reflected to the actual implementation and a good system performance is expected in a multitransaction environment.

1 INTRODUCTION

We have conducted research into knowledge base mechanisms within Japan's Fifth Generation Computer Systems Project. In an intermediate project lasting four years, a knowledge base machine named Mu-X is being developed. Although the term "knowledge base" seems to have various meanings, Mu-X is designed as an advanced database machine with the features typical to the fifth generation computer systems.

We developed the relational database machine, Delta, as the first step of the research [Kakuta 85], [Sakai 86]. Its experience led us to the following conclusion: while

a shared database is useful for knowledge information processing, the relational model of the first normal form is not sufficient. The initial idea of adopting the relational model was that one can manage a number of facts with the same name and arity as a relation. However, it is difficult to handle complex objects such as list and term in Prolog programs efficiently in this model.

Therefore, the basic data type in Mu-X is the term type which includes the integer and string type. An extended relational model which is called the term relational model is adopted so as to hold a term as an attribute value [Morita 86]. Fig. 1 shows an example which contains variables in term structure. In an operation over relations, unification between terms is allowed as a condition like comparison between constants. The data dictionary describing the knowledge about relations is also such a relation, where the knowledge about one relation is stored as a tuple and the knowledge about its attributes forms a term structure and is stored in an attribute. The logical interface to a host machine PSI is also represented in term structure. Therefore, a command may be stored in an attribute and an attribute value may be interpreted as a command, which is an example of extending database with procedures [Stonebraker 87].

This paper focuses on the features of Mu-X as a multi-processor database machine, especially on its management software reflecting the parallel architecture.

2 Design Considerations

2.1 Design Goals

Mu-X is assumed to be used in a multi-transaction environment where high throughput is as important as quick response. The queries within a transaction are classified into two types; some of them require small resources like indexed database access with high selectivity and the others require many more resources like a join operation.

In order to preserve the atomicity of the transactions, the three basic transaction operations of start, commit, and abort may be used as well as locking protocols. Locking on a small part of a relation is necessary as that on an entire relation.

A session, an interval from the logon to the logoff of a host computer is assumed to be a sequence of a number of transactions. In this paper, the term session is often used in place of the term transaction. In knowledge information processing, a retrieval result may be used as the source of another operation. Such a result should be treated as a temporary relation, the scope of which is limited within the session.

As for the internal schema of a relation, the limitations should be loose. The tuple size should not be restricted by the physical page size. The number of attributes of a relation should not restricted to a small number. Each attribute should contain variable length data for space efficiency. The internal representation of a term should follow that of the inference machines in order to improve the efficiency and capability of the system.

2.2 Basic Design of Mu-X

Mu-X aimed at a versatile database system flavored with features of the fifth generation computer systems, and therefore the quantity of software was estimated to become considerable. So the multiprocessor approach with hybrid shared memory systems has been chosen.

Mu-X is designed to take advantage of the techniques which have been developed in both the conventional software oriented database systems and the multiprocessor database machines. Clustering on a primary key attribute and indexing on secondary keys greatly improve the system performance in most cases. Declustering (horizontal partitioning) of a relation and parallel processing algorithms give the system performance proportional to the number of processors as long as the number of processors is small.

Dedicated hardware like the Relational Database Engine incorporated in Delta [Sakai 84] was not adopted by the two reasons. The first reason is that there are so many miscellaneous operations that it seems better to achieve higher cost/effectiveness by making all the varied operations faster by the conventional hardware techniques i.e. adopting the memory cache and/or incrementing the number of processors. The second reason is that such an engine designed for the processing of large volume of data is of little use in the typical transaction operation where only a small portion of a relation is

processed.

2.3 Hardware Architecture

Mu-X is a tightly coupled multiprocessor system with a front-end processor connected to the local area network called PSI-net and eight processing elements (PEs) as illustrated in Fig. 2. A multiprocessor system with a conventional shared memory is not appropriate for parallel database processing, since heavy data transfer will take place at the shared memory, thus degrades performance. In order to resolve the problem, a conflict-free shared memory system is incorporated in the hardware design. Mu-X has two kinds of shared memories; one is a 2 Mbyte conventional memory with a shared bus, and the other is a multiport page-memory (MPPM) which was proposed by Tanaka [Tanaka 84]. The MPPM consists of a fixed number of memory banks (eight in our current implementation), the controllers (one for each PE), and a switching network which periodically changes the one-to-one connections between the PEs and the memory banks. The switching network prevents data contention and assures constant data transfer rate for each port, though the data access is limited to multiples of a certain fixed size (512 bytes in our current implementation).

Each PE has a MC68020 MPU, a 2 Mbyte local memory, a 47 Mbyte disk device, and a 12 Kbyte dual-port memory. In our current implementation, only the data transfer between the MPPM and the dual-port memory is supported. When the PE needs to access the MPPM, it has to set up the parameter block within the dual-port memory so as to make the controller of the MPPM transfer the data between them. It takes about one millisecond for a PE to complete transferring 2 Kbytes of data including the processing time of the software. In order to transfer data from the MPPM to the local memory or the shared memory, the PE has to transfer the data into the dual-port memory as stated above first. And then the MPU of the PE has to transfer the data into the local memory or the shared memory word by word. It takes about seven milliseconds in this case.

The overall architecture of Mu-X is not suitable for highly parallel needs because the shared memory is used. However by properly use of the MPPM and the local memory, it

is expected to be useful so far as the number of PEs is less than about fifty. The MPPM with tens of ports could be developed unless the pin count bottle-neck becomes serious. One could equip even more PEs than the number of ports of the MPPM by connecting several PEs to each port like a block multiplexing channel.

2.4 Design Principles of the Software

The management software running on the above hardware architecture is designed based on the following principles.

(1) Effective Use of the Hardware Architecture

This principle is reflected especially in the functional distribution among the processors and the use of the hybrid shared memory systems, the MPPM and the shared memory. As for the functional distribution, it is expected that the performance of the system can be improved by simply adding PEs and increasing the number of the ports of the MPPM. Therefore the design aims to minimize the work of the front-end processor and equalize the work of each PE in terms of quality as described in 3. As for the use of the storage, the details are described in 4.

(2) Minimizing Software Overhead

This principle is reflected in the control mechanisms for parallel processing of database operations and the software configuration of the PE. As for the control mechanisms, the internal commands designed for the purpose is proved effective as described in 5. As for the software configuration, it is designed as a single task program which takes two roles alternately as described in 3.2.

(3) Parallel Processing for High Throughput

Parallel processing algorithms on multiprocessor database machine have been extensively studied and implemented in order to achieve quick response [Boral 82], [Hanson 87], [Nakamura 87], [Wilkinson 87], [Bitton 83], [Kitsuregawa 84], [Bitton 84], [Shapiro 86]. Though Mu-X adopts some of these algorithms tailored to the hardware architecture, it still aims to achieve high throughput in the multitransaction

environment. This principle is reflected in the various portions of the software and often merged with other principles.

3 Basic Design of the Software

3.1 Distributing Functions among Processors

Bus-structured multiprocessor database systems in general have a control computer and a number of data processors [Su 88]. The control computer manages transaction execution and the global data dictionary, translates queries from a host computer to low-level internal commands, dispatches them to the data processors for execution, and collects retrieval results from the data processors. The data processors handle I/O between the private main memories and the secondary storage devices and perform the specified database operations.

The configuration, in such multi-transaction environment described in 2.1, seems to have the disadvantage that the ratio between the processing power of the control processor and that of the data processors is not changeable dynamically. That is, since queries from host computers require similar processing power of the control computer and various processing power of the data processors, the control computer becomes the bottle-neck for the system throughput in the case where most queries require small processing power of the data processors. Although systems with several control computers like DBC/1012 resolve the bottle-neck, they are not able to change the ratio of the processing power automatically according to the tendency of the queries.

Mu-X resolves the problem by assigning each PE both the role of the control computer and that of the data processor.

As for the first role, each PE is in charge of at most one session. When a host computer requests to create a session, the front-end processor seeks a free PE to be responsible for it. The request is permitted by the front-end processor if such a PE is found, otherwise it is refused. Therefore the number of the concurrent sessions is restricted as many as the number of the PEs. This limitation comes from the design principles of the software and not from the hardware architecture. During the session, the

front-end processor delivers the queries from the host computer to the related PE and sends the responses made by the PE to the host computer. When Mu-X is operational, the front-end processor treats the PSI-net protocols only so that it might not become the bottle-neck.

As for the second role, each PE works just the same way as the data processors described above.

3.2 Configuration of the Software of the PE

Since each PE, a single processor, has to play the above two roles concurrently, the execution should be interleaved with little overhead. The software of the PEs includes the management module and the processing module, corresponding to the two roles respectively. This software interleaves the two modules in a simple way. Let us examine the execution process of the management module in order to find a proper way. Fig. 3 illustrates the general flow of the execution process of the management module.

At the state S1, the module is waiting for a query arrival. On the arrival, its state becomes S2.

A query, as shown in Fig. 4(a), is represented in term structure. In this example, the result of a selection operation and the result of a join operation over three relations are put together into a temporary relation. Here, a variable which appears in several places relates the values stored in each place. Since the term structure is not convenient for the management module to handle, the module transforms it to a query tree at S2. Fig, 4(c) shows the tree corresponding to the above query.

At S3, the module accesses the data dictionaries to get the information about the relations specified in the query tree and to lock the entire relations if necessary.

At S4, the module selects a part of the query tree to be processed according to the information acquired at S3 and generates and dispatches the corresponding internal commands. Fig. 4(d) shows the essence of the internal command. As for the join operation, besides the multiprocessor nested loop algorithm, Mu-X adopts the hash based algorithm which was proposed by Kitsuregawa [Kitsuregawa 84] and improved by

Shapiro [Shapiro 87]. In this case, the join operation is processed by three internal commands; the first two commands request the dynamic clustering on the specified attributes over each relation and the last command requests hash based join operation on each pair of the corresponding clusters.

At S5, the module is waiting for the response to the internal commands.

At S6, the module reduces the query tree according to the response. Fig. 4(e) shows the modified query tree. If the modified query tree is still to be processed, its state becomes S4. Otherwise it becomes S7.

At S7, the module generates the response to the host computer and dispatches it to the front-end processor. Then its state becomes S1 in order to wait for the arrival of another query.

As described above, the management module generates and dispatches the internal commands step by step as it reduces the query tree. The process seems to have the advantage that a command which is appropriate to the situation can be generated easily in comparison with the case the query is translated into the internal commands at once.

Following the consideration on the management module, the PE software was designed as a single task program. That is, the control module calls the processing module at S1, S3 (only if the lock operation is suspended), and S5. Then the processing module seeks the internal commands which the PE can process and handles I/O operations to process the individual data according to the command. When there is no such internal commands, the processing module returns and the management module continue the execution.

In most systems, these modules are realized as two individual programs on a real-time multitasking operating system. However, the Mu-X approach has the following advantages. The first advantage is to minimize the task switching overhead. This is important especially in a multitransaction environment. The second advantage is that the dual-port memory, the I/O buffer to the MPPM, can be fully shared by these modules since the execution is separated. The third advantage is that the requested code area becomes smaller by sharing common procedures.

4 Considerations on the Storage for Management of Information

4.1 Considerations on the Storage

In order to achieve a good performance on the Mu-X architecture, there are two major points: avoidance of the access contention to the shared memory and full use of the MPPM.

In Mu-X, a session is related to a PE as described in 3.1. It has an effect to avoid the access contention to the shared memory since the knowledge local within a session can be stored in the local memory of the corresponding PE. Otherwise, it would be stored in the shared memory.

As for the MPPM, [Tanaka 84] proposed that a MPPM can be used as a shared disk cache within a massive paraliel database machine architecture. The management software of Mu-X, however, treats the MPPM as a fast common storage independent from the disk devices. This is because in such a software oriented system like Mu-X, the management software knows the characteristics of each data better than any page replacement algorithms for disk cache systems. In the case where an overflow takes place, the overflow portion is stored into a disk device. The MPPM is used to keep temporary relations, the copy of the global data dictionary, and the new version of a page data which is modified and not committed yet. The details are described below and summarized in Fig. 5.

4.2 Management of Relations

(1) Temporary Relations

A temporary relation is created typically by a retrieval operation and is accessible only within the session in which it is created. It is stored in what is called an MPPM file so that it may be processed by any PEs.

An MPPM files is a sequence of fixed size pages (2 Kbytes in our current implementation) allocated within the MPPM. Control tables are used to get the required page location within the MPPM from the logical page number of the MPPM file. They are stored in the shared memory so that an MPPM file can be accessed by any PE. An MPPM file could be used to store a permanent relation. Thus, the locking protocols and the version management on each page of the MPPM file are also realized in the control

tables. There are 256 MPPM files available in our current implementation.

(2) Permanent Relations

A permanent relation is accessed many times and is sharable among the sessions as long as the combination of the locking protocols is permissible. It is usually horizontally divided and stored into the disk devices. As for the declustering, it is assumed that most relations are stored according to the hash value of an attribute of each tuple, so that a typical transaction concerning one relation may be processed by one PE. In addition to the hash based scheme, the round robin scheme is also adopted.

Within a PE, the individual parts of a relation is stored in what is called a disk file. A disk file is basically a sequence of fixed size pages within a disk device and can be shared among the sessions. Control tables are used to realize the locking protocols and the version management on each page like the MPPM file. However, most of them are stored in the local memory.

In an update operation, the new version of a page data is stored into a newly allocated page of the MPPM and the control tables are modified so that the transaction ID, the page number of the disk file, and the allocated page address of the MPPM are memorized. Such use of the MPPM seems to be useful, since the stored data is often accessed more than once. When the transaction is committed, the new version of the page data is actually stored in the disk device, where the file management facilities of a real-time operating system is used.

A disk file does not support any clustering or indexing scheme in itself. This is because such work, like a selection operation, should be done by the processing module so that the software overhead may be totally minimized.

(3) Temporary Clusters

A set of temporary clusters are generated in a dynamic clustering operation. Each cluster is stored in what is called a bucket file.

A bucket file is stored into the MPPM like the MPPM file. The control tables, however, are simpler than that of the MPPM file, because its use is limited. Since a number of bucket files should be allocated and released at once, the Buddy algorithm is

used to realize these operations.

4.3 Data Dictionaries

Most database systems have both a data dictionary and directory in order to keep the knowledge about the system. The former is for a friendly user interface and the latter for the system efficiency. The data directory, however, is not necessary in Mu-X because the information can be extracted efficiently from the data dictionary. A data dictionary is classified into two types, the local data dictionary and the global data dictionary.

(1) Local Data Dictionary

A local data dictionary belongs to a session and mainly keeps the knowledge about temporary relations such as the data type, the name of each attribute, and the number of tuples. It is stored in the local memory of the PE responsible for the session. In order to process a query which requires access to the local data dictionary, the management module makes its copy as a temporary relation.

(2) Global Data Dictionary

The global data dictionary keeps the knowledge to be shared among the sessions such as knowledge about the permanent relations. It is realized as a relation horizontally partitioned across the disk devices before the startup of Mu-X. The round robin scheme is used so as to equalize the volume among the disk devices. During the initialization phase of Mu-X, a copy of the global data dictionary is made into the MPPM by the PEs. On the other hand, during the shutdown phase of Mu-X, the copy within the MPPM is divided horizontally and stored into the disk devices. In order to process a query which requires access to the global data dictionary, the copy within the MPPM is used.

To share the global data dictionary relation among the sessions efficiently, a hash table and a pool of page buffers with two related tables are provided within the shared memory. The hash table allows fast access to the address of the tuple specified by the given relation name (i.e. the page number and the offset within the page). A page buffer of the pool holds the copy of the currently concerned page (128 page buffers in our current implementation). The pool has the following two tables. One has as many entries as the

number of the pages of the global data dictionary. Each entry holds the page buffer address if the page is staged in the page buffer. The other table has as many entries as the number of the page buffers. Each entry holds the reference count over all the transactions and the tuples within the page. For example, assume that a page buffer holds the page data which has the knowledge about the relations R, Q, and the relation R is referred by three sessions and the relation Q is referred by one, then the reference count is set to four.

When a PE wants to know about a relation, it gets the page number of the copy and the offset within the page through the hash table. Then it checks whether the page is currently staged in a page buffer through the first table. If it is not staged, the PE must find a page buffer the reference count of which is equal to zero and transfer the page data into the page buffer. Certainly such a page exists so long as the number of the page buffers is larger than the summation of the reference counts. At last the PE has to increment the reference count of the page buffer.

It takes less than 1 millisecond for a PE to get the information of a relation when the related page is already staged, and about 8 milliseconds otherwise.

When a relation is modified in a transaction and it is committed, the information about the relation should be also updated. The PE responsible for the session modifies the information within the page buffer and puts the page data back to the copy of the global data dictionary relation.

Intent lock protocols [Gray 76] on an entire relation are realized within the page buffers. In our current implementation, each tuple of the global data dictionary has a 6 byte attribute for the purpose which allows at most fifteen concurrent transactions.

In summary, the global data dictionary is stored within the MPPM when Mu-X is operational so that it may be accessible by any PEs. The pages having the knowledge about the active relations are staged in the shared memory and shared among the sessions.

5 Internal Commands for Parallel Processing of the Database Operations

An internal command requires the PEs to process a relational algebra level operation

in parallel. In order to minimize the software overhead, only one command is generated and shared among the related PEs.

(1) PE Specific and Nonspecific Commands

Internal commands are classified into two types, PE specific and PE nonspecific ones.

A PE specific command concerns a relation stored in the disk devices and must be processed by the PEs which own the devices. Therefore a PE specific command has an affinity mask, a bit array which indicates whether each PE has to process it or not. A PE finishes the processing of the command when all the objects within its disk device are processed.

A PE nonspecific command concerns only those relations stored in the MPPM and can be processed by any PE. In Mu-X, the idle PEs process such a command, which helps to balance the processing load among the PEs. A PE nonspecific command has a counter indicating how many additional PEs can join the processing. A PE finishes the processing of the command when there is no more objects to be processed. Note that it is possible that some objects are still being processed by other PEs.

(2) Starting and Finishing Masks

It is important to decide whether an internal command is completed by all the related PEs. It is not obvious because several internal commands belonging to the different transactions are to be processed concurrently and therefore an internal command might not be processed synchronously by the related PEs. For example, in case of a PE specific command, it may happen that some of the PEs have already finished the processing while the others have not even started it yet.

A starting and finishing masks within an internal command are used for the purpose. The starting mask is a bit array indicating whether each PE has started the processing and the finishing mask is a similar array indicating whether each PE has finished it. The software finds a PE specific command is completed when the finishing mask becomes equal to the affinity mask, while it finds a PE nonspecific one is completed when the finishing mask becomes equal to the starting mask.

These masks are useful to improve the average response time of the system.

Suppose a command requiring much processing time is being processed by a PE and there exists another PE specific command related to the same PE requiring little processing time. If the PE could not suspend the processing of the current command, the average response time would become worse. In Mu-X, this is avoided by interleaving the processing of the commands within a PE like conventional multi-tasking systems. A PE checks whether the processing time exceeds a certain limit (1 second in our current implementation) on finishing the processing of each object. If the time exceeds the limit, the PE looks for other PE specific command related to itself. In case that such a command is found, the PE can safely begin the processing of the new command by leaving the finishing mask of the current command unchanged.

(3) An Object Counter

An object counter within the internal command is used to decide which PE should process each object. For example, in a selection operation which requires the scan of a relation within the MPPM, each page of the relation must be processed only once by an arbitrary PE. In this case, the object counter is set to zero (the first page number) when created. Each PE tries to get the content of it (i.e. the page number to be processed next) and to increment it by one in a critical section. The CAS (compare and swap) instruction of the MC68020 MPU is used efficiently in such a operation.

In a join operation on each pair of the corresponding clusters, the object counter counts up the pair number.

In summary, an internal command is used to process a relational algebra level operation in parallel. By taking advantage of the shared memory, the synchronization among the PEs is realized with little software overhead.

6 Conclusion

The design considerations on the management software running on the parallel architecture of Mu-X are discussed. Three design principles of the software, (1) effective use of the hardware architecture, (2) minimizing software overhead, and (3) parallel processing for high throughput are reflected to the actual implementation.

Although Mu-X is still under construction, it has just begun to process queries from a host computer PSI. It is found Mu-X processes queries as fast as estimated under single transaction environment. Since the performance evaluation is considered important in such a parallel architecture, we plan to make a thorough evaluation using hardware monitors as well as software methods.

ACKNOWLEDGEMENT

We would to express our special thanks to Prof. Y. Tanaka for permitting the use of his idea concerning the MPPM.

REFERENCES

[Bitton 83] Bitton, D., et al.: "Parallel Algorithms for the Execution of Relational Database Operations", ACM Transactions on Database Systems, Vol. 8, No. 3, pp. 324-454.

[Date 82] Date, C. J.: "An Introduction to Database Systems", The Systems Programming Series, Addison Wesley, 1982.

[Gray 86] Gray, J. N., et al.: "Granularity of Locks and Degrees of Consistency in a Shared Data Bases", 1976.

[Hanson 87] Hanson, J. G., Orooji, A.: "Experiments with Data Access and Data Placement Strategies for Multi-computer Database Systems", Proceedings of the Fifth International Workshop on Database Machines, pp.597-610, 1987.

[Kakuta 85] Kakuta, T., et al.: "The Design and Implementation of Relational Database Machine Delta", Proceedings of the Fourth International Conference of Database Machines, 1985.

[Khoshafian 87] Khoshafian, S., Valduriez, P.: "Parallel Execution Strategies for Declustered Databases", Proceedings of the Fifth International Workshop on Database Machines, pp.626-639, 1987.

[Kitsuregawa 84] Kitsuregawa, M., et al.: "Architecture and Performance of the Relational Algebra Machine GRACE", Proceedings of the International Conference on

Parallel Processing, 1984.

[Morita 86] Morita, H., et al.: "Retrieval by Unification Operation on a Relational Knowledge Base, Proceedings of the 12th International Conference on Very Large Databases, 1986.

[Nakamura 87] Nakamura, H., et al: "A High Speed Database Machine - HDM", Proceedings of the Fifth International Workshop on Database Machines, pp.340-353, 1987.

[Papachristidis 87] Papachristidis, A. C.: "Dynamically Partitionable Parallel Processors: The Key for Cost-Efficient High Transaction Throughput", Proceedings of the Fifth International Workshop on Database Machines, pp.328-339, 1987.

[Sakai 84] Sakai, H., et al.: "The Design and Implementation of the Relational Database Engine", Proceedings of the Fifth Generation Computer Systems '84, 1984.

[Sakai 86] Sakai, H., et al.: "Development of Delta as a First Step to a Knowledge Base Machine", Database Machines Modern Trends and Applications, NATO ASI Series F, Vol 24, 1986.

[Shapiro 86] Shapiro, D. L.: "Join Processing in Database Systems with Large Main Memories", ACM Transactions on Database Systems, Vol. 11, No. 3, pp. 239-264.

[Shibayama 85] Shibayama, S., et al.: "A Knowledge Base Architecture and its Experimental Hardware", Proc. IFIP TC-10 Working Conference on Fifth Generation Computer Architectures 1985.

[Su 88] Su, S. Y. W.: "Database Computers Principles, Architectures & Techniques", McGraw-Hill, 1988.

[Tanaka 84] Tanaka, Y.: "A multiport Page-Memory Architecture and a Multiport Disk-Cache System", New Generation Computing, Vol.2, No.3, pp.241-260, 1984.

[Wilkinson 87] Wilkinson, W. K., Boral, H.: "KEV - A Kernel for Bubba", Proceedings of the Fifth International Workshop on Database Machines, pp.29-42, 1987.

Rule Relation	Head	Body
	Ancestor(X. Y)	father(X. Y)
	Ancestor(X. Y)	mother(X. Y)
	Ancestor(X. Y)	(father(X. Z). ancestor(Z. Y))
	Ancestor(X. Y)	(father(X. Z). ancestor(Z. Y))

Fig.1 Example of a Term Relation

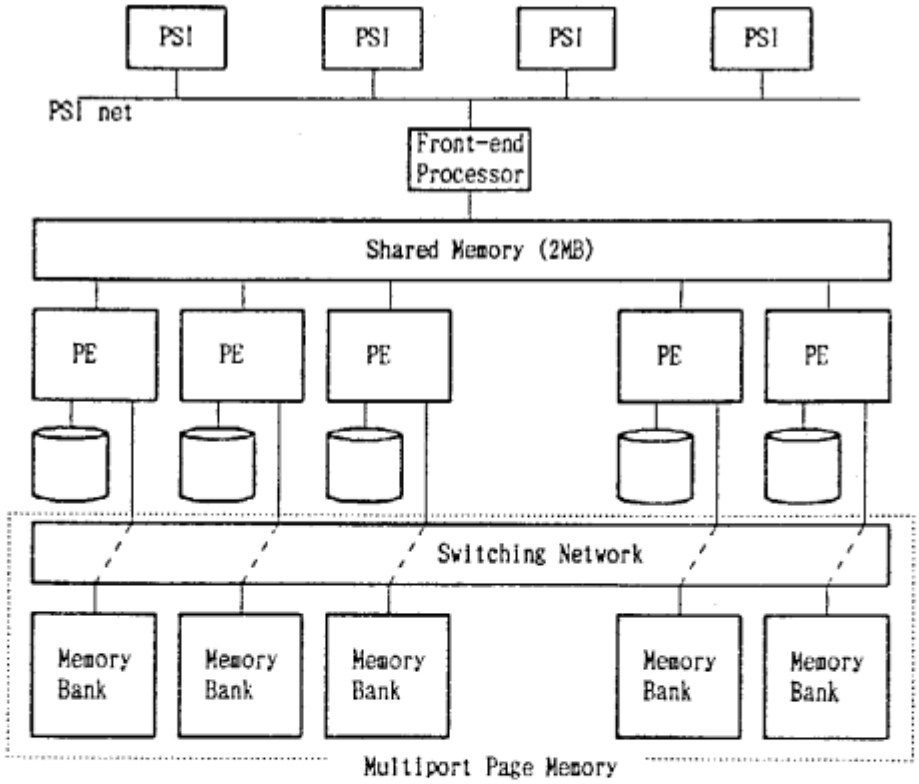


Fig.2 Hardware Architecture of Mu-X

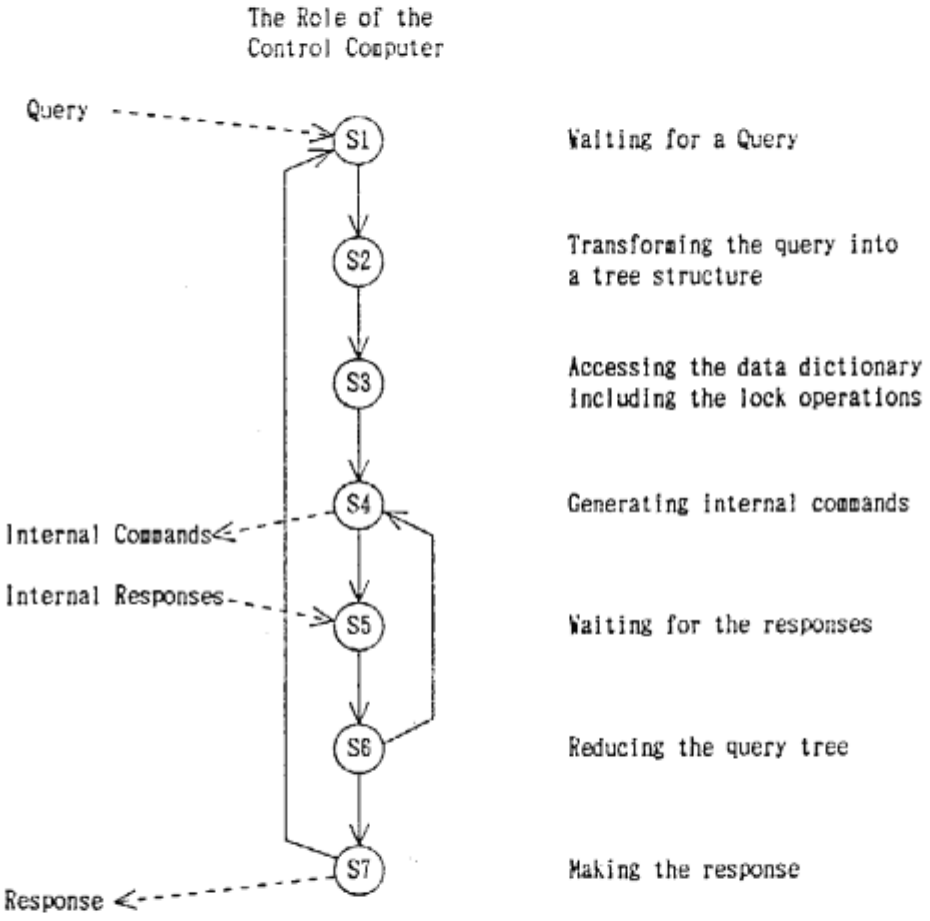
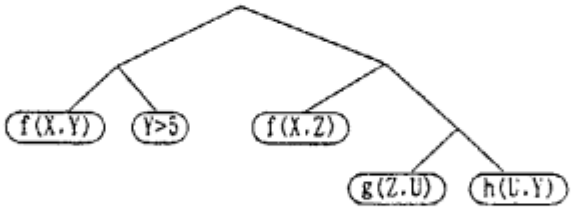


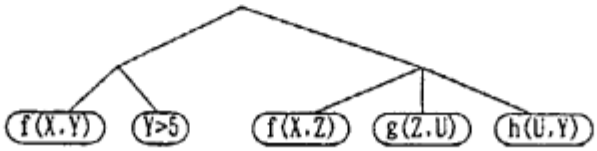
Fig. 3 The General Flow of the Execution Process of the Management Module

retrieve(r(X.Y), (f(X.Y).Y>5);(f(X.Z).g(Z.U).h(U.Y)))

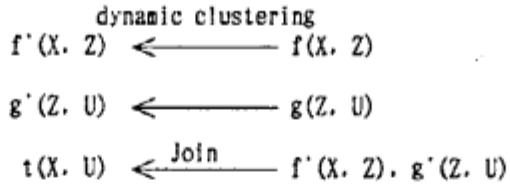
(a) example of the query reprsented in term structure



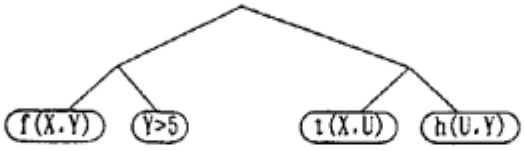
(b) The binary structure corresponding to the query(a)



(c) The query tree transformed from the query(a)



(d) essence of the internal commands



(e) the reduced query tree

Fig. 4 Example of the reduction process of a query tree

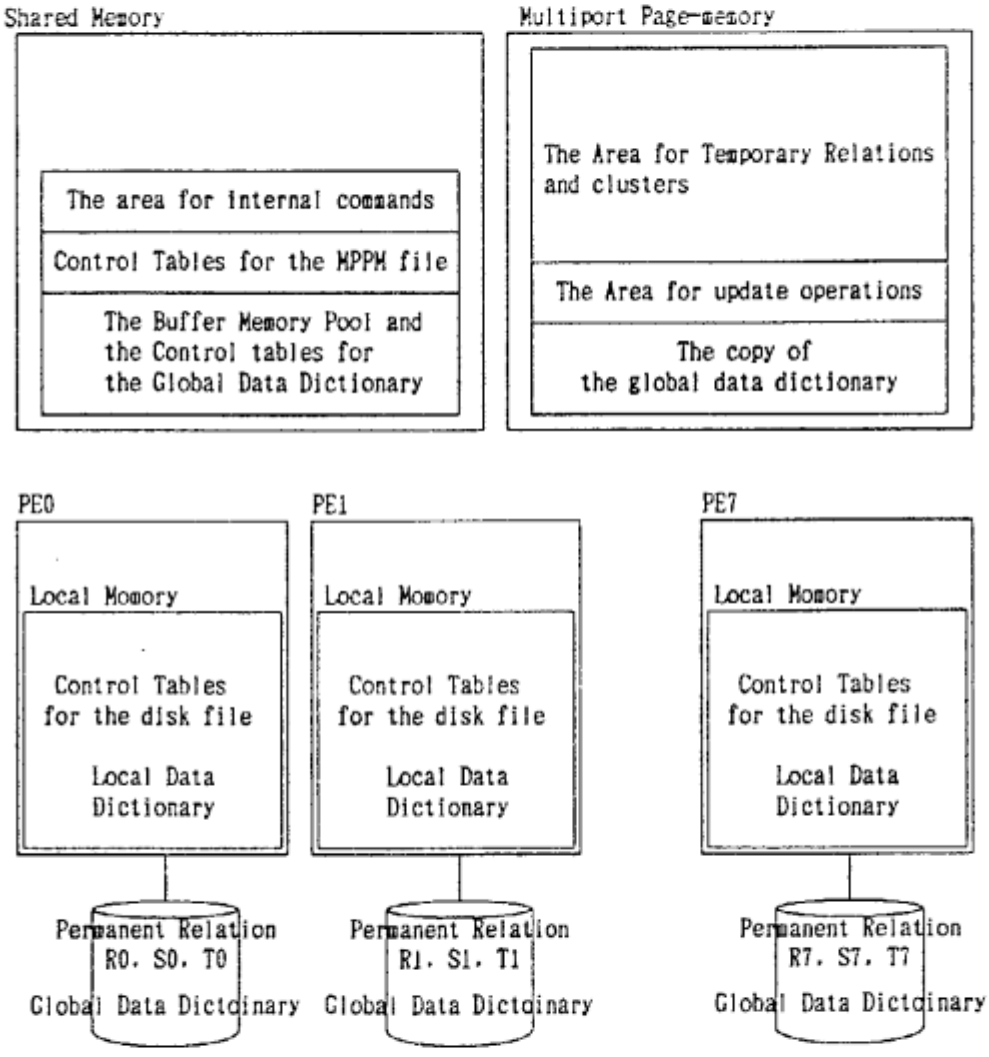


Fig. 5 Use of the storage in Mu-X