PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

Overview of Knowledge Base Mechanism

Shigeki Shibayama, Hiroshi Sakai, Toshiaki Takewaki
Toshiba Research and Development Center
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, 210, Japan
Hidetoshi Monoi, Yukihiro Morita, Hidenori Itoh
Institute for New Generation Computer Technology
4-28, Mita 1-Chome, Minato-ku, Tokyo, 108, Japan

ABSTRACT

This paper describes an experimental knowledge base system, which is one of the knowledge base mechanism research efforts carried out in the intermediate stage of the Japan's Fifth Generation Computer Project. The system employs the relational knowledge model, an extension of the relational data model, which allows a "term" data type.

The hardware adopted a hybrid shared memory multiprocessor architecture. The processing elements shares a conventional shared memory and a multiport page-memory, a page-based conflict-free memory. Dedicated knowledge base management software was built on the hardware. The efficiency of the software and the effectiveness of the architecture are shown in the preliminary evaluation.

1 INTRODUCTION

In this paper, we will describe a parallel knowledge base machine research effort carried out in the intermediate stage of the Japan's Fifth Generation Computer Project. In another article [Itoh 88] interrelation among the four research efforts, including this one, within the knowledge base mechanism research is given.

In the initial stage of the FGCS project, a relational database machine Delta was built as a first step toward a knowledge base machine [Kakuta 85], [Shibayama 84]. Though Delta was operational, it was recognized that the strict relational model adopted in Delta was not sufficient for the basis of the further knowledge base machine research [Shibayama 85]. The modification of Delta's software was considered to be difficult because of the large amount of software. The quantity came from the fact that there was software for processors which were specialized to different purposes. Also, the amount of parallelism in Delta was limited (four engines) for carrying out research into parallel knowledge base processing.

With the introduction of the relational knowledge model [Yokota 86], we thought that this model was appropriate for a basis of the interface between logic programming languages and knowledge base systems. This is because the model enhanced the relational data model to allow terms as a data primitive and enabled a kind of deduction without the aid of inference machines. The parallel processing method on special hardware was investigated [Morita 87].

However, a simulation study [Sakai 87] disclosed that the deduction was not so attractive in terms of processing speed. This is mainly because there are many repeated relation transfers in the deduction process. Even dedicated hardware (unification engines and a multiport page-memory [Tanaka 84]) did not remedy the situation.

With these intermediate results, we shifted the research direction to implement a backend knowledge base machine. In the course of the simulation study, an experimental parallel machine was being built. Originally it was aimed to be used for the hardware version of the simulation. The architecture proposed in [Yokota 86] incorporated hardware unification engines as the processing element core. The experimental hardware, however, did not incorporate unification engines. It was because at the outset of its development the unification engine design was considered to be premature and the hardware amount was predicted to be too much. This experimental machine was not used for the further simulation but was used for the implementation of the knowledge base management software.

The knowledge model we adopted was still the relational knowledge model. We were more concerned with the incorporation of the relational-algebra-like primitive operations and unification-based query language as an interface to host PSI machines than the deduction capability with the model though it is possible to do that if we do not care much for the processing time.

The hardware was completed by the end of 1986 fiscal year. The knowledge base management software's work has been carried out since 1987.

In the context described above, we have been engaged in the design and implementation of an experimental knowledge base system, named Mu-X. The knowledge base is defined as a collection of term relations, that is,

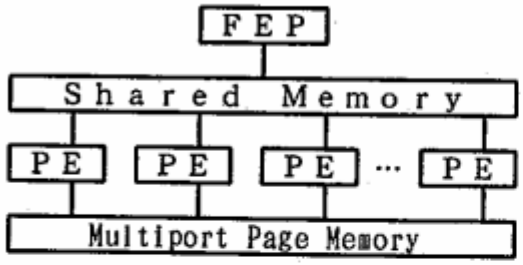

Figure 1. Hardware Configuration

relations extended with attributes that have structured items and variables. Conventional attributes such as integer and character data types are also supported.

In section 2, the hardware of the system is briefly described. In section 3, the software of the system is described somewhat in detail. In section 4, preliminary performance evaluation results are given and discussed. In section 5, discussions are presented on the system architecture and possible improvements. Section 6 introduces a sample information retrieval system developed using the interface set up for the PSI connection. Section 7 is the conclusion.

2 HARDWARE

Mu-X adopted a shared memory multiprocessor architecture. Mu-X mainly consists of eight processing elements (PEs), a conventional shared memory and a multiport page-memory (Figure 1). Each PE consists of a general-purpose microprocessor, a moving-head disk, a local memory and a multiport page-memory interface. There is no special-purpose hardware for functional distribution of database tasks [Shibayama 87].

The multiport page-memory [Tanaka 84] is a conflict-free memory system shared through the ports it provides. The multiport page-memory consists of a set of 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. This is illustrated in Figure 2. In our implementation, the port count is equal to the PE count, i.e. eight. Each PE is connected to a port of the multiport page-memory. For the switching network, we used multiplexor between the memory banks and the ports to make the hardware simple. More highly parallel implementation would require, for example, a multistage network.

Hence Mu-X has two types of memory systems. The multiport page-memory and the shared memory both

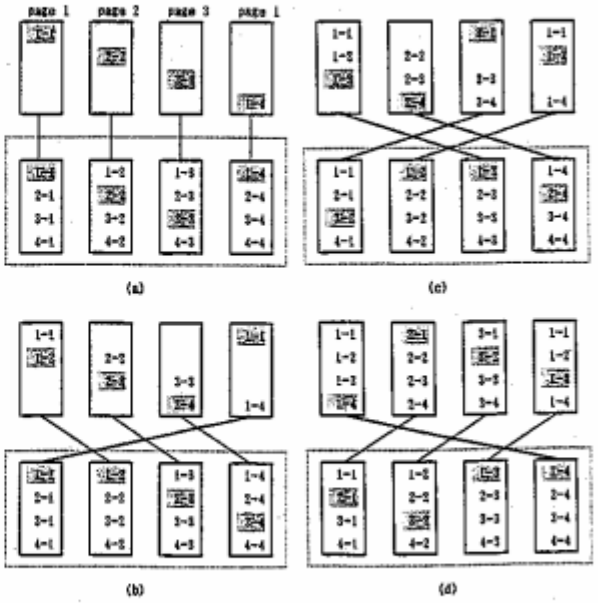

Figure 2. Multiport Page-Memory Principle

provide capability for exchange of information among the PEs. The conventional shared memory and the multiport page-memory have the following characteristics.

(1) For conventional shared memory, the unit of access is typically a word, while for multiport page-memory data is accessed on page basis.

(2) Conventional shared memory has potential access conflict among multiple PEs, while for multiport page-memory no access conflict occurs.

(3) For shared memory, access (when there is no memory access conflict) is quick, typically one or a few microsecond, while there is at least a page transfer time overhead for the multiport page-memory page access.

These characteristics are taken into account in the software implementation.

To complete the machine quickly, we used off-the-shelf components for the processor, local memory, shared memory, disk controller and disk unit. The multiport page-memory, its interface within each PE, and the shared memory arbiter are newly designed and fabricated. The hardware consists of two cabinets. In one cabinet eight processing elements and the conventional shared memory are installed and in another cabinet the multiport page-memory is installed. The front end processor is an off-the-shelf personal computer (VME-10),

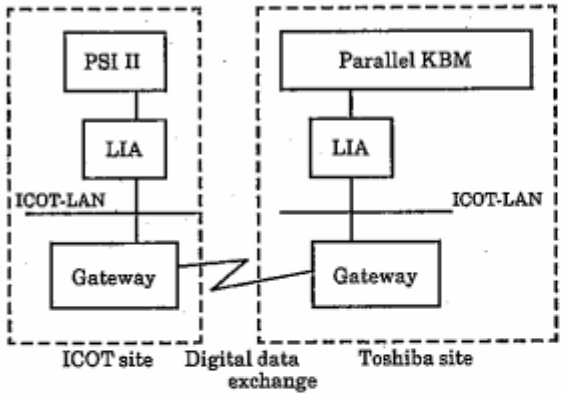

Figure 3. Network Connection

Table 1. Hardware Specifications

PE count	8
PE processor	MC68020 at 12.5MHz
Local memory	2MB/PE
Shared memory	2MB
Multiport page-memory	8 ports
	64MB with 512-byte pages
	5MB/sec/port transfer speed
Multiport page-memory Buffer	12KB
Disk Unit Capacity	47MB, formatted

placed adjacent to Mu-X and connected to the conventional shared memory. The hardware's specification is summarized in table 1.

For the connection to PSI-II machines, Mu-X is connected to the PSI-Net. Mu-X is located at the Toshiba Research and Development Center in Kawasaki City adjacent to Tokyo. Mu-X is connected to ICOT's local PSI-Net via gateways (Figure 3).

3 SOFTWARE

3.1 Design Goals

FUNCTIONALITY

(1) Term data support

As the system is based on the relational knowledge model, capability of manipulation of terms is essential. Additional operations to normal comparison operations are required. The unification operation is the most basic operation associated with terms; the operations such as list membership check must also be supported if we try to use terms in a database context. Figure 4 shows an example of the internal representation of a term.

We also adopted the term representation in various system data structure. For example, in the data dictionaries the information about a relation is stored in a tuple and the information about its attributes is stored in an attribute using the term structure. The messages exchanged with a host machine are also represented using the term structure.

salad(tomato(2),lettace(1))

0111	000000001000
00000010	"s"
"a"	"l"
"a"	"d"
0111	000000001010
00000010	"t"
"o"	"m"
"a"	"t"
"o"	00000000
0000	000000000010
0111	000000001010
00000001	"l"
"e"	"t"
"t"	"a"
"c"	"e"
0000	000000000001

Figure 4. An Example of a Term

(2) Variable-length record support

This is also a derivation from the term data support. The attribute length of a term attribute cannot be defined at the schema definition time. This is because the length of a term may drastically change when a unification is performed. The data structure is determined so that the variable-length records can be manipulated with the least loss of efficiency.

(3) Multitransaction support

Considering the environment that this machine is used, an efficient realization of multitransaction facilities is pursued.

EFFICIENCY

(1) Parallel processing

Parallel processing algorithms on multiprocessor database machine have been extensively studied and implemented [Boral 82], [Hanson 87], [Nakamura 87], [Wilkinson 87], [Bitton 83a], [Kitsuregawa 84], [Shapiro 86]. The algorithms are for the most part aimed at improving the response time of a query. In our parallel processing scheme, we are not only interested in the response time but also the throughput of the system.

(2) Minimizing software overhead

The motivation behind this goal is that the conventional operating systems are not suited for the construction of special-purpose software, in particular database or knowledge base machine implementation. The management software is designed and coded with this strongly in mind.

(3) Effective use of the hardware architecture

As is described in the previous section, a feature of the hardware is that it has the hybrid shared memory systems of different nature. The management software aimed to make most of the memory systems. Care is also taken to preserve the scalability as much as possible. Hardware having the same architecture with more PEs could run the software with increased performance.

3.2 Basic Design

DECENTRALIZATION OF FUNCTIONS

Most of the multiprocessor database systems have a control processor and a number of data processors or PEs [Su 88]. The control processor is usually responsible for general and miscellaneous management tasks such as transaction management, data dictionary management, query compilation, parallel execution control and response generation. The PEs are responsible, on the other hand, for database operation execution.

The configuration, in a multi-transaction environment, has the disadvantage that the processing power of the control processor is not enough for managing the many PEs and becomes a performance bottleneck.

Some systems such as DBC/1012 adopt multiple control processors approach to eliminate this bottleneck. Conversely, the processing power of the control processors are not used for the processing of queries. To solve this problem, we assigned both the control processor functions and the data processor functions to each PE. Each PE is in charge of at most one transaction for the control processor tasks. When a host computer issues a transaction, the front end processor seeks an idle PE to be responsible for it. The PE becomes the "transaction master" of that transaction and takes care of the transaction until the end of the transaction. The front end processor is only responsible for the manipulation of network protocols.

CONFIGURATION OF THE PE SOFTWARE

Since each PE has to work on the control processor and data processor tasks concurrently, the execution of each task should be interchanged with little software overhead. We decided not to use the task switch mechanism provided by the residing operating system. Our solution was that the software consist of two modules, the transaction management module and command processing module, and let the transaction management module call the command processing module at certain program states. Thus the two modules actually comprise a single control flow program where the transaction management program is the main program with the command processing module subroutine. The flowchart of the main program is shown in Figure 5. The following is a brief explanation of the chart.

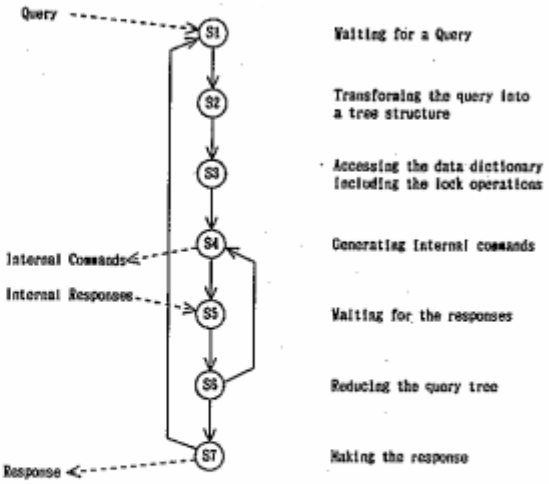

Figure 5. Program Main Flow

State S1: idle (waiting for the arrival of a query)

State S2: query transformation to a query tree

State S3: data dictionary consultation and relation locking, if necessary

State S4: internal command(s) generation

State S5: idle (waiting for the responses of the internal command(s))

State S6: reduces the query tree and go to S4 if there remains query tree to be processed

State S7: return response to host, go to S1

The transaction management module calls the command processing module at states S1, S3 (only if the lock operation is suspended), and S5. The command processing module, when called, seeks the internal commands which the PE can, or is supposed, to process. (See the internal command description for details.)

When there is no such internal commands, the control is returned to the management module.

The transaction management module generates and dispatches the internal commands step by step as it reduces the query tree. Optimizations in query processing can be better applied in the process than in the case where internal commands are generated in batch before the query processing, because the temporary relations' sizes are exactly known after a tree reduction. The size information can be used to determine the algorithm for successive operations.

We adopted a single control flow program approach mainly to eliminate task switching overhead between transaction management task and command processing task. The other advantage is that the multiport page-memory interface memory, which is the I/O buffer of the multiport page-memory, can be fully utilized by the current task since the execution is exclusive.

3.3 Internal commands for parallel processing

Internal commands are specifications of relational algebra level operations that PEs should process in parallel. According to the nature of the commands, they are classified into two types, PE specific and PE nonspecific commands.

(1) PE Specific and Nonspecific Commands

A PE specific command is concerned with accessing a portion of a relation stored in the disk. It must be processed by the PE(s) which owns the data in the disk. Therefore a PE specific command has an assignment mask, a bit array which specifies which PE(s) should process it.

A PE nonspecific command is concerned with only those relations stored in the multiport page-memory. In this case it can be processed by any PE. Idle PEs are responsible for processing such commands, which helps balance the processing load among the PEs. Note that both PE specific and nonspecific commands are shared by PEs. This requires a synchronization mechanism as described below.

(2) Assignment, Participation and Completion Masks

These mask fields are provided within the internal command used for synchronization. As the nature of parallel processing, it is not obvious to know when an internal command is finished. For example, as for a PE specific command, there may be a case where some of the PEs have already finished the processing while the others have not even started it yet.

A PE specific command is associated with an assignment mask and a completion mask. The setting of the assignment mask bit indicates that the corresponding PE must process the PE specific command. The completion mask bit is set by a PE that finished the command. If those coincide, it indicates the completion of the command. Similarly, the PE nonspecific command is associated with a participation mask and a completion mask. The participation mask bit is set when a PE participates in the PE nonspecific command processing. The meaning of the completion mask is the same as in the PE specific command. If those coincide, the PE nonspecific command is known to be completed. Using these masks, PEs can suspend the execution of an internal command, which helps to prevent the average response time from getting long in bad situations. In other words, this is part of a mechanism that substitutes the multi-tasking mechanism in a conventional operating system. There is a limit time for a PE to continue the processing of one internal command. If the time exceeds the limit, the PE looks for a queued PE specific command related to itself. If such a command is found, the PE begins the processing of the new command by leaving the completion mask of the current command unchanged.

(3) Object Counter

There is a counter field called the object counter within a PE nonspecific internal command. This is used to specify the current object (page) number ready for processing. For example, in a selection operation which requires the scan of a relation within the multiport page-memory, each page of the relation must be processed only once by an arbitrary PE. In this case, the object counter is initialized to the first page number. Each PE tries to get the content of the object counter (i.e. the page number to be processed next) and to increment it by one in a critical section of its program.

3.4 Data Objects

It is crucial to high performance to effectively use the hybrid memory systems, namely, conventional shared memory and the multiport page-memory.

The multiport page-memory behaves very much like a disk-cache for data stored in the disks; however, strictly speaking, it is not a disk cache. Rather, it is used as a large-capacity buffer. This means that the replacement is under control of the control software at any time. In the following, the major data objects in the system are described with a focus on where they are stored.

(1) Temporary Relation

A temporary relation is created typically by a retrieval operation. It is stored in a "multiport page-memory file", MPPM file for short, which is a sequence of pages allocated within the multiport page-memory.

Control tables are used to get the required page location within the multiport page-memory 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 the current implementation.

(2) Permanent Relation

A permanent relation is horizontally divided and stored across the disks. Hash-based or round-robin partitioning can be specified at the relation schema definition time. Within a PE, the partitioned portion of a permanent relation is stored in a "disk file". A disk file is basically a sequence of fixed-size pages within a disk device.

In an update operation, the new version of a page data is created and stored in a new page of the multiport page-memory. When the transaction is committed, the new version of the page data is actually stored in the disk device.

(3) Temporary Cluster

A set of temporary clusters are generated in a dynamic clustering operation. Each cluster is stored in a "bucket file". A bucket file is stored in the multiport page-memory like the MPPM file. The control tables, however, are simpler than that of the MPPM file, because their use is limited.

(4) Data Dictionaries

There are two types of data dictionaries, the local data dictionary and the global data dictionary.

(a) Local Data Dictionary

A local data dictionary is associated with a transaction and mainly keeps the information 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 transaction.

(b) Global Data Dictionary

The global data dictionary mainly keeps the information about the permanent relations. It is a special permanent relation, partitioned across the disks as other permanent relations, and loaded into the multiport page-memory at the system startup time. To process a query which requires access to the global data dictionary, the related portion of the global data dictionary is further copied to the shared memory and is used. The shared memory works as a cache for the global data dictionary. So, once a portion is copied to the shared memory, successive access to the portion goes to the shared memory. This can be determined by examining a hash table. The updates to the global data dictionary are reflected to the cached portion in the shared memory, if it exists, and to the copy in the multiport page-memory.

In summary, we have designed and been implementing the software that we believe is appropriate for knowledge base or database purposes. For the recovery facilities, the transaction recovery mechanism is included. We took into consideration the logging facilities for system recovery, however, it is not included in the implementation.

4 PERFORMANCE EVALUATION

We have done a preliminary performance evaluation using the Wisconsin Benchmark database [Bitton 83b]. According to our data storage scheme, the fixed-length database is stored using variable-length record format though the variable-length field is never used. A record header is attached to each record, while some attributes are stored using a two-byte short integer format against the original four-byte field.

The size of the relation we used is one thousand tuples (Thoustup relation). This is because we used the database that is also used for debugging. The Thoustup relation is stored in 112 2K-byte pages. Each page contains 9 tuples except for the last one.

The evaluation is done without any indexing scheme. That is, the values are all "nonindexed". A full scan of a relation is performed in the selection case and a paged nested-loop algorithm is used in the join case. In all the cases the processing times are measured assuming "cache hit", in other words, assuming that the relation already resides in the multiport page-memory. The evaluation was done in a single-user environment.

We measured the execution times of three queries as shown below. Two of them are selections and one is a join. The join is performed between the copies of the same Thoustup relation. The temporary relation is formed in the multiport page-memory. The values include processing times of all tasks required to execute a query within *Mu-X*, such as query compilation, communication and synchronization. The query transfer time from PSI-II and result transfer time are not included.

Query 1 (1% selection)

```
insert into temp select * from Thoustup
where unique2 between 101 and 110
```

Query 2 (10% selection)

Table 2. Evaluation Results

query	PE count	1	2	4	8
1	total	197ms	110ms	68ms	49ms
	manage.	24ms	24ms	24ms	25ms
	proc.	173ms	86ms	44ms	24ms
2	total	255ms	139ms	83ms	57ms
	manage.	24ms	24ms	24ms	25ms
	proc.	231ms	115ms	59ms	32ms
3	total	85.2s	43.1s	21.2s	10.9s
	manage.	44ms	44ms	44ms	46ms
	proc.	85.1s	43.0s	21.1s	10.9s

```
insert into temp select * from Thoustup
where unique2 between 101 and 200
```

Query 3 (join)

```
insert into temp
select Thoustup1.*, Thoustup2.*
from Thoustup1, Thoustup2
where Thoustup1.unique1 = Thoustup2.unique1
```

Table 2 shows the evaluation results. The total execution time, management task time and net processing time for each case are given. As the used relation size is small, the management task times are comparable to the processing times in the selection cases. According to our nonindexed selection processing scheme, much of the transaction management task done by the transaction master PE remains the same regardless of the size of the target relation. So we can extrapolate the selection processing time to the larger target relation sizes as long as the disk buffer (multiport page-memory) size is reasonably larger than the size of the target relation. We can predict, then, the processing times of 0.27 second for 1% selectivity selection and 0.35 second for 10% selectivity selection against the ten-thousand-tuple (TenKtup) benchmark relation.

OBSERVATIONS

To give the reader the idea of the basic processor speed, the Dhrystone benchmark result of the PE is shown in table 3 in comparison with other processors.[1] The PE is by no means the fastest hardware to the today's standard. If the PEs were replaced with faster hardware of the day, substantial speedup could be obtained for the processor-intensive portion of the processing times.

Still, we think these values are satisfactory for the nonindexed selections. For the join query, because a nested-loop algorithm is used, the processing time is not as small as other latest evaluation values. However, we think that the values are quite reasonable with respect to the algorithm used and we are expecting a considerable speedup when the bucket-wise hash-join algorithm [Kitsuregawa 83] is completed.

[1]The values are shown for the relative comparison in our environment and not for comparison with the reported values elsewhere.

Table 3. PE Performance

Processors (All CPUs are 68020s.)	Dhrystone loops/sec
PE 12.5MHz, no cache, off-board memory	925
Sun3/50 15MHz, no cache, on-board memory	2437
Sun3/160 16.7MHz, no cache, on-board memory	3083

In this evaluation we omitted the disk access times. In the nonindexed cases, we estimate that the disk access time for the TenKtup relation is about 0.6 to 0.7 second in the eight processors case. In the current implementation, as the processing time and disk access time are serialized, the 10% selection time against the TenKtup relation will be about one second. In the join query there will be little effect on the evaluation result because we can store the whole relation in the buffer (multiport page-memory) after the first accesses to the relation.

In summary of this section, though there are only a few evaluation results available now, they show encouraging performance potential of the machine. This will be made further evident after more detailed and exhaustive evaluations.

5 DISCUSSIONS

SYSTEM ARCHITECTURE

Mu-X can be thought to have an architecture that has dynamic processor-memory assignment capability. This is because the portions of the shared memories (both the conventional shared memory and the multiport page-memory) can be used by any PE without explicitly transferring data. In this class of architectures, if the PEs cannot access the data at a rate that matches the PE's processing speed, the performance will become poor. If only the conventional shared memory is shared among PEs, the shared memory and the memory bus should have the bandwidth that pace the processing speed of multiple PEs.

This is considered to be a challenge to the memory system since the clock cycle of processors are becoming faster at a greater rate than the memory chips and memory bus speedup. Even single processor systems'

memory bus often does not have enough bandwidth for the processor's speed. The situation is worse in the multiprocessor case. Generally, cache memories are the solution to the problem; however, it is still questionable if the database processing manifests good memory access locality.

The multiport page-memory is a cost-effective solution to the memory-bandwidth enhancement. The cost-effectiveness results from the separation of ports while a common bus has to shorten the cycle time of bus transfers for high bandwidth. For example, assume that there is a shared bus N-processor multiprocessor with B memory banks each having T bytes/sec transfer capability. For each processor to read a page of K bytes in a buffer it requires $(K \times N)/(T \times B)$ time, provided that the shared bus has enough bandwidth. If N is equal to B, obviously the time is K/T. If the multiport page-memory is used, the time will be also K/T.[2] However, the bandwidth of the shared bus must be more than $T \times B$ bytes/sec to fulfill this. In other words, the shared bus must have the technology that enables $T \times B$ bytes/sec transfer. While the multiport page-memory's port and the switching network must only have T bytes/sec technology.

In the current implementation, the multiport page-memory has eight 16-bit ports each having 5MB/sec transfer capability. Thus in total the bandwidth is 40MB/sec. This is a figure that cannot be achieved using the same class of technology as was adopted to implement the experimental machine.

Overall, the architecture of Mu-X is similar to that of DIRECT [DeWitt 79], [Boral 82] with respect to the processor memory interconnection. The similarity is even stronger when compared to the DIRECT implementation; DIRECT used multiport memory instead of original crossbar connection of the CCD modules and the query processors (or PEs).

The difference with the hardware is the PE configuration. The PEs of Mu-X are provided with separate disks that the PEs can access in parallel. We did not include disk access times in the evaluation, so it is too early to claim the advantage that the parallel disks will bring. However, we are certain that the parallel disks will reduce the I/O bottleneck [Agrawal 84] and can be used effectively in (1) nonindexed query processing and (2) multiuser query processing.

The difference in software is that (1) Mu-X does not have a centralized control processor and (2) neither conventional operating system nor modules of existing database system are used. The effectiveness of the former is not yet proven in the single-user evaluations. The effectiveness of the latter, on the other hand, is shown in the evaluation values.

POSSIBLE IMPROVEMENTS

Though we are not through with the full implementation of the software and evaluations, we can point out some improvements to the machine architecture and software design. For hardware, it is matter of course that the processing speed would be faster if we used a processor board with a processor cache. To do this, care must be taken for the shared memory caching to keep the coherency of the data as in the case of usual shared memory multiprocessors. One simple way to do this is to avoid caching the shared memory. For the local memory within a PE, information is never shared among PEs; so no coherency problem occurs. We will not discuss this further because the discussion may be too general to be done here.

Another possible improvement is the multiport page-memory buffer allocation in the PE memory space. In the current implementation, in each PE there is a separate memory (multiport page-memory interface memory) for the input/output buffer use. The memory is implemented with dual-port RAMs for enabling the simultaneous access from both PE processor and multiport page-memory port. As a multiport page-memory is an electronically rotating device, the interference of its rotation causes data loss. The usage of dual-port RAMs was a simple solution to the problem. However, the software has come to be responsible for the data page transfer from the dual-port RAM buffer to the local memory for further processing. This transfer is a source of performance degradation.

To remedy this, it is possible to provide hardware which takes care of the buffering of multiport page-memory port data. In that case, a destination buffer in the local memory can be assigned as the destination of multiport page-memory page transfer. When the page transfer begins, the hardware temporarily buffers the data sent from the multiport page-memory port and forwards it to the destination buffer on the fly. The data loss probability can be made reasonably low by appropriately designing the hardware's buffering capability.

The software leaves room for optimization for query processing algorithms. There is a plan of incorporating some of the sophisticated algorithms found in the literature. For example, the bucket-wise hash-join algorithm

[2] For the processor connected to a multiport page-memory, K/T is the time that the processor must wait for a page to be filled in a buffer. While the processor connected to a shared bus does not have to wait for a buffer to be filled if the processor's program directly accesses the page in shared memory. In this case, if P is the time for a processor to process a page, then $K/T + P$ is the time for the processor connected to the multiport page-memory to finish the processing of a page. For a processor connected to shared memory, P is the time to finish the processing of a page. For multiple-page processing, double buffering can be used to make the overhead (K/T) effectively negligible.

[Kitsuregawa 83] is being implemented. The scheme will also be used for operations such as projection and set difference. Other algorithms will be incorporated that help increasing the processing efficiency.

For indexes, a hash-based primary index is being implemented. User-specifiable secondary indexes will also be implemented.

For the basic control mechanism described in section 3.2, there are some points where there is room for further optimizations. For example, currently, disk access specified by PE specific command and internal command processing specified by PE nonspecific command are serialized. This is done because we did not want to run the residing operating system's interrupt routine which is very slow, and we suspected that it would make the software too complex.

6 A SAMPLE INFORMATION RETRIEVAL SYSTEM

To evaluate performance in a real environment and to verify the effectiveness of the unification-based query language [Monoi 88], we used a sample information retrieval system. The knowledge base we built is a technical report database. A QBE-like user-interface is used on the PSI-II using the multi-window facility.

The technical report database contains technical reports from overseas research institutes, which were sent to ICOT on an exchange basis. The database consists of `report( author, title, institute_name, keywords, report_no, date_of_issue, received_date, reference)`, `reference(author, title, institute_name, report_no, date_of_issue)`, and `institute( institute_name, address, research_topic, members)`. Some of the attributes that have multiple instances, for example, author, keywords, reference, are represented using list structure.

First the user selects the relation(s) concerned with the query. The schema(s) of the relation(s) appear with attribute names. A simple qualification condition such as selection to a value can be specified in the schema window. The inter-relation relationship is specified by means of common variables (specifying unification-join) or by assigning separate variables and relating them in the condition window. For example, to check if a keyword is contained in the keyword list attribute the variable for the keyword (`Keyword`) and the keyword list (`Keyword_list`) are related by writing a membership predicate "`member(Keyword, Keyword_list)`". The output attributes are specified in the result window. The query is generally constructed using the mouse.

In the query example shown in Figure 6, a query is specified that retrieves the author list, title and report number of reports that contain "`database`" in the keywords and that are issued by institutes that research artificial intelligence. The variable numbers such as `Z101_1` are computer-generated because these attributes are specified (mouse-clicked) to be the result relation attributes without explicitly given names by the user. In Figure 6, the query language form of the query is shown in the execution window. This translated query is sent to Mu-X via the knowledge base machine interface classes. The result relation is obtained in the Mu-X and successive "`get`" commands retrieve the result relation into the PSI-II machine and display it.

Throughout the experience, though this is only a prototyped system, we could see that the query language is powerful enough for the implementation of an application system. Actually, the transformation of the QBE-like query to the query language was quite straightforward; it took only a week to complete that part of the system.

7 CONCLUSION

We have described the hardware, software and evaluation of an experimental parallel knowledge base machine. This can be thought of a parallel relational database machine with an extended data type. The hardware employed a hybrid shared memory multiprocessor architecture. With the software dedicated to the parallel knowledge base operations, the system exhibits a good potential for knowledge/database operations. The effectiveness of the multiport page-memory has been shown by the almost linear speedup in the net processing times of queries.

However, as is mentioned in the foregoing sections, the machine is, at the time of this writing, still under development. The more detailed and thorough evaluations are needed to examine the pros and cons of the system.

References

[Agrawal 84] Agrawal, R., DeWitt, D. J., "Whither hundreds of processors in a Database Machine?", *Proc. Int'l Workshop on High-Level Architectures*, Los Angeles, 1984.

[Bitton 83a] Bitton, D., Boral, H., DeWitt, D. J., Wilkinson, W. K., "Parallel Algorithms for the Execution of Relational Database Operations", *ACM Transactions on Database Systems*, Vol. 8, No. 3, pp. 324-454.

[Bitton 83b] Bitton, D., DeWitt, D. J., Turbyfill, C., "Benchmarking Database Systems: A Systematic

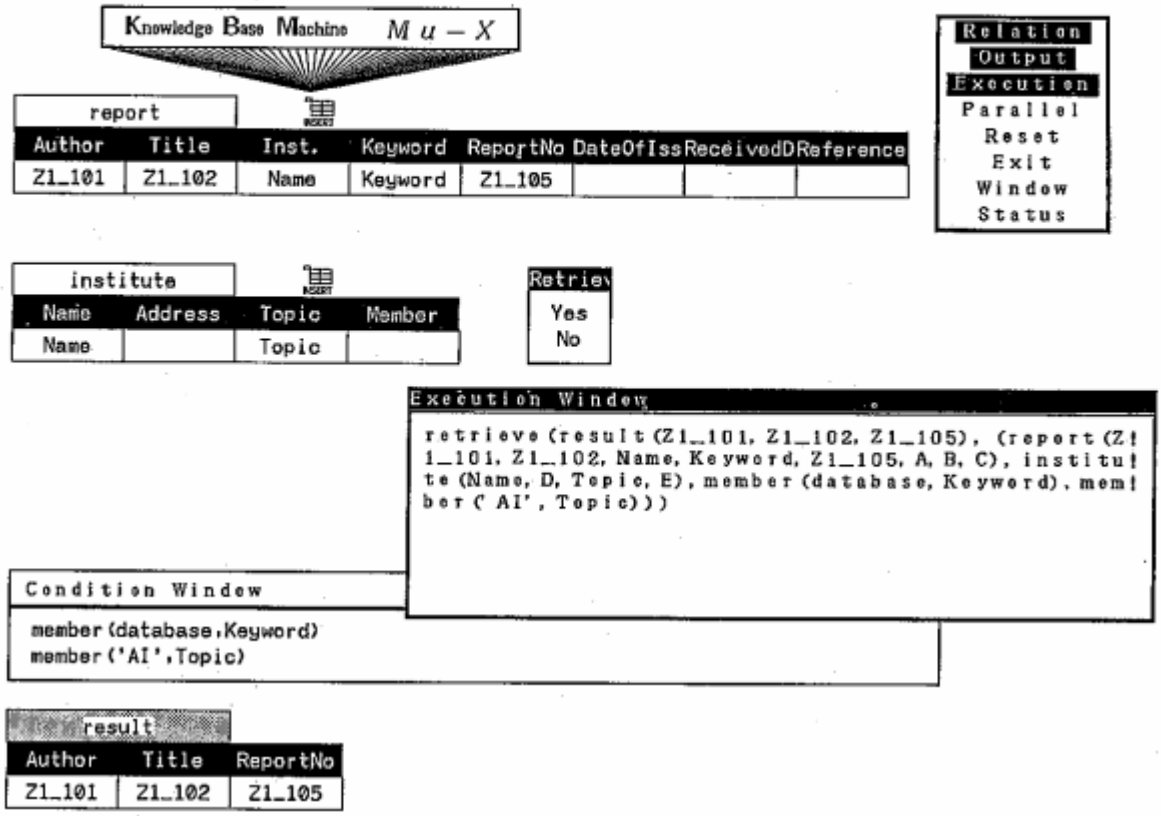

Figure 6. A query example

Approach", *Proc. VLDB*, 1983.

[Boral 82] Boral, H., DeWitt, D. J., Friedland, D., Jarrel, N., Wilkinson, W. K., "Implementation of the database machine DIRECT", *IEEE Trans. on Software Engineering*, vol. SE-8, no. 6, Nov., 1982.

[DeWitt 79] DeWitt, D. J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems", *IEEE Trans. on Computers*, vol. C-28, June, 1979.

[DeWitt 86] DeWitt, D. J., Gerber, R. H., Graefe, G., Heytens, M. L., Kumar, K. B., Muralikrishna, M., "GAMMA: A High Performance Dataflow Database Machine", *Proc. 12th VLDB*, Kyoto, 1986.

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

[Itoh 88] Itoh, H., Monoi, H., Shibayama, S., Miyazaki, N., Yokota, H., Konagaya, A., "Knowledge Base Subsystem Based on Logic Programming", *Proc. FGCS'88*, Tokyo, 1988.

[Kakuta 85] Kakuta, T., Miyazaki, N., Shibayama, S., Yokota, H., Murakami, K., "The Design and Implementation of Relational Database Machine Delta", *Proc. Fourth International Workshop on Database Machines*, 1985.

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

[Kitsuregawa 83] Kitsuregawa, M., Tanaka, H., Motooka, T., "Application of Hash to Data Base Machine and Its Architecture", *New Generation Computing*, vol. 1, no. 1, 1983.

[Kitsuregawa 84] Kitsuregawa, M., Tanaka, H., Motooka, T., "Architecture and Performance of the Relational Algebra Machine GRACE", *Proc. International Conference on Parallel Processing*, 1984.

[Monoi 88] Monoi, H., Morita, Y., Itoh, H., Takewaki, T., Sakai, H., Shibayama, S., "Unification-Based Query Language for Relational Knowledge

Bases and Its Parallel Execution", *Proc. FGCS'88*, Tokyo, 1988.

[Morita 86] Morita, Y., Yokota, H., Nishida, K., Itoh, H., "Retrieval-by-Unification Operation on a Relational Knowledge Base", *Proc. 12th VLDB*, Kyoto, 1986.

[Morita 87] Morita, Y., Oguro, M., Sakai, H., Shibayama, S., Itoh, H., "Performance Evaluation of a Unification Engine for a Knowledge Base Machine", *ICOT Technical Report*, TR-240, 1987.

[Nakamura 87] Nakamura, S., Minemura, H., Minohara, T., Itakura, K., "A High Speed Database Machine - HDM", *Proc. Fifth International Workshop on Database Machines*, Karuizawa, 1987.

[Sakai 87] Sakai, H., Shibayama, S., Monoi, H., Morita, Y., Itoh, H., "A Simulation Study of a Knowledge Base Machine Architecture", *Proc. Fifth International Workshop on Database Machines*, Karuizawa, 1987.

[Shibayama 84] Shibayama, S., Kakuta, T., Miyazaki, N., Yokota, H., Murakami, K., "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor", *New Generation Computing*, Vol. 2, No. 2, 1984.

[Shibayama 85] Shibayama, S., Sakai, H., Iwata, K., "A Knowledge Base Architecture and its Experimental Hardware", *Proc. IFIP TC-10 Working Conference on Fifth Generation Computer Architectures*, Manchester, 1985.

[Shibayama 87] Shibayama, S., Sakai, H., Monoi, H., Morita, Y., Itoh, H., "Mu-X: An Experimental Knowledge Base Machine with Unification-Based Retrieval Capability", *Proc. 2nd France-Japan Computer Science and Artificial Intelligence Symposium*, Cannes, 1987.

[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.

[Su 88] Su, S. Y. W., *Database Computers Principles, Architectures and 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", *Proc. Fifth International Workshop on Database Machines*, pp.29-42, 1987.

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