TR-156

# A Large-Scale Knowledge Base Machine Control Technigue Using Multi-Port Page-Memory

by
H. Monoi, H. Yokota, M. Murakami
and
H. Itoh

February, 1986

# A Large-Scale Knowledge Base Machine Control Technique Using Multi-Port Page-Memory

Hidetoshi MONOI, Haruo YOKOTA, Masaki MURAKAMI, Hidenori ITOH

Institute for New Generation Computer Technology(ICOT)

Mita Kokusai Building, 21F

1-4-28 Mita, Minatoku, Tokyo 108 Japan

February, 1986

## Abstract

The architecture and control of a knowledge base machine are discussed. The machine retrieves knowledge from a relational knowledge base, and consists of a number of dedicated hardware units called unification engines, several disk systems, a control processor, and a multi-port page memory. The multiport page-memory located between dedicated processors and disk systems enables efficient retrieval from data streams.

The aim of control is to achieve parallel execution on the unification engines and disk systems for retrieving knowledge stored in the disk systems. We propose an event-driven control technique for allocating unification engines to decrease control overhead.

## 1 Introduction

The research and development project for the Fifth Generation Computer Systems in Japan aims to establish inference and knowledge base functions for knowledge information processing. ICOT (Institute for New Generation Computer Technology) is currently developing a knowledge base machine as one of the fundamental elements of the system.

The knowledge base machine is a specialized machine that stores massive amount of knowledge in secondary memory devices, and executes high-speed knowledge retrieval in response to requests from host systems. In systems which manage large amounts of data like this, one of the major problems concerning performance is the bottleneck between the main memory and the secondary memory [1] as well as what is called the von Neumann bottleneck.

In order to resolve the bottlenecks, we have decided to adopt multiport page-memory[2] and proposed the economic and practical way to configure it[3]. A multiport page-memory is capable of simultaneously accessing multiple secondary memory devices and data processors. It prevents access collision by allowing access to a page (collection of words) and not a word. We also propose a dedicated processor to retrieve data from the knowledge base, which performs pipeline processing on the data stream.

This paper describes the configuration and control techniques for a knowledge base machine utilizing the multiport page-memory and multiple dedicated processors. Chapter 2 discusses retrieval methods proposed for a relational knowledge base. Chapter 3 covers the configuration of a knowledge base machine utilizing multiport page-memory and multiple dedicated processors, while Chapter 4 presents the method for retrieving knowledge in parallel and the control technique for allocating processors in the event-driven manner to decrease the control overhead.

```
$R \Leftarrow \Phi$;                                  /* Initialize output area.                    */
$T_0 \Leftarrow \sigma_{head \Diamond goal}(S)$;      /* Unification-restriction by goal clause.    */
$i \Leftarrow 0$;                                     /* Initialize iteration counter.              */
while $T_i \neq \Phi$ do                              /* Iterate while $T_i$ is not empty.          */
 begin                                                /*                                            */
  $T' \Leftarrow \sigma_{body=[\ ]}(T_i)$;            /* Restriction to get result.                 */
  $R \Leftarrow R \cup T'$;                           /* Store it into output area.                 */
  $T'' \Leftarrow T_{i\ body} \overset{\bowtie}{\Diamond}_{head} S$;   /* Unification-restriction.      */
  $T_{i+1} \Leftarrow \Pi_{T_i \cdot head, T \cdot body}(T'')$;    /* Projection for next iteration.    */
  $i = i + 1$;                                        /* Increase iteration counter.                */
 end                                                  /*                                            */
```

Figure 1. Example for Relational Knowledge Base Retrieval

## 2 Relational Knowledge Base Retrieval

The knowledge base machine currently under investigation at ICOT is designed to retrieve data from a relational knowledge base composed of *term relations*. A term relation is the expansion of the elements of a relation in a relational database to a term (a structure containing variables)[6]. A relational knowledge base enables manipulation of data by set operations and efficient retrieval from large volumes of data, while a Prolog machine manipulating large volumes of data[5] executes unification efficiently. It is also suitable to retrieve knowledge on stream-formed data by a dedicated processor as in a relational database machine.

Operations used in the relational knowledge base are *unification-join*, *unification-restriction*, projection, and so on. The unification-join and unification-restriction operations are based on the unification operation corresponding to the equality checks in relational algebra. The projection operation is the same as in conventional relational algebra. An example of a knowledge base retrieval utilizing these operations is given in Figure 1[4]. This also performs resolutions. In this example, term relations consisting of 2 attributes (head and body) are used to store the Horn clauses, and retrieval is executed by unification-join and unification-restriction operations. In this example, $S$ is the source relation of the retrieval, $R$ is the relation holding the resultant data, and $T_i$, $T'$, and $T''$ are temporary relations holding intermediate resultant data. The restriction operation is indicated by $\sigma$, the projection operation by $\pi$, the join operation by $\bowtie$, and the unification conditions by $\diamond$.

We call the knowledge retrieval operations based on unification operation such as unification-join and unification-restriction Retrieval By Unification (RBU) operations[6].

From now on, we use terms of *data* and *relation* to refer to the term relation introduced in this chapter.

## 3 Architecture

### 3.1 Unification Engine

In relational knowledge base retrieval, processing loads of the search scheme in join operation are extremely high, just as for conventional relational databases. In addition, as the unification-join operation requires the unification operation as well, a major increase in processing data is expected. In order to assure a reasonable response time to a host processor, it is therefore essential to develop methods to handle processing involving such major processing loads as quickly as possible.

In response to this problem, we propose to use a special-purpose hardware device capable of handling knowledge retrieval processing, including unification-join and unification-restriction operations[7]. It has three channels for input or output of data and performs retrieval processing on input data streams from two channels and issues the result to the other channel. This special-purpose hardware executes RBU operations and also executes relational algebra operations such as those realized in the relational database engine developed for the relational database machine Delta[8]. This new device is referred to as a *Unification Engine* (UE).

The UEs can relieve the system control processor handling large amounts of data. The control processor has only to control the entire system and manage interfacing with the host processor.

### 3.2 Multiport Page-Memory

For a UE to achieve its potential processing power, high-speed data transfer between the UE and memory device is essential. Especially in knowledge base and database machines, where data is stored in secondary memory devices, the gap of speed between processing and data transfer becomes a major problem. To resolve this problem, a buffer memory could be located between the processors and the secondary memory devices. When processors, buffer memory and secondary memory devices are all linked by a single data bus, data collision is likely to increase as the number of processors and processing speed increases, obstructing satisfactory system performance. For this reason, we propose multiport page-memory (MPPM) as the buffer memory.

As indicated in Figure 2 the MPPM is composed of multiple I/O ports, several memory banks, and a switching network connecting them. Access through an I/O port to a word is not allowed, only to a fixed number of words, which we call a

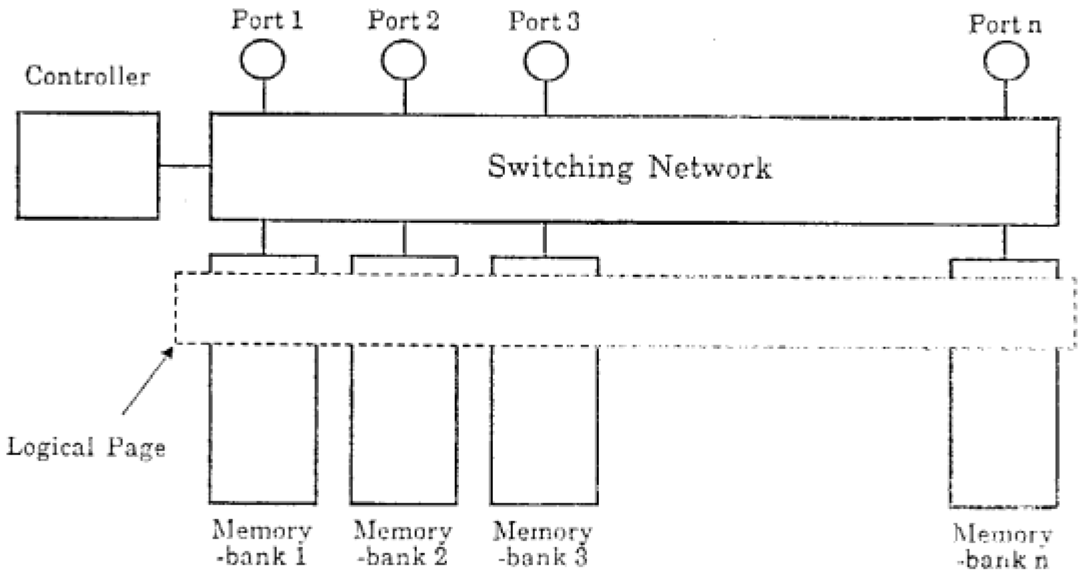


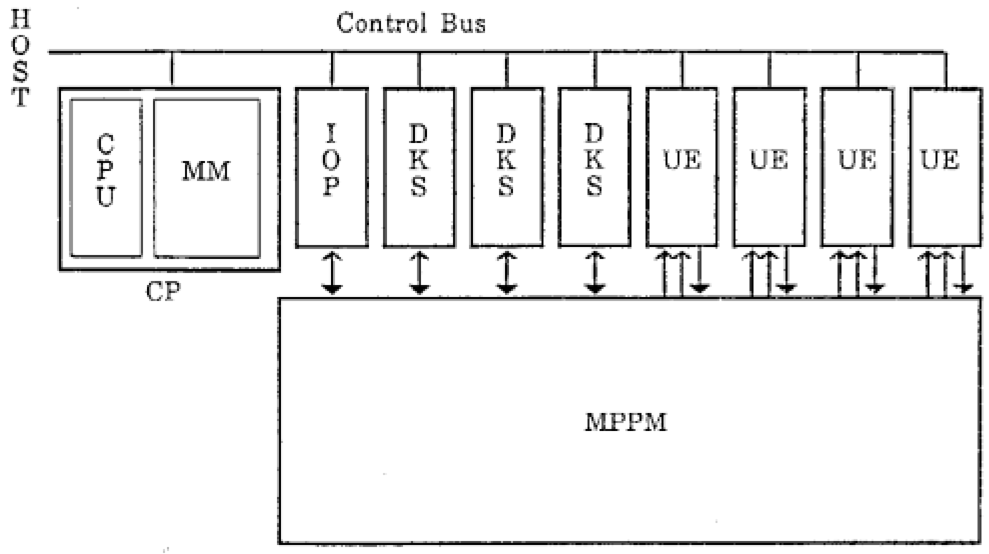
Figure 2. Multiport Page-Memory Configuration

logical page. Multiple I/O ports can access any given page at the same time. All logical pages are grouped in the memory bank (Figure 2). Logical pages defined over the memory banks horizontally are assigned contiguous page addresses. To avoid memory bank access collision, the switching network fixes the I/O port-memory bank connections for a fixed time.

This MPPM makes it possible tp link the UEs, buffer memory, and secondary memory devices without data collision. The MPPM also acts as a buffer memory between the UEs and secondary memory devices avoiding any data collisions.

### 3.3 Hardware Configuration

The configuration of the knowledge base machine equipped with the MPPM and UEs is shown in Figure 3. This machine is composed of several UEs, the MPPM, and a control processor (CP), multiple disk systems (DKS), and an I/O



| | | | |
|---|---|---|---|
| CP | : Control Processor | UE | : Unification Engine |
| MM | : Main Memory of CP | DKS | : DisK System |
| IOP | : I/O Processor | MPPM | : Multi-Port Page-Memory |

Figure 3. Knowledge Base Machine Configuration

processor (IOP). The DKS include secondary memory devices such as disks devises and handle data transmission to the MPPM. The IOP executes data transmission between the host systems and the MPPM, and the MM of the CP and the MPPM. This configuration provides the following features:

1. Multiple data buses between the UEs and the DKS eliminate data processing bottlenecks.

2. Simultaneous retrieval for UEs.

In the configuration in Figure 3, data flow is as follows. The data in the secondary memory device is staged by the DKS. The UEs then accept the data stream and output the resultant stream to the MPPM. The processing result is either re-input to the UE, sent to the secondary memory of the DKS, or the control processor MM through the IOP. The UEs and the DKS can independently access

MPPM pages, so that while the UEs are processing data on some pages, it is possible for the DKS to be staging data to the other pages. Thus we can provide an extremely efficient data processing system utilizing the MPPM.

Interface control between the knowledge base machine and the host and overall knowledge base machine control are handled by the CP. For this reason, the CP includes resource management control software to manage the MPPM and processors connected to the MPPM, as well as a knowledge base manager. These software packages control parallel knowledge retrieval using the processors connected to the MPPM. The CP, UE, DKS, and IOP all function as I/O devices. The CP issues processing commands to the individual processors, beside initiating processing. The CP keeps track of processor status by interrupts with control data. The control data transmissions between the CP and the individual processors are handled via the control bus.

## 4 Multiport Page-Memory Control Technique

As mentioned above, we locate the multiport page-memory between the unification engines and the disk systems as a buffer memory. In this configuration, it is possible for multiple UEs to retrieve data simultaneously. It is also possible for the disk systems to pre-stage the data from secondary memories, while the unification engines are retrieving the data in the multiport page-memory.

In the backend knowledge base machine proposed in this paper, it is common to execute several retrievals simultaneously in response to request from multiple users. In order to increase the efficiency of data retrieval, we must investigate the methods for executing retrievals in parallel utilizing several unification engines. From the point of view of multi-processor database systems, there are several levels of parallel processing, such as parallel processing by the transaction, relation, or page. Especially, parallel processing by relation and page are important for scheduling multiple processors.

In this chapter, we first discuss parallel processing capability in relation-sized units, and then we propose a scheduling method of the processors attached to the multiport page-memory for executing page-sized parallel retrieval in an event-driven manner. Lastly, we propose a multiport page-memory management method enabling efficient retrieval in page-sized units.

## 4.1 Parallel Processing in the Knowledge Retrieval

### 4.1.1 Parallel Search Capability

As described above, the proposed knowledge base machine includes multiple UEs, and therefore a major performance problem is how these UEs will operate in parallel during retrieval. It is possible for UEs and DKS to operate in parallel, so that the UEs retrieve data staged to the MPPM, and the DKS stage data required by the UE in the near future.

The retrieval example for a relational knowledge base in Figure 1 can process operations in parallel by staging all relations being used to the MPPM. Taking this parallel capability into account, it is possible to describe the retrieval procedure using the temporary relations $T_1$, $T_2$, $T_3$ and $R_1$ in Figure 4. That is, multiple UEs can execute retrieval simultaneously separating output relations. In Figure 4, individual nodes represent data processing handled by the UEs.

The relations used in the individual nodes in Figure 4 must be either staged by the DKS prior to UE execution, or may have already been staged. Using the

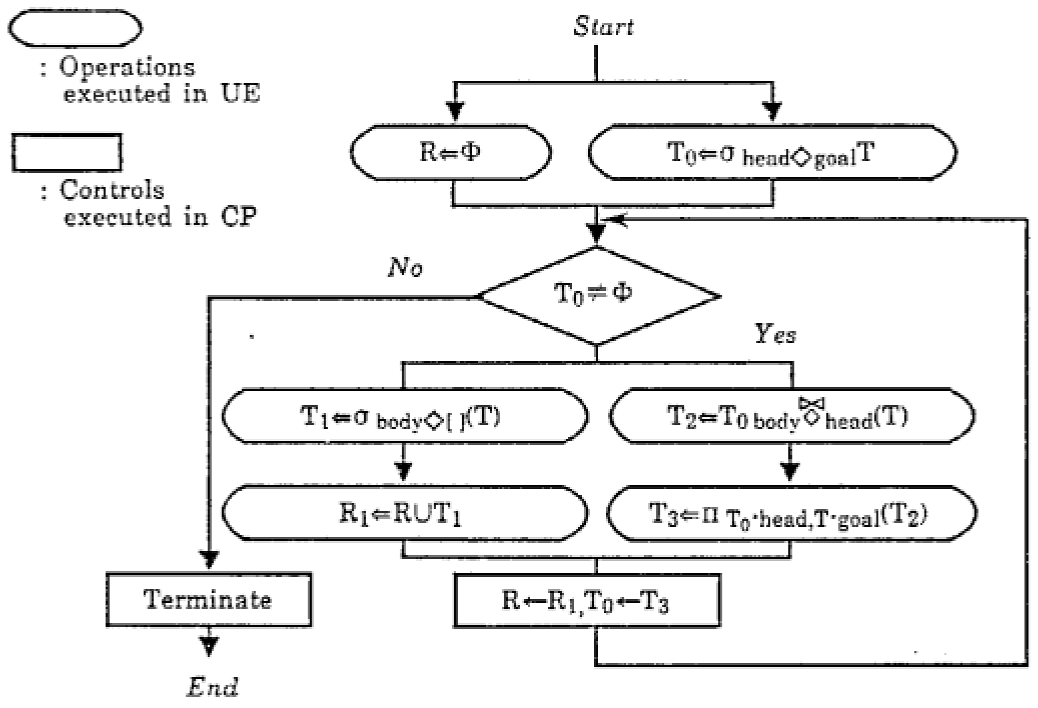


Figure 4. Retrieval in Relation-Sized Units

MPPM enables parallel operation of the UEs and the DKS, making it possible to run UE processing at one node while performing staging for other nodes.

To execute this parallel control, CP control can be divided into following three phases:

1. Node Execution Management,
2. Data Staging, and
3. UE Execution.

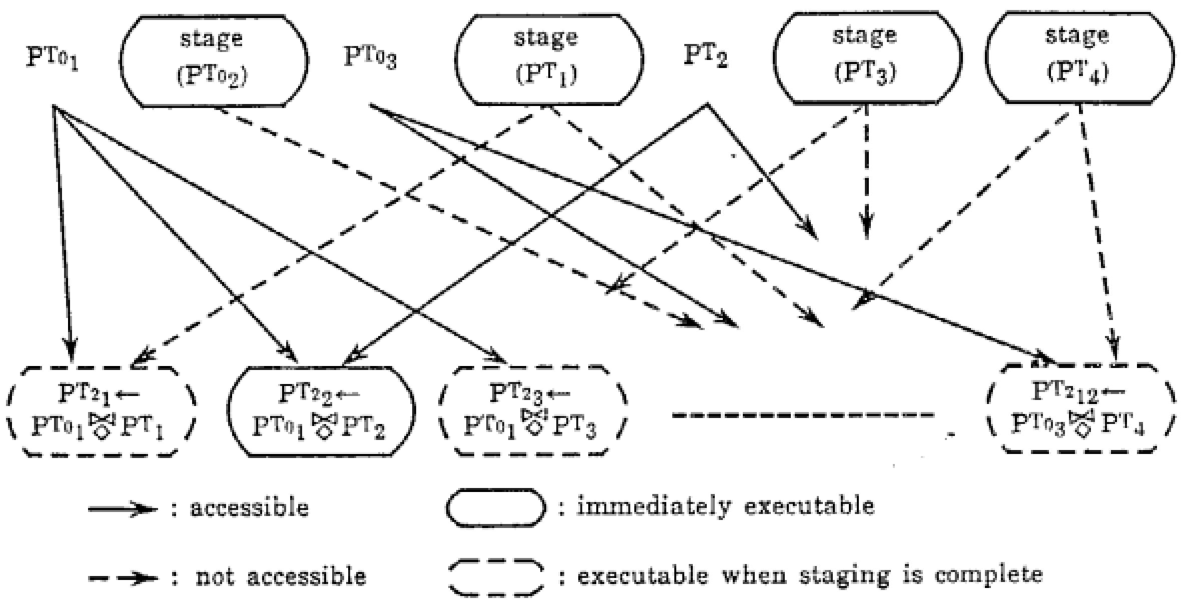
In the first phase of 'node execution management', a decision is made as to whether each node is executable or not. Here, '*executable*' means that input and output relations used in a node are ready to be processed. If a node is executable, then it is passed to the next phase. In the second phase of 'data staging', a determination is made as to whether staging is required or not. For nodes where staging is required, input requests are allocated to the corresponding DKS. Nodes are passed to last phase either when staging is not required or when staging by DKS is complete. In the last phase, free UEs are allocated to UE operations at each node.

### 4.1.2 Parallel Control in an Event-Driven Manner

The above section described the parallel processing capability in retrieval from the relational knowledge base. The MPPM can be accessed simultaneously in page-sized units, and relations can be grouped into page-sized unit as well. We can further increase the degree of parallel processing by introducing data retrieval between pages. However, as granularity of processing decreases, control overhead in allocating UEs is likely to increase. Therefore, it is important to establish an efficient method for allocation of UEs.

A number of database machines with an architecture consisting of a control processor, several dedicated processors, and connecting devices have been proposed[9][11][12]. Each of the dedicated processors plays a major part in processor allocation procedures within those machines. Control overhead such as inter-processor communication and procedures for allocating processors, however, are likely to increase as the number of transactions to be processed increases. We therefore propose a scheduling method in an event-driven manner. In this method the CP plays a major role as a master processor in allocating UEs for each retrieval. UEs are allocated to each retrieval in page-sized units and activated by commands from the CP.

Let us assume that for the node processing the unification-join indicated in Figure 4,



$PT_{0_1}$
stage ($PT_{0_2}$)
$PT_{0_3}$
stage ($PT_1$)
$PT_2$
stage ($PT_3$)
stage ($PT_4$)
$PT_{2_1} \leftarrow PT_{0_1} \bowtie PT_1$
$PT_{2_2} \leftarrow PT_{0_1} \bowtie PT_2$
$PT_{2_3} \leftarrow PT_{0_1} \bowtie PT_3$
$PT_{2_{12}} \leftarrow PT_{0_3} \bowtie PT_4$
: accessible
: not accessible
: immediately executable
: executable when staging is complete

Figure 5. Unification-Join in Page-Sized Unit

$$T_2 = T_0 \underset{body \diamond goal}{\bowtie} T$$

the input relations $T$ and $T_0$ are composed of pages as follows.

$$T = \{P_1^T, P_2^T, P_3^T, P_4^T\}$$

$$T_0 = \{P_1^{T_0}, P_2^{T_0}, P_3^{T_0}\}$$

In addition, $P_2^T$, $P_1^{T_0}$, and $P_3^{T_0}$ are already staged to the MPPM. Unification-join processing can be divided into page-sized unit processing steps to enable the processing in Figure 5.

In the individual nodes in Figure 5, the unification-join processing for $P_2^T$, $P_1^{T_0}$, and $P_3^{T_0}$ can be handled by the engines immediately, but the other nodes executing unification-join have to wait for staging by the DKS. In order to enable this type of waiting, the following *packet-form operation* is proposed.

| Condition | Operation | Post-processing |
|---|---|---|

Where objectives of each field of the packet are as follows.

*Condition:* The conditions indicating whether the operation indicated by the packets is executable or not.

*Operation:* The operation and relation pages to be executed by the UEs, DKSs, or IOP.

*Post-Processing:* The condition settings for the effects the result will have on other packets.

Using this packet the flow in Figure 5 develops into the packet string indicated in Figure 6. In Figure 6, $S(P)$ is the status of page $P$, $S(P) = T$ indicates that page $P$ is accessible, and $S(P) = F$ indicates that page $P$ is not. The initial state is $S(P) = F$ for every page. Futhermore, $stage(P)$ indicates the allocation of the page to page $P$ of the relation, and staging by the DKS.

These packets are distributed among the processors such as the UEs, DKS, and IOP executing operations indicated in the operation field. Every processor determines independently whether a packet is executable or not. When a packet becomes executable, operations indicated in the packets are sent to processors for actual processing. Processing results are passed on to other packets. This control flow is indicated in Figure 7, and the processing for the indicated boxes is as follows:

*Command Compile:* Retrieval requests for the knowledge base are supplied as a combination of retrieval operations in a relation-sized unit. The command compile divides those retrieval operations into staging processes and UE processes with the packet form by a page-sized unit. Finally, the command

| *Processor* | *Condition* | *Operation* | *Post-processing* |
|---|---|---|---|
| *DKS* | T | ; stage($P^{T}_{1}$) | ; S($P^{T}_{1}$)$\leftarrow$T |
| | T | ; stage($P^{T}_{3}$) | ; S($P^{T}_{3}$)$\leftarrow$T |
| | T | ; stage($P^{T}_{4}$) | ; S($P^{T}_{4}$)$\leftarrow$T |
| | T | ; stage($P^{T_o}_{2}$) | ; S($P^{T_o}_{2}$)$\leftarrow$T |
| *UE* | S($P^{T}_{1}$)=T&S($P^{T_o}_{1}$)=T | ; $P^{T_2}_{1} \Leftarrow P^{T}_{1}\,{}_{\text{head}}\overset{\bowtie}{\diamond}{}_{\text{body}}P^{T_o}_{1}$ | ; S($P^{T_2}_{1}$)$\leftarrow$T |
| | S($P^{T}_{2}$)=T&S($P^{T_o}_{1}$)=T | ; $P^{T_2}_{1} \Leftarrow P^{T}_{2}\,{}_{\text{head}}\overset{\bowtie}{\diamond}{}_{\text{body}}P^{T_o}_{1}$ | ; S($P^{T_2}_{2}$)$\leftarrow$T |
| | S($P^{T}_{3}$)=T&S($P^{T_o}_{1}$)=T | ; $P^{T_2}_{1} \Leftarrow P^{T}_{3}\,{}_{\text{head}}\overset{\bowtie}{\diamond}{}_{\text{body}}P^{T_o}_{1}$ | ; S($P^{T_2}_{3}$)$\leftarrow$T |
| | | -------------------- | |
| | | -------------------- | |
| | S($P^{T}_{4}$)=T&S($P^{T_o}_{3}$)=T | ; $P^{T_2}_{12} \Leftarrow P^{T}_{4}\,{}_{\text{head}}\overset{\bowtie}{\diamond}{}_{\text{body}}P^{T_o}_{3}$ | ; S($P^{T_2}_{12}$)$\leftarrow$T |
| | all S($P^{T_2}_{i}$)=T | ; Nop | ; S($T_2$)$\leftarrow$T |
| *CP* | S($T_2$)=T | ; <Terminate Process> | |

Figure 6. Retrieval in the Packet Form

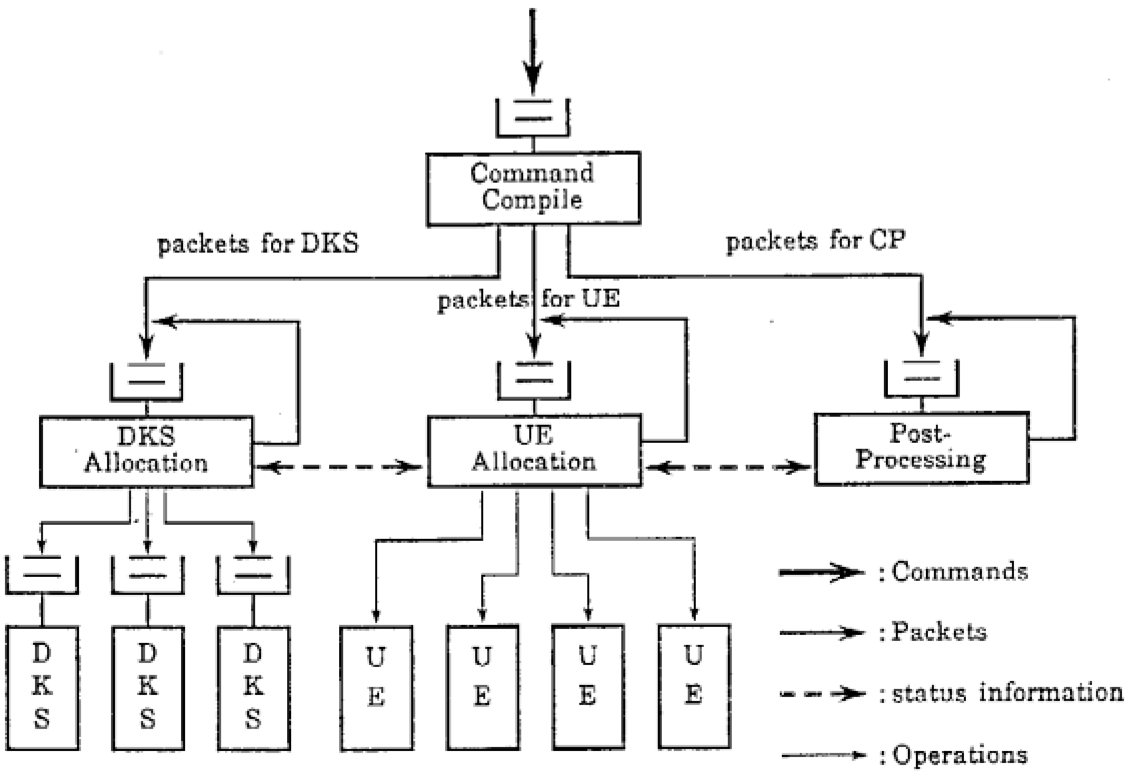


Figure 7. Control Flow for Page-Sized Retrieval

compile distributes the packets to each processor allocation process corresponding to the operation field.

*DKS Allocation:* DKS allocation examines the condition of each packet and determines whether staging is executable or not. If staging is executable, a DKS is allocated according to the object pages and operations are sent to the DKS driver. When staging is complete, the status of the page is sent to the UE allocation process.

*UE Allocation:* UE allocation also examines the condition of each packet and determines whether the operation is executable or not. If the operation is executable, a free UE is allocated and operations are sent to the UE driver. When execution of the operation is complete, status is sent to the post-processing process.

*UE:* UE means actual driving of UEs. It includes interruption handling for actual unification engines.

*DKS:* DKS means actual driving of DKSs. It includes a request scheduler and interruption handler for each actual device.

All of these control processes are executed by the CP. Therefore, individual boxes in Figure 7 will be implemented as independent processes of the CP. Each packet distributed to the processors is executed repeatedly until the retrieval request is satisfied.

### 4.2 Multiport Page-Memory Management

Relations in the knowledge base are stored in the secondary memory devices. They are staged to the MPPM when they are necessary to UEs for data retrieval.

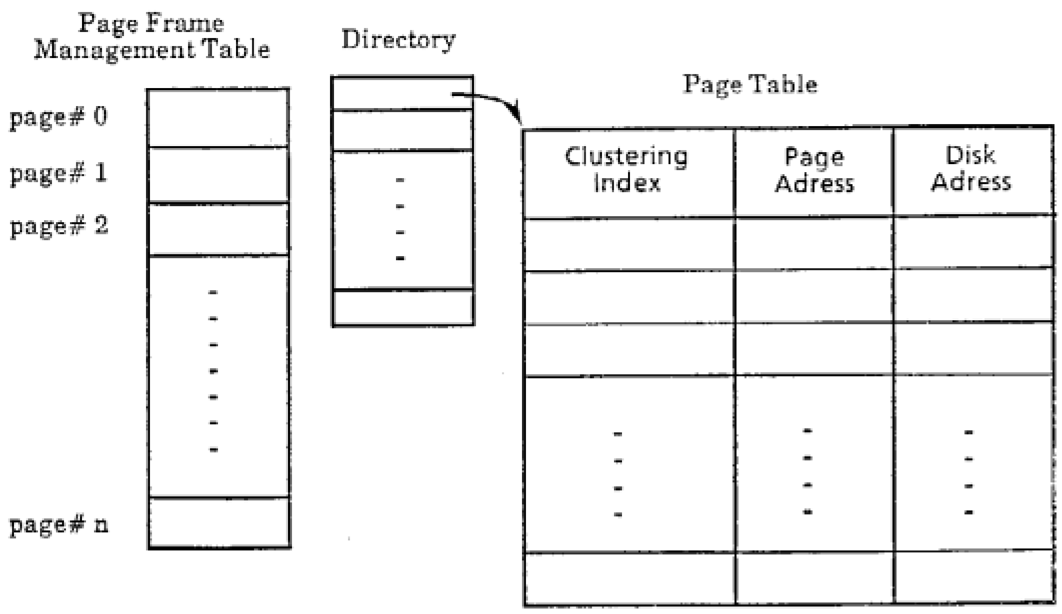


Figure 8. Memory Management Tables

Conventional virtual memory management schemes are usually applied to MPPM systems. However, since the MPPM stores only relation-type data, it is more efficient to treat relations directly in MPPM management. In addition, because the access unit for the MPPM is the page, high-efficiency buffer management is obtained by staging/destaging data from the knowledge base in page-sized units, rather than handling staging/destaging in relation-sized units. Managing relations in page-sized units also enables indexing every page by its contents.

From now on, the logical page of the MPPM is refered to as the *MPPM page frame*, and page-sized units grouping relations are refered to as simply *relation pages*.

The *page table* and the *page frame management table* in Figure 8 are used in MPPM management. These tables have the following objectives.

*Page Table:* Created for each relation to manage the relation memory status such as being staged or destaged. Page tables also keep indexing data for every relation page and the page address in the MPPM and the disk where the relation page is stored.

*Page Frame Management Table:* Created only once in the system. Individual entries of this table are created for each page frame of the MPPM, and the table is used to manage the page frame utilization status (either free or allocated to a page).

These tables are used to manage the MPPM page frames in the following way.

- The data stored in the knowledge base has a specific disk address for every page.
- When pages are staged to the MPPM, the address of the allocated page frames are set in the page table corresponding to the relation. At the same time, the page frame management table entry corresponding to the page being used is altered to indicate that it is in use.
- When the page frame is no longer needed by the MPPM, the corresponding page table entry in the page frame management table is rewritten to show that it is free.

A page table is generated for each relation and management of these page tables is performed by the memory manager. This means that the knowledge base management software does not need to be aware of the memory status of individual relations and can directly handle relational operations. Individual relations are stored in tuples, and items in a key attribute are assigned sequentially, enabling pagewise clustering in the page tables, as indicated in Figure 8. This clustering data can be used to reduce the object relation pages and enable staging of only essential pages.

## 5 Summary

This paper discribed the architecture and control techniques for a knowledge base machine using multiport page-memory and unification engines. In knowledge base machines where large volumes of data must be processed, the combination of multiport page-memory and dedicated processors appears to be a highly efficient approach to the elimination of the data processing bottleneck and enabling high-speed data processing on a data stream. Utilization of parallel control techniques in the event-driven manner reduces overhead in processor allocating procedures and inter-processor communication. It is also possible to execute retrievals in parallel efficiently.

Future research topics will include further investigation of the control technique proposed and quantitative measurement of granularity for parallel processing using hardware and software simulators. We must resolve the problem of so-called relation fragmentation[10] for efficient use of multiport page-memory and efficient parallel retrieval.

ACKNOWLEDGEMENTS

The authors thanks Mr. H. Sakai of TOSHIBA Corporation and members of ICOT's KBM (Knowledge Base Machine) working group for many useful discussions.

## References

[1] Boral, H., and DeWitt, D.J., "Database Machines: An Idea Whose Time has Past ? A Critique of the Future of Database Machines" *Database Macchines. H.O. Leilich and M. Missikoff (eds.) (Springer-Verlag. Berlin*, 1983, pp 166-187

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

[3] Yokota, H., Monoi, H., Morita, Y., and Itoh, H. "Construction of Multiport Page-Memory" *ICOT Technical Memo* No. TM-132 .

[4] Yokota, H., and Itoh, H. " A Model and Architecture for a Relational Knowledge Base" *ICOT Technical Report* No. TR-144, to appear in The 13th International Symposium on Computer Architecture, November 1985.

[5] Sabbatel, G.B., Dang, W., Ianeselli, J.C., and Nguyen, G.T. "Unification for a Prolog Data Base Machine" *Proceedings of the Second International Logic Programming Conference*, pp207-217july 1984

[6] Yokota, H., and Itoh, H. "Retrieval By Unification on Knowledge Base" *ICOT Technical Memo* No. TM-132.

[7] Morita, Y., Yokota, H., Nishida K., and Itoh, H. "Processing Method for Retrieval by Unification Operation on Relational Knowledge Base" submitted to the 12th International Conference on VLDB August 1986.

[8] Kakuta, T., Miyazaki, N., Shibayama, S., Yokota, H., and Murakami, K. " The Design and Implementation of Relational Database Machine Delta" *Proceedings of the International Workshop on Database Machines'85*, March 1985.

[9] DeWitt, D.J., "DIRECT - A multiprocessor organization for supporting relational database management systems" *IEEE Trans. comput.* C-28, 6(June 1979), pp.395-406

[10] DeWitt, D.J., "Query Execution in DIRECT" *In proceedings of ACM-SIGMOD 1979 International Conference on Management of Data*,(May 1979), pp.13-22

[11] Bancilhon, F., Richard, P., and Scholl, M., "The relational database machine VERSO" *6th Workshop on Computer Architecture for non numeric processing*, june 1981

[12] Gardarin, G., "An introduction to SABRE multimicroprocessor data base machine" *6th Workshop on Computer Architecture for non numeric processing*, june 1981