TM-0121

Development of Delta as a First Step to a Knowledge Base Machine

by
H. Sakai, K. Iwata, S. Shibayama,
(Toshiba Corp.)
M. Abe and H. Ito

July, 1985

Development of Delta as a First Step to a Knowledge Base Machine

Hiroshi Sakai Kazuhide Iwata
Shigeki Shibayama

Toshiba R & D Center
Kawasaki Japan

Masaaki Abe
Hidenori Itoh

ICOT Research Center
Tokyo Japan

ABSTRACT

Delta, a relational database machine is under development at ICOT (Institute for New Generation Computer Technology) to study knowledge base machines (KBMs). This paper focuses on its architecture especially on its specialized processor, RDBE (relational database engine) and also presents our approach to a knowledge base machine.

1 INTRODUCTION

Development of a knowledge base machine (KBM) is one of ICOTS's major themes. By KBM, we mean a backend machine to inference machines, which is able to store and manipulate a large amount and variety of knowledge.

One may regard knowledge information processing in general as a search problem. In this sense, database machines have the advantage of finding all items which match the search condition in a primitive but huge search space, while inference machines have the advantage of finding an arbitrary item in a complicated search space using heuristics in order to avoid combinatorial explosion. The authors think a KBM must have both mechanisms in an integrated manner, since most actual problems seem to need both brute-force search and heuristic search. Since other research groups of ICOT developed inference machines PSI [Taki 84], PIM-R [Onai 84], and PIM-D [Ito 84], the KBM group first focused on the database mechanism and developed Delta [Shibayama 84], a relational database machine. We chose the relational model among others because it seemed to provide a general storage of Prolog facts [Codd 70] [Gallaire 78].

An efficient relational database machine with an intimate interface to logic programming was the prime development objective of Delta. We chose (1) a functionally-distributed architecture, (2) relational database engines (RDBEs) to perform relational database operations, and (3) a large capacity hierarchical memory system.

The current status of the KBM group activities is as follows. Delta is now available to the inference machines and an effort to make the PSI accessible to Delta is being continued. To evaluate Prolog programs on a relational database system, Yokota proposed some methods [Yokota 84] [Yokota 85]. In order to develop a KBM, we have started the discussion about its functions and architecture.

This paper focuses on the development of Delta. In section 2, the overview of Delta and an exemple of query processing are presented. In section 3, the RDBE is described in detail. In section 4, a preliminary performance of some primitive functions is presented. In section 5, current problems of Delta and our efforts to solve them are discussed. In section 6, we will show our informal approach to a KBM.

2 OVERVIEW OF DELTA

2.1 System Configuration

Delta consists of five different kinds of components so that queries from inference machines may be efficiently processed on specialized components. Its architecture is shown in Figure 1.

The components of Delta are as follows:

(1) An interface processor (IP), which serves as a front-end to inference machines via a local area network (LAN).
(2) A control processor (CP), which provides database management functions, such as query language analysis, concurrency control, and dictionary/directory management.
(3) Relational database engines (RDBEs), which are the key components for processing relational database operations.
(4) A maintenance processor (MP), which has an operator console and provides functions for reliability and serviceability.
(5) A hierarchical memory (HM), which provides other units with a common storage.

The rest of this section presents an overview of Delta. Since the Delta system can be easily understood from its memory system, the HM unit is described first.

2.2 Hierarchical Memory Unit

The HM provides a common storage to other units of Delta. It is implemented using a conventional mainframe to achieve large storage and high speed data transfer. It has 128 Mbytes of main memory, 20 Gbytes of secondary memory and four magnetic tape drives. The HM is connected to the IP, CP and MP via block-multiplexer channels (one channel for each unit). As for the RDBEs, the HM has two channels for each so that data may be transferred from HM to RDBE and vice versa simultaneously.

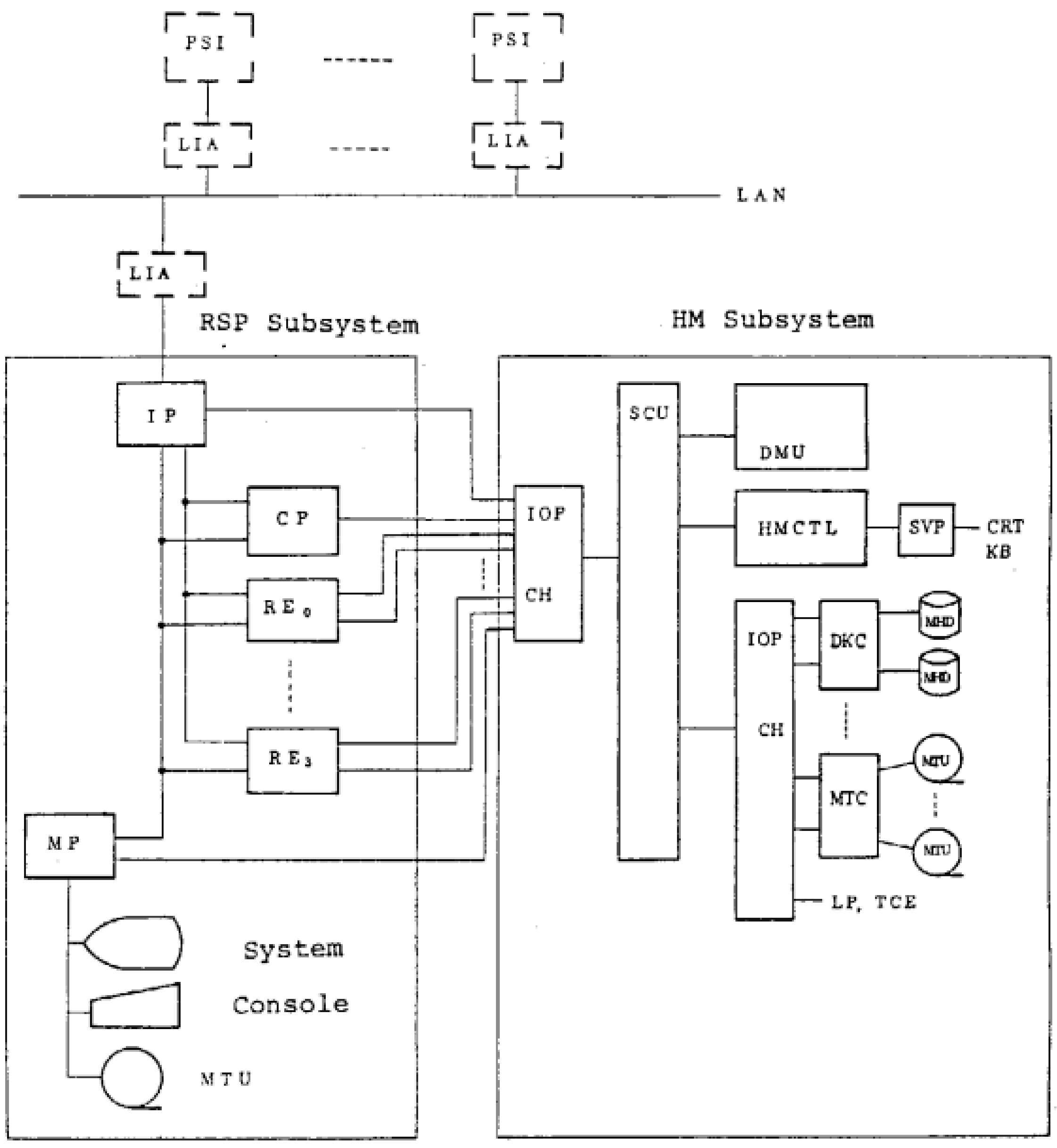
Units other than HM send commands to HM in order to access data in it. That is, the HM may well be regarded as a fast secondary storage. The storage of the HM can be classified into three types as illustrated in Figure 2.

The first type is called Attribute Dataset. It stores ordinary relations. Each relation is split into attributes and each attribute is stored in the secondary storage with an index. The HM software manages it using VSAM (virtual storage access method) and the page size is 4 Kbytes. As for the internal schema, we will discuss its detail later.

This area cannot be accessed directly from other units. In order to access an attribute of a relation, they send a command to make working data from the attribute. When a command has some conditions, the HM selects the data referring to the index as specified by the conditions.

The second type is called RSP Memory. It stores data copied from a relation, data generated by a RDBE and data sent from a host machine via the IP. It consists of 32 Kbyte pages of main memory. A set of pages is assigned to each context of data. For a large amount of data, the HM software stores the overflow portion in the secondary storage without using the virtual storage function of the operating system.

The third type is called RSP Dataset. It stores the directory of Delta. It is segmented into pages of 512 bytes and the HM software manages it using VSAM. The CP manages the contents of each page and the HM is responsible for logging modified pages.



PSI	: Personal Sequential Inference Machine			
LIA	: LAN Interface Adapter			
RSP	: RDBM Supervisory and Processing Subsystem			
IP	: Interface Processor	CP	:	Control Processor
RDBE	: Relational Database Engine	MP	:	Maintenance Processor
HM	: Hierarchical Memory Subsystem			
HMCTL	: HM Controller	DMU	:	Database Memory Unit
IOP	: I/O Processor	SCU	:	Storage Control Unit
MHD	: Moving Head Disk	DKC	:	Disk Controller
MTC	: MT Controller			

Figure 1 Delta configuration

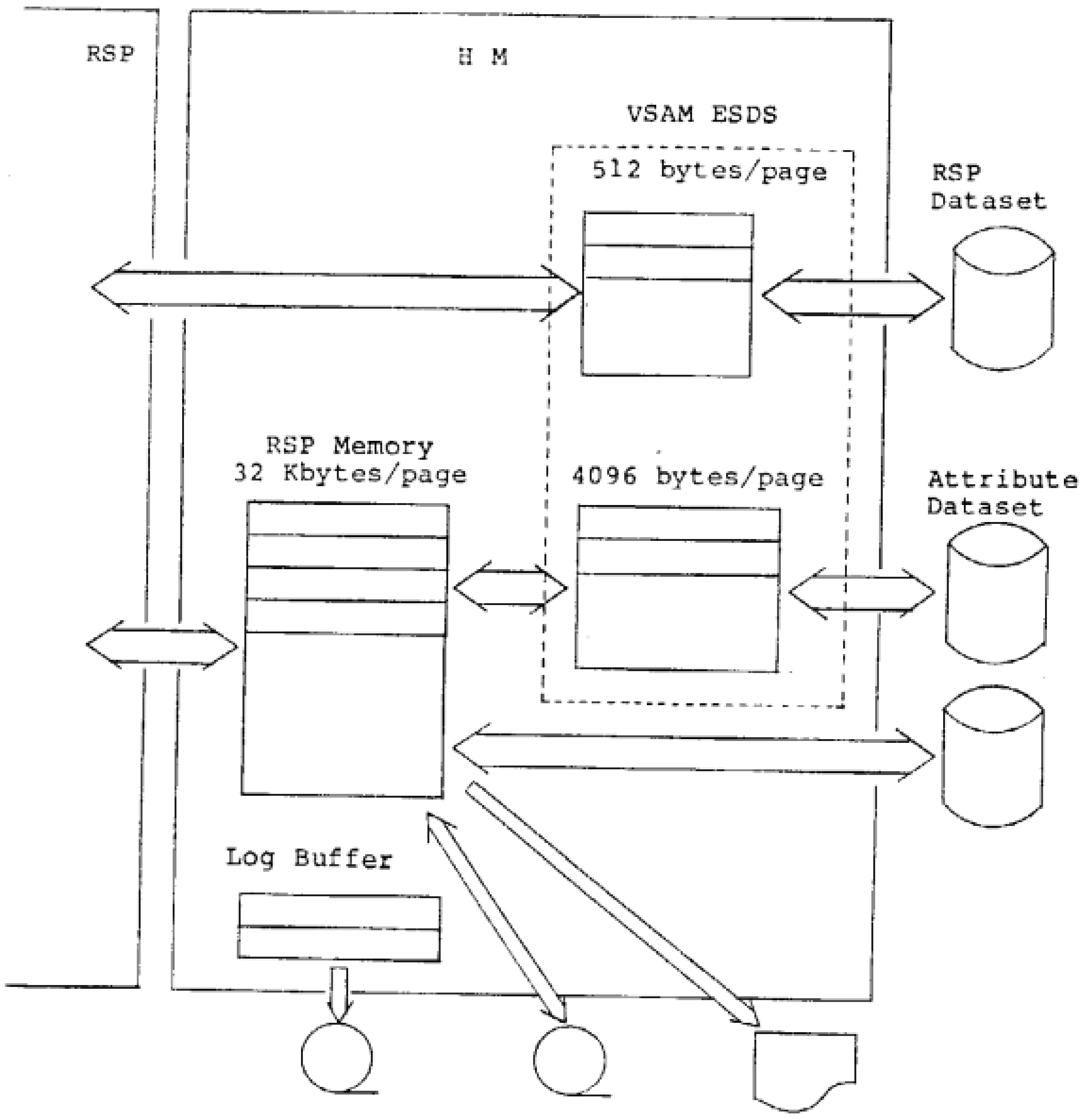


Figure 2 Storage management of the HM

2.3 Internal Schema

Conventional database management systems store a relation as a file in which a tuple is treated as a record and an attribute as a field. Indexing and hashing techniques are applied to rapidly obtain tuples satisfying specific criteria. These methods are useful when the user knows how to use the relation. A DBMS has to scan the entire relation if an indexed or hashed attribute cannot be used as an access path for a given query.

We expect Delta to have unconventional access characteristics because Delta is planned to be used as external storage of Prolog facts for inference machines. Access to the database stored in Delta is predicted to have the following characteristics, based on the usage of Prolog programs:

(1) There are only a few attributes in most relations.
(2) The attributes used as conditions are unpredictable.
(3) The frequency of access to each tuple is relatively uniform.

Delta adopts an attribute-wise schema to efficiently process these kinds of requests. Instead of storing all the attributes of a tuple together, a relation is split into a collection of attributes and stores all occurrences of each attribute together. A TID (tuple identifier) is attached to each attribute value to identify the tuple it belongs to. A two-level indexing method is used for clustering as illustrated in Figure 3.

The merits of the attribute-wise schema are as follows:
(1) Delta can avoid operations for attributes unnecessary for a given request.
(2) Attributes are treated uniformly.

However, there are several demerits as well.
(1) Transformation between the tuple-wise format and attribute-wise format is necessary.
(2) Tuple identifiers occupy additional storage space.
(3) The number of internal commands among the units grows.

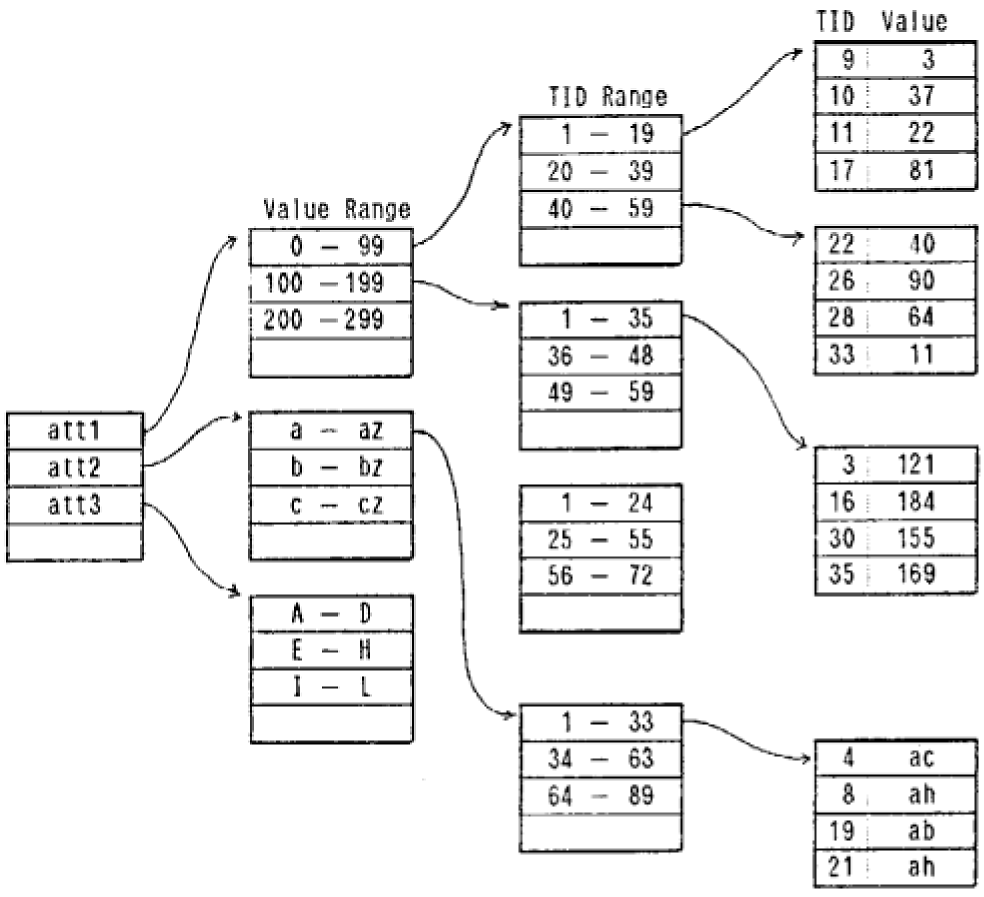


Figure 3 Two-level clustering method of Delta

Command name	C o m m e n t s
PASS	intratuple operation
JOIN	$=, \neq, <, \leq, >, \geq$, Cartesian product
RESTRICT	$=, \neq$, range
SORT	ascending/descending
AGGREGATE	aggregate operation
UNIQUE	eliminating duplicate tuples
UNION	set operation
INTERSECTION	set operation
DIFFERENCE	set operation
EQUAL	equality test between relations
CONTAIN	inclusion test between relations
COMPARE	compare attributes of each tuple
ZONE-SORT	for clustering
DELETE	for updating

Figure 4   List of RDBE commands

2.4 Relational Database Engine

The relational database engine (RDBE) is a specialized processor to perform various operations on the working data in the HM. Whenever the RDBE performs an operation, data is transferred from the HM to the RDBE and from the RDBE to the HM through channels.

An alternative is to place the RDBE between the HM's main memory and its secondary storage, as in VERSO [Bancilhon 82]. This would reduce data transfer time and improve system throughput. However, we did not choose this alternative because it was difficult to modify the disk controller of the HM.

The RDBE offers various kinds of commands necessary for relational database processing. The list of commands are shown in Figure 4. The RDBE performs these commands using its hardware modules, the sorter and merger, and also using its general-purpose microprocessor. The sorter and merger are designed to perform intertuple commands, i.e. commands which require comparison between records. For ease of implementation, the comparison is limited between a contiguous field of a tuple and a contiguous field of another, i.e. typically an attribute or the entire tuple. Comparison between an attribute value of each tuple and a list of constant values, and comparison between two attribute values of each tuple are also performed by them. The rest of the commands are performed by the microprocessor itself or their combination. The above decision was made according to the frequency expectation of the commands and their processing time, and also the functional flexibility of the RDBE.

The RDBE takes the entire tuple, and not only the key field. An alternative is to make a copy of the key field of each tuple and process it, which would reduce the data transfer between the RDBE and HM, as well as the required RDBE's memory. We did not adopt this alternative because of the following reasons:

(1) The HM would have to process the original tuples according to the result of the RDBE's operation.

(2) Since the set operation requires comparison of the whole tuples, the RDBE must have enough memory either way.

Although Delta adopts the attribute-wise internal schema, there exist working records having several attributes as well, and the RDBE must process them. The internal representation of a record, in general, is illustrated in Figure 5. Each record in a relation has the same length (less than 4 Kbytes) and the same number of fields; corresponding fields over a relation have the same data type and length. A field usually has an extra area called a tag, which indicates whether the value is null. The data types are unsigned integer, signed integer, and single-precision floating point. The length of the first two types must be even and less than 4 Kbytes.

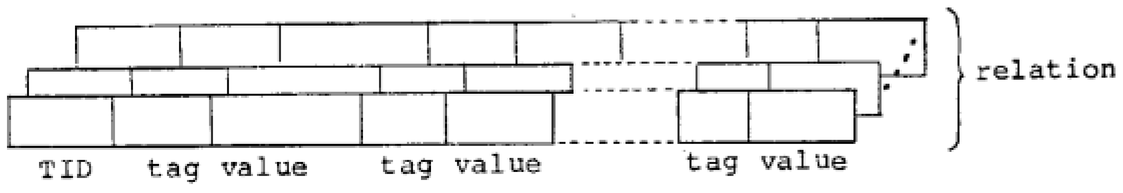


Figure 5 Internal representation of a record

2.4 Query Processing in Delta

This section shows how Delta processes a query using an example. Let us assume the example in Figure 6. A host machine wants to get the names and areas of nations having more than 1,000,000 people. The host machine has to send the sequence of Delta commands based on relational algebra. The IP receives the sequence and sends it to the CP. The CP translates the query into a sequence of internal commands as shown in the figure, each of which is then issued to an RDBE or HM to make them cooperatively perform the specified database operation. After its completion, the IP gets the result from the HM and sends it to the host.

```
Host Machine
     SELECT name, area
     FROM nations
     WHERE population > 1,000,000

               Sequence of Delta Commands
     Selection(nations.population, > 1,000,000, templ)
     Projection(templ, [name, area], intl)
     Get(intl)

Delta
     CP->HM  : PQB(nations.population, value > 1,000,000, bufl)
     CP->RDBE: RESTRICT(bufl, value > 1,000,000, buf2)
     CP->HM  : PQTB(nations.name, buf2, buf3)
     CP->RDBE: RESTRICT(buf3, TID  buf2.TID, buf4)
     CP->HM  : PQTB(nations.area, buf2, buf5)
     CP->RDBE: RESTRICT(buf5, TID  buf2.TID, buf6)
     CP->RDBE: SORT(buf4, TID, buf7)
     CP->RDBE: SORT(buf6, TID, buf8)
     CP->HM  : TTT([buf7, buf8], buf9)
                 /* from attribute-wise to tuple-wise format */
     IP->HM  : SPI(buf9)
```

Figure 6 Example of query processing in Delta

3 DETAILS OF THE RELATIONAL DATABASE ENGINE

3.1 Basic Idea

The basic idea is that a join operation is performed efficiently by sorting tuples of each relation according to their values and comparing tuples from the relations in a manner resembling a two-way merge operation. This idea is profitable since it can be applied not only to an equi-join operation but also to nonequal join operations and other relational database operations that take two relations. Althogh the idea has also been realized in other database machines, the relational database engine has the following advantages:

(1) The combination of the sorter and merger improves performance as in pipeline processing.
(2) The RDBE can process null values and duplicate values efficiently.
(3) The projection operation is performed during another operation.
(4) Parity check and sorting check mechanisms improve reliability.
(5) Data processing by the RDBE's microprocessor enhances its functional flexibility.

3.2 Configuration

The RDBE configuration is shown in Figure 7. It is designed for high-speed relational database processing by means of pipelined sorting and merge-like operation. The RDBE consists of the following components:

(1) A general-purpose microprocessor, which controls all the hardware modules to perform RDBE commands.
(2) Two HM adapters, which serve as interfaces between the RDBE and HM.
(3) The IN module, which transforms input data into an internal format suitable for the sorter and merger modules. Among these transformations are:
  * field ordering, which shifts a key field to the head of the tuple
  * data type transformation
  * generation of null value bit signals
(4) A sorter, which generates sorted tuples.
(5) A merger, which performs external sorting and relational database operations using a processing algorithm resembling a two-way merge operation.

In Figure 7, DT, PT, NL and DP stand for data lines, parity lines, a null line and a duplication line, respectively. The null line is used to denote that there is a tuple with a null value key on the data lines. The duplication line is used to denote that there is a tuple having the same key value as the subsequent one on the data lines. These modules are controlled to run simultaneously.

Data transfer is performed in the handshake mode between these modules. Each module is designed to achieve a data processing rate as high as the data transfer rate between the RDBE and HM. The main data path is from the HM adapter(IN) to the HM adapter(OUT) through the IN module, sorter, and merger.

If an RDBE operation takes two relations, as in a join operation, the operation is performed in the following way:

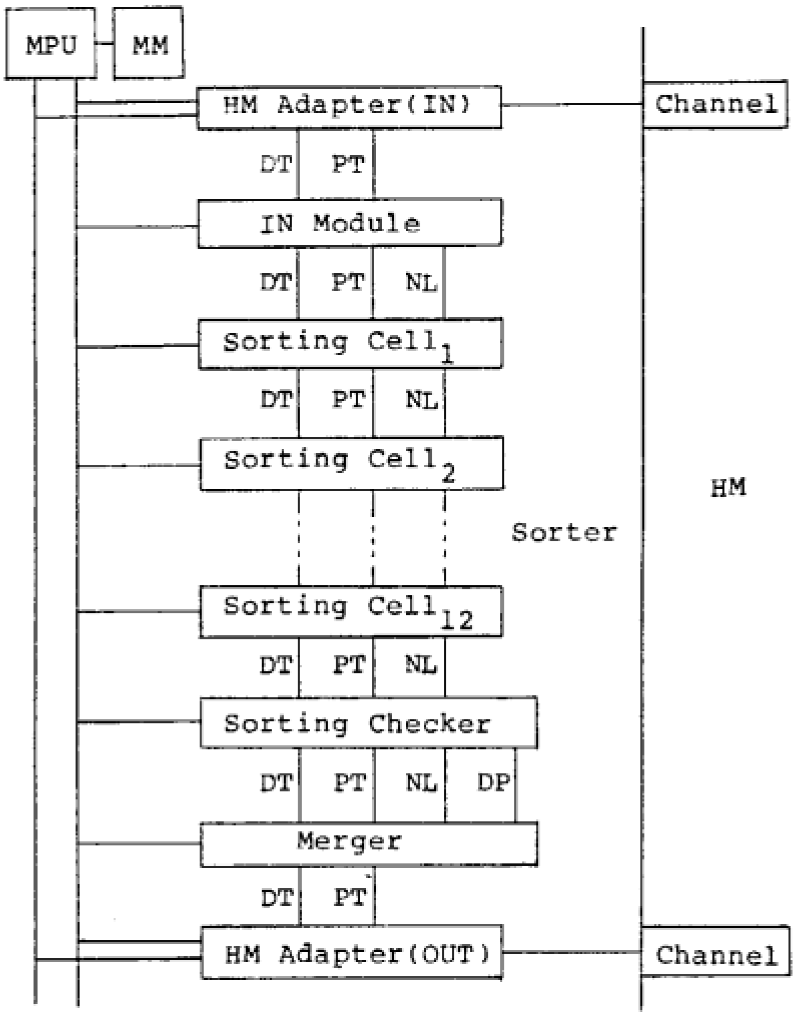


Figure 7 RDBE configuration

The tuples of the first relation, which was originally stored in the HM, pass through the HM adapter(IN), are modifyed by the IN module, sorted by the sorter, and are finally stored into a buffer of the merger. Then the tuples of the second relation pass through the HM adapter(IN), are modified by the IN module, sorted by the sorter, and are stored into another buffer of the merger. While storing the second tuples, the merger also compares these with the previously stored tuples, and generates the results. They are sent to the HM through the HM adapter (OUT).

If the microprocessor itself has to manipulate the data, the result from the merger is sent to its main memory via the HM adapter(OUT). After the microprocessor has finished the operation, the final result is sent to the HM via the HM adapter (OUT).

3.3 Sorter

In order to apply a sorter in the RDBE environment, the following conditions must be satisfied:

(1) It receives the original sequence of tuples from the IN module, and sends the sorted sequence to the merger.
(2) The data transfer rates both at its entrance and at its exit are equal to that between the RDBE and HM.
(3) The delay between the ending of the input data transfer and the beginning of the output data transfer is small.
(4) It is able to sort a small number of tuples at reasonable speed.
(5) It is able to process absolute values of standard binary notation up to 4094 bytes long, possibly with the null value signal on.

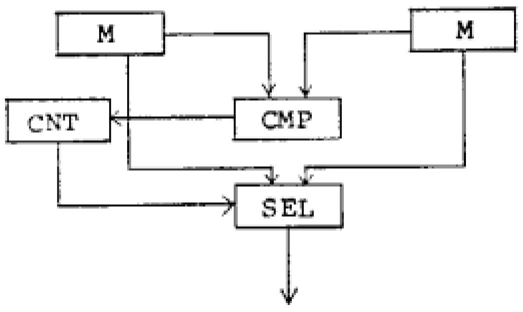
Various kinds of sorting algorithms have been studied [Knuth 73], and hardware sorters based on them have been proposed and implemented. Tanaka proposed and implemented a sorter based on the heap sort [Tanaka 80]. Although it satisfies the above four conditions, it is difficult to implement so that it satisfies the last.

Our sorter, based on two-way merge-sort, is similar to Todd's [Todd 78]. It is slightly inferior to Tanaka's for the third condition, but it satisfies the last condition. Our sorter has the following features:
(1) The sorter consists of a linear array of 12 processing elements, called the sorting cell, and one processing element, called the sorting checker; these arrange input data elements in a specified linear order (ascending or descending). Since the data bus consists of 16 lines, the unit size of data, word, is 2 bytes. The sorting operation is performed by pipeline processing.
(2) The sorter performs only the internal sort operation. The maximum number of tuples that the sorter is able to process is shown by the following expression.
min(2**N, [M/L])
Here, N is the number of sorting cells (currently 12), M is the memory size of the last sorting cell (currently 64 Kbytes), and L is the tuple length.
(3) The sorting cell has two operation modes: the sort mode and the pass mode. The former merges two sorted sequences of tuples into one. The latter does not merge, but transfers input data directly to the next cell. Let C be the number of tuples to be sorted, then $[\log_2(C-1)]$ of the sorting cells become the sort mode, and the others become the pass mode.
The time required is (2LC + N - 1)T sec, excluding the time which the sorting checker and the control program take, where T is the reciprocal of the data transfer rate (currently 3 Mbyte/sec). For example, 4096 tuples of 16 bytes are sorted in 43 milliseconds. Note that the processing time does not depend on the length of the key field.
(4) The sorter processes null values by recognizing the tag field and locates them at the last part of the sorted sequence.
(5) The sorter performs stable sort operations on equal values, i.e., it keeps the original relative order of the input sequence of tuples having the same values.
(6) The sorting checker compares the key field of each tuple with that of the next one, so that it checks the results to increase the reliability of the sorter. It also generates a

duplicate signal when the values are the same. Since it takes an additional time of LT, the time required is (2LS + L + N)T excluding the software overhead time.

Figure 8 is a block diagram of the sorting cell. It contains two memories, each with a first-in/first-out function (FIFO), a comparator and a control circuit.

The sorting cell operation for every two bytes consists of three cycles. They are memory read cycle, another memory read cycle, and compare-transfer cycle. In the first cycle, the word of the first sorted subsequence is read and stored into the register of the comparator. In the second cycle, the same operation for the other subsequence takes place. In the last cycle, the comparator compares them and the selector outputs the smaller or greater word to the (i+1)th cell, according to the ascending or descending mode. In this cycle, the word sent from the (i-1)th cell is stored into the memory. Each cycle takes 220 nsec, and the two-byte merge operation takes 660 nsec.



M : Memory
CNT : Controller
CMP : Comparator
SEL : Selector

Figure 8 Block diagram of a sorting cell

3.4 Merger

The merger is the central module of the RDBE, which performs relational algebra operations and other operations using a processing algorithm based on the two-way merge-sort operation. These are called merger commands and are classified into the five types of operations listed in Figure 9. They are characterized by their ability to process null values and duplicate values.

A block diagram of the merger is shown in Figure 10. The merger consists of an operation section and an output control section. The operation section contains a comparator, a control ROM table, and two 64 Kbyte memories (U-memory and L-memory) having a FIFO function. This section performs the following steps:

(1) Store two sorted streams from the sorter into the memories
(2) Read a tuple from each of two memories simultaneously and providing them to the comparator and the tuple memory in the next section
(3) Compare the keys of each tuple and detect output tuples satisfying the conditions of the command

PASS COMMAND	RESTRICT COMMAND
LOAD	REST-NULL
PASS-1(NOP)	REST-NONULL
PASS-2(UNQ-IN)	REST-EQ
SORT COMMAND	REST-NE
SORT-IN	REST-RANGE
SORT-EX	
UNQ-EX	
COMPARE COMMAND	JOIN COMMAND
COMP-ALL	JOIN-ALL
COMP-NULL	JOIN-NULL
COMP-NONULL	JOIN-NONULL
COMP-EQ	JOIN-EQ
COMP-NE	JOIN-NE
COMP-LT	JOIN-LT
COMP-GT	JOIN-GT
COMP-LE	JOIN-LE
COMP-GE	JOIN-GE

NOP	: No operation	UNQ-IN	: Unique-Internal
EX	: External	NONULL	: Not null

Figure 9 List of merger commands

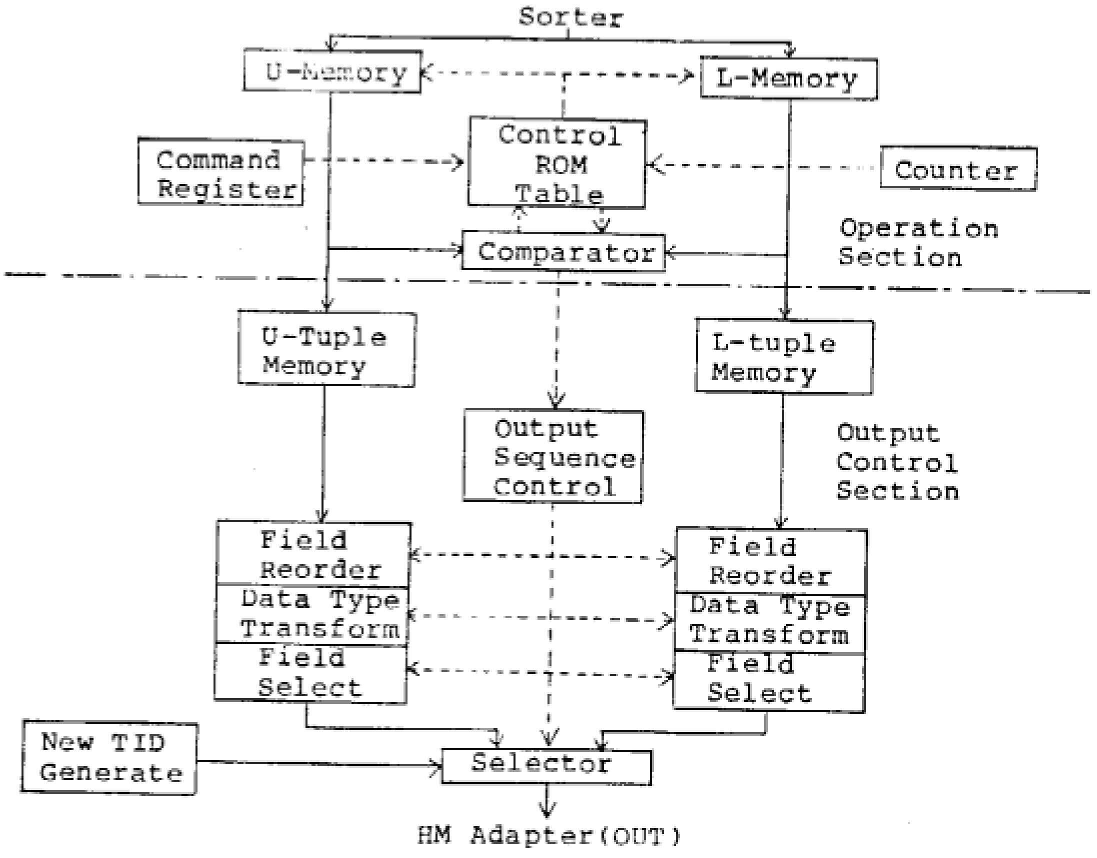


Figure 10 Block diagram of the merger

These functions are executed under the control of a 1-Kword * 10-bit ROM table. The address of the ROM table consists of a null signal, duplication signals, the comparison result flag and so on. The output of the ROM table consists of memory address control signals, tuple-selection signals used for the output control section, and an operation-end signal.

The output control section consists of two 16 Kbyte tuple memories, two field-ordering circuits, two field-selection circuits, two data-type-transformation circuits, a new TID (tuple-identifier) generator, a selector and an output sequence controller. This section performs the following functions under software control:

(1) Reorder the fields of an output tuple
(2) Select fields of an output tuple
(3) Recover the original notation of the key field
(4) Add a new TID to an output tuple.

Examples of these functions are shown in Figure 11. Figure 11(a) shows the reordering of the fields of an output tuple. A tuple(1) with five fields (A, B, C, D, E; B is a key field) is rotated to tuple(2) by the IN module, so that the key field is positioned at the head of the tuple, and tuple(2) is rearranged to the original tuple(3) by the merger. The selection of the fields of an output tuple is shown in Figure 11(b). In this figure, tuple(4) is projected to tuple(5) or tuple(6) by the assignment of the two pointers, p_1 and p_2. Figure 11(c) shows the addition of a new TID to an output tuple.

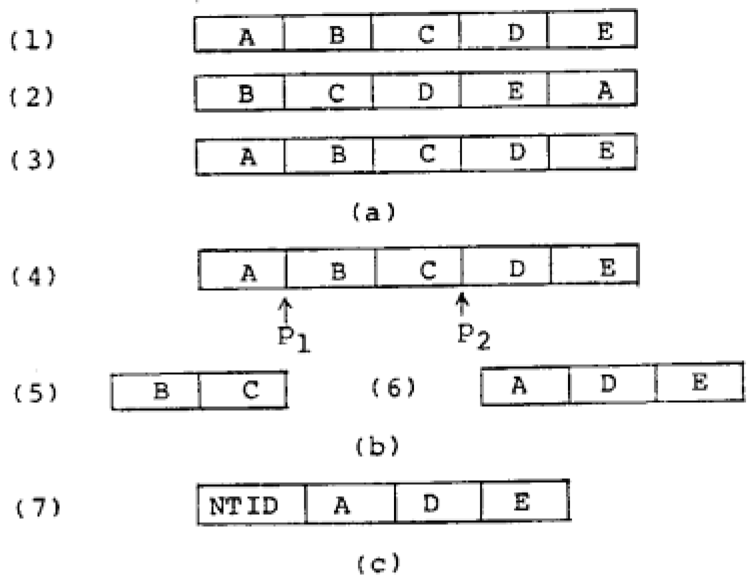


Figure 11 Functions of the output control section

An example of the JOIN-EQ operation is illustrated in Figure 12. JOIN-EQ command is typically used when an equi-join of two relations is performed. Figure 12(a) shows two input streams (S1 and S2) sorted in ascending key-order (A1 and B1); these are stored in the U- and L-memories, respectively. UADR and LADR provide sequence numbers, explaining the address control scheme for each memory.

Figure 12(b) shows the output tuples and Figure 12(c) illustrates the execution process. The processing algorithm of the JOIN-EQ command is as follows:

```
If A1 > B1 then LADR := LADR + 1;
if A1 < B1 then UADR := UADR + 1;
If A1 = B1 then
        output a matched tuple pair
        if the DP of A1 and the DP of B1 are on,
           then LADR := LADR + 1;
        if the DP of A1 is on and the DP of B1 is off,
           then UADR := UADR + 1;  LADR := LADR*
        if the DP of A1 is off and the DP of B1 is on,
           then LADR := LADR + 1;
        if the DP of A1 and the DP of B1 are off,
           then UADR := UADR + 1;  LADR := LADR + 1;
```

Here, DP stands for the duplication line and LADR* points to the first tuple of those which have the same values. Adopting this algorithm, the merger is able to perform the JOIN-EQ command on the attributes having duplicate values efficiently.

(a)

Stream S1
(U-Memory)

UADR	A1	A2
0	2	B
1	3	C
2	3	D
3	4	A

Stream S2
(L-Memory)

LADR	B1	B2
0	1	T
1	3	S
2	3	V
3	5	U

JOIN A1=B2

(b)

A1	A2	B1	B2
3	C	3	S
3	C	3	V
3	D	3	S
3	D	3	V

(c)

UADR	LADR	U-Tuple	L-Tuple	Result	Output
0	0	2 B	1 T	>	
0	1	2 B	3 S	<	
1	1	3 C	3 S	=	3 C 3 S
1	2	3 C	3 V	=	3 C 3 V
2	1	3 D	3 S	=	3 D 3 S
2	2	3 D	3 V	=	3 D 3 V
3	3	4 A	5 U	<	
END	3				

Figure 12 Example of processing JOIN-EQ command

The operation of the merger is divided into three cycles. These are the four-byte read cycle, compare-transfer cycle and the ROM table read cycle. Each cycle takes 220 ns and is synchronized with the sorter.

3.5 Data Processing by the Microprocessor

Since the operations performed by the sorter and merger are limited to the intertuple comparison concerning one field (typically one attribute) for each relation, the other operations must be performed by its microprocessor. These are as follows:

(1) Selection under complex conditions
(2) Arithmetic operations
(3) Aggregate operations

In order to improve the performance, the RDBE has a compiler which generates the native machine instructions into the main memory. The instructions are executed on the tuples generated by the merger and stored in its main memory by the HM adapter(OUT). The final result is sent to the HM through the HM adapter(OUT).

Since data processing using its microprocessor can be overlapped with the sorter and merger operation, a combined operation is able to be performed in one shot. The following is an exemplified query;

```
SELECT *
FROM A, B
WHERE a1 = b1 AND a2 > b2
```

a join operation with a conjunctive condition. The RDBE is able to perform the join operation in one shot; the equal condition, using the sorter and merger, and the other, using its microprocessor.

3.6 Control Mechanisms

The RDBE's control mechanisms of the modules, in order to perform a RDBE command are described in this section. Since the sorter and merger have limited capacity; i.e. the maximum amount of data which the sorter and merger are able to process in one scan, the microprocessor controls the modules in a different way. This depends on the category of the RDBE command and the amount of data. They are as follows:

(1) Unary intratuple operation;
Arithmetic operations and selection are involved in this category. When the amount of data is so huge that the modules (including the capacity of the main memory) is not able to process it in one shot, the microprocessor controls the modules for each nonintersecting portion (called a substream) of the original data repeatedly.

(2) Sort type operation;
When the amount of data is small enough for the sorter to process in one shot, the microprocessor indicates the sorter to sort it and merger to pass it.
When the data is not greater than twice the sorter's capacity, the microprocessor first indicates the sorter to sort one half of it and the merger to store it into its U-memory. Then it indicates the sorter to sort the rest and the merger to merge them.

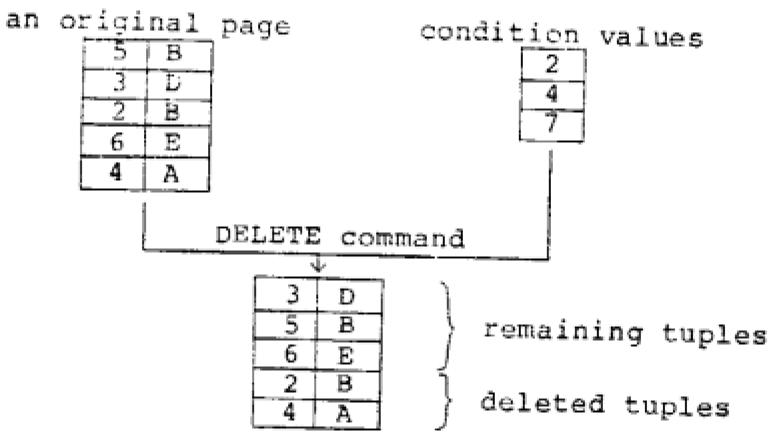


Figure 13 Example of the DELETE command

Otherwise, the microprocessor first controls the modules as in the second case, to generate two partially sorted sequences. Then the microprocessor indicates the merger to merge them repeatedly. Since each step is based on a two-way merge operation, it becomes inefficient when the amount of data becomes large, in comparison with the multi-way merger [Dohi 83].

(3) Binary operation (type 1)

In a join-like operation, the microprocessor controls the modules to perform the operation for each combination of the substreams of the relations repeatedly. An alternative is to sort each tuple first and to process the JOIN command on them. However, this is not adopted for two reasons; (1) it takes more time when the amount of data is not eight times greater than the sorter's capacity, and (2) It does not work well when a large number of tuples have the same value.

(4) Binary operation (type 2)

In a difference-like operation, the microprocessor controls the modules as follows. Let Ra and Rb be the original relations, and the operation be to get (Ra - Rb). It first indicates the sorter to sort the first substream of Rb (say Rb1) and the merger to store it into its U-memory. Then it indicates the sorter to sort each substream of Ra and the merger to perform the RESRICT-NE operation between each substream of Ra and Rb1 repeatedly.

Since the operations described above generates a temporary result of (Ra - Rb1), The microprocessor repeats the same operations to generate the final result.

Besides the control mechanisms described above, the microprocessor controls the merger to perform different operations on the same data. This is useful in the DELETE command. In the DELETE operation, the RDBE deletes those tuples the key of which match any of the condition values. First, the microprocessor controls the sorter and merger to store the condition values into the merger's U-memory. Then, for each page, It controls the sorter and merger in two steps; (1) store all tuples into the merger's L-memory and at the same time, output tuples which unmatch any of the condition values, (2) output tuples in the L-memory which match one of the condition values, Figure 13 illustrates an example. It helps the HM check whether the contents of each page are modified or not.

3.7 Increasing Reliability

The RDBE has the following features to gain reliability. A parity check mechanism and the sorting checker detect hardware errors with very little increase in processing time.

When an error occurs, the microprocessor resets the modules and then controls the HM adapters to inform HM to retry the data transfer. The HM only has to treat it as an ordinary I/O error.

During the power-up sequence, the microprocessor performs RDBE operations on certain test data. The test data in the main memory is provided to the IN module through the HM adapter(IN). The result is stored in the main memory via the HM adapter(OUT) and is checked by the microprocessor.

4 Performance of Primitive Operations

This section presents the performance of some primitive operations of the HM and RDBE. We have a plan for thorough performance evaluation of Delta after finishing its refinement task, which will continue till the end of this year.

As for the HM, performance should be evaluated for each type of storage, since its storage is classified into three types. However, we have only obtained till now the performance of the RSP Memory which stores working data and behave as a fast common secondary storage. The access time and the transfer rate are as follows:

Access Time	8 msec (Typ.)
Transfer Rate (to IP/CP/MP)	1 Mbyte/sec
Transfer Rate (to RDBE)	3 Mbyte/sec

As for the RDBE, we have obtained the performance of its representative commands JOIN (equi-join) and SORT. We will first discuss the theoretical performance estimation of the JOIN operation. Then we will compare it with actual performance.

Since every RDBE module, except the merger, proceeds in a deterministic way, the time required to perform an RDBE operation can be estimated. The activity of the merger, however, depends on the distribution of values in the input tuples, so we present the worst-case estimation. The following list sums up the parameters necessary to estimate performance:

N	: number of sorting cells
M	: merger's U- and L-memory capacity
Cn	: tuple count of the n-th relation
Ln	: tuple length of the n-th relation
Fn	: number of substreams; equal to [(Cn-1)/min(2N,M/Ln]+1
Snj	: j-th substream of the n-th relation
R	: tuple count of the result
T	: time required to transfer one byte

In a join operation, the microprocessor controls the modules in the following way:

```
send a request to HM through the HM adapter(IN)
while the first stream is not exhausted
begin
  get a substream from HM
  modify it in the IN module
  sort it in the sorter                                          A
  store it into the U-buffer of the merger
  send a request to HM through the HM adapter(IN)
  while the second stream is not exhausted
  begin
    get a substream from HM
    modify it in the IN module
    sort it in the sorter                                        B
    compare it with the previous one to generate the result
  end
end
```

Here, the statements enclosed are executed in parallel. Figure 14 shows the time chart of the activities of the modules while executing section A. The total time taken in this section is calculated as follows:

The sorter, including the sorting checker, takes
$(2S_{1i}L_1 + N - 1 + L)T$
for the i-th substream of the first relation. The IN module takes an extra time of L_1T and the merger takes an extra time of T. The total time is equal to the following expression.

$$(2S_{1i}L_1 + N + 2L_1) = 2C_1L_1 + F_1(N + 2L_1)$$

In section B, the merger takes an extra time of
$((S_{1i} + S_{2j})\max(L_1+L_2) - S_{2j}L_2)T$
for scanning the i-th substream of the first relation and the j-th substream of the second relation. In addition, generating a record as a result takes $(L_1 + L_2)T$.

After all, the JOIN command takes the following time:

$$(2C_1L_1 + F_1(N + 2L_1))T + F_1(C_2L_2 + (C_1 + C_2)\max(L_1 + L_2) + F_2(N + 2L_2))T + R(L_1 + L_2)T$$

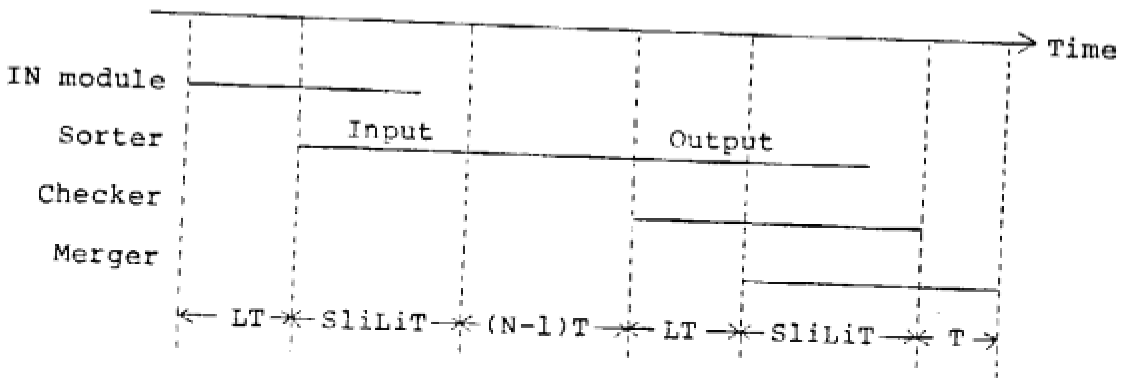


Figure 14 Time table of the activities of the RDBE modules

When C1 and C2 are equal to 4096, and L1 and L2 are equal to 16 bytes, and the resulting tuples are small, the RDBE takes approximately 128 milliseconds. Note that the processing time does not depend on the key size.

Figure 15 shows the theoretical performance estimation and the actual values for JOIN and SORT commands. Here, record size is always 16 bytes. The result shows that the RDBE, cooperating with the HM, achieves the expected performance.

(a) JOIN command

IN1	IN2	OUT	T	A
1	1	1		31
4096	4096	4096	107	150
8192	4096	4096	215	279
16384	4096	4096	301	377
32768	4096	4096	559	660
65536	4096	4096	1075	1345
131072	4096	4096	2107	2366
262144	4096	4096	4171	4793
524288	4096	4096	8299	10856

(b) SORT command

IN	OUT	T	A
1	1		21
4096	4096	43	64
8192	8192	107	136
16384	16384	349	445
32768	32768	961	1299
65536	65536	2446	3139
131072	131072	5940	7356
262144	262144	13978	17127
524288	524288	32148	42292

IN	:	number of records in the source relation
OUT	:	number of generated records
T	:	theoretically estimated processing time (msec)
A	:	actual processing time (msec)

Figure 15 Performance of the RDBE

5 PROBLEMS OF DELTA

Several problems concerning the performance of the Delta system have been disclosed, although we have not made thorough evaluation. We are refining Delta, especially concerning the internal commands among units and the basic software in order to improve efficiency.

Several crucial problems are derived from the configuration of Delta, especially the separation of the CP and HM. We think that they should be integrated into a single unit and the integrated unit should use the RDBEs as I/O devices because of the following reasons:

(1) The number of internal commands would be reduced approximately by 80%. The integration of the CP and HM would directly make the internal commands between them unnecessary. It would reduce the number approximately by 40%. If the integrated unit would use the RDBEs as I/O devices, then the internal commands between the RDBE and HM would become unnecessary. It would reduce the number by additional 40%.

(2) Since a query compilation, which the CP does now, requires several pages of the directory to be read and modified, the directory should be accessed by the CP as its main memory, and not as its secondary memory.

(3) The scheduling of the query execution, which the CP does now, should be related to the considerations of the HM resources. In order to realize it, the integration of the CP and HM is necessary.

(4) Currently, the RDBE receives an internal command (e.g. JOIN) from the CP and then the RDBE sends several internal commands which require data transfers to the HM. When the RDBE executes an external sort operation on a large amount of data, it is a significant problem on performance that the HM does not know the operation of the RDBE and it cannot predict the RDBE's requirement.

(5) Currently, staging a relation from the HM's moving head disk to its main memory occurs by an internal command sent from the CP. After its completion, the RDBE can perform some relational database operation on it. So the staging phase and the execution phase are not overlapped.

Other problems are derived from the attribute-wise schema of the relations. It needs several times more internal commands than in the case a tuple-wise schema would be adopted. The conversion from attribute-wise data to tuple-wise ones and vice versa requires much resources when the data size is large. The two level clustering, in which each attribute is clustered by the value first and then is clustered by the TID, is not easy to maintain and results in dull selectivity.

Figure 16 shows an example of a preferable internal schema for Delta. The attributes having a fixed length should be gathered together and one of them should be treated as a primary key. The other attributes (e.g. image data or explanations of a dictionary) should be stored separately.

Some more problems are derived from the operating system of the IP and CP. Since they have a modified version of a time-sharing operating system, they are not able to respond internal commands quickly because of the following reasons:

(1) Although Delta is required to support a multi-transaction environment, its operating system need not necessarily support a multi-task environment. Such a powerful operating system consumes non-negligible time for context switching and concurrency control. A monitor of an event-driven type seems best for the IP and CP.

(2) The control program of a special-purpose machine, such as Delta, should be allocated at a certain address during its start-up sequence. An ordinary time-sharing operating system, however, does not allow the static memory allocation for a user program.

(3) Since specific communication requires certain protocol handling, the device handler has to support it in order to achieve quick interrupt handling. Most general-purpose operating systems, however, have only physical-level handlers and it is a user program that must handle the protocol.

TID	name	area

TID	characteristics

TID	map

Figure 16 Example of a preferable internal schema

We plan to improve the efficiency of these units by exchanging the operating system to a realtime monitor of our own. Several times improvement of the systems throughput is expected and also the whole software of the IP and CP will be reduced much in size.

6 Toward a Knowledge Base Machine

In this section, we will present our ideas concerning the KBM. Since we have just started the discussion and various approaches are being proposed and discussed, the contents of this section are still informal and indefinite.

As for the functions of a KBM, we think that operations concerning knowledge (i.e. inference and pattern matching) should be added in order to realize both an intelligent database management system (DBMS) and knowledge manipulation.

We think a conventional DBMS has the following problems:

(1) In a conventional retrieval system using keywords as a search condition, the keywords must be originally registerd by the users.

(2) In a query system with a natural language interface, a conventional DBMS cannot make an incomplete query accurate for the lack of the common sense and knowledge about the contents of the database.

(3) The more a database grows in size, the more important its integrity check becomes. Generally the user must direct the condition timely.

(4) In a conventional database system, the user must choose a suitable file organization for each relation in order to achieve high performance. It needs skill and the choice becomes inadequate when the usage or the size varies.

(5) A conventional DBMS can process a query only when the database has the immediate answer.

We think that these problems should be investigated in the course of our KBM research.

Delta can handle knowledge of tabular form, which, in Prolog, corresponds to facts with constants as arguments. One may well regard a query processing on Delta as a breadth-first evaluation of a Prolog goal. It is a natural idea that a KBM should handle more generalized knowledge than Delta. The following are our fundamental ideas:

(1) Conventional database can only process knowledge as data. That is, it is impossible that the database recognizes the contents of knowledge or executes specific operations (e.g. matching and inference).

(2) In Prolog, knowledge is represented as facts and rules. It is well known that there can exist many facts with the same predicate name. However, it is said that the number of rules with the same predicate name is almost always less than ten. So the Delta architecture which is designed to handle only facts seems insufficient to a KBM. We have an idea of developing a specialized processor called inference engine to perform a breadth-first evaluation of Prolog programs.

(3) We think a KBM should also treat several kinds of knowledge representations other than Horn clauses, (e.g. semantic nets, frames, relations) directly for the sake of efficiency and

ease of understanding. Specialized mechanisms will be required for each knowledge representation and for the conversion between them. Delta-like RDBM should be regarded as a special component for tabular knowledge.

(4) We expect that a KBM must have considerable size of main memory. While the development of semiconductor technology will reduce the cost of memory, the bottle-neck between the processor and the memory will remain. According to such considerations, we have an idea of an intelligent memory system which stores knowledge. Its detail will be presented at a succeeding paper.

7 Conclusions

Delta is a relational database machine developed at ICOT research center as a first step to a KBM. In order to achieve efficiency, we adopted (1) a functionally-distributed architecture, (2) relational database engines (RDBEs) to perform relational database operations, and (3) a large capacity hierarchical memory system. Although the RDBEs, cooperating with the hierarchical memory system, achieved expected performance, the Delta system as a wohle had several problems and we are trying to solve them now.

As for a real KBM, we plan to develop a new machine which can treat several kinds of knowledge representations. The Delta is regarded as its compnent for tabular knowledge.

ACKNOWLEDGMENTS

The present research effort is part of a major research and development project of the fifth generation computer, conducted under a program set up by the Ministry of International Trade and Industry.

We would like to express our sincere appreciation to the members of the ICOT KBM group for their valuable discussions, and to Dr. K. Mori, director of the TOSHIBA Information Systems Laboratory, who provided the opportunity to conduct the present research. Special thanks to Mr. F. Umibe for polishing our English.

REFERENCES

[Bancilhon 82] Bancilhon, F., et al. VERSO: A Relational Back-End Data Base Machine, Proc. of Int'l Workshop on Database Machines, Aug., 1982.

[Boral 82] Boral, H., et al Implementation of the Database Machine DIRECT, IEEE Trans., Software Eng., Vol. SE-8, No.6, Nov., 1982.

[Codd 70] Codd, E.F. A Relational Model of Data for Large Shared Data Banks, CACM, Vol.13, No.6, June, 1970.

[Dohi 83] Dohi, Y., et al. Sorter Using PSC Linear Array, International Symposium on VLSI Technology, Systems and Applications, 1983, pp.255-259.

[Gallaire 78] Gallaire, H., and Minker, J. (eds) Logic and Data Bases, Plenum Press, 1978.

[Hell 1981] Hell, W. RDBM-A Relational Database Machine Architecture and Hardware Design, Proc. 6th Workshop on Comp. Arch. for Non-Numerical Processing, Hyeres, France, June, 1981.

[Ito 84] Ito, N., et al. Parallel Inference Machine Based on the Data Flow Model, Proc. of Int'l Workshop on High-Level Computer Architecture 84, 1984.

[Kakuta 85] Kakuta, T., et al. The Design and Implementation of Relational Database Machine Delta, Proc. IWDM'85, (1985)

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

[Onai 84] Onai, R., et al. Parallel Inference Machine Based on Reduction Mechanisms - Its Architecture and Software Simulation, ICOT Technical Report TR-077, 1984.

[Shibayama 84] Shibayama, S., et al. A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor, ICOT Technical Report, TR-055, 1984.

[Taki 84] Taki, K., et al. Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI), Proc. of International Conference on FGCS '84, Nov., 1984, pp.398-409.

[Tanaka 80] Tanaka, Y., et al. Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer, IFIP80, pp.427-432, 1980.

[Todd 78] Todd, S. Algorithm and Hardware for a Merge Sort Using Multiple Processors, IBM J. RES. DEVELOP., Vol.22, No.5, Sept., 1978.

[Yokota 84] Yokota, H., et al An Enhanced Inference Mechanisms for Generating Relational Algebra Queries, Proc. of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Waterloo, April, 1984, pp.229-238.

[Yokota 85] Yokota, H., et al Deductive Database System based on Unit Resolution, ICOT Technical Report, TR-123, 1985.