ICOT Technical Report: TR-053

A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor

TR-053

by
Shigeki Shibayama, Takeo Kakuta
Nobuyoshi Miyazaki, Haruo Yokota
and Kunio Murakami

March, 1984

©1984, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Technical Report of ICOT Research Center : TR - 053

A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor

Shigeki Shibayama, Takeo Kakuta
Nobuyoshi Miyazaki, Haruo Yokota
and Kunio Murakami

ICOT Research Center
Institute for New Generation Computer Technology

Mita Kokusai Bldg. 21F, 1-4-28 Mita,
Minato-ku, Tokyo, 108
Tel:03-456-3195, Telex:ICOT J32964

A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor

Shigeki SHIBAYAMA, Takeo KAKUTA, Nobuyoshi MIYAZAKI,

Haruo YOKOTA, and Kunio MURAKAMI

ICOT Research Center

Institute for New Generation Computer Technology,
Mita Kokusai Bldg. 21F, 1 - 4 - 28, Mita
Minato-ku, Tokyo 108

- Contents -

Abstract

1. Introduction

2. Delta Architecture
 2.1 Interface Processor
 2.2 Control Processor
 2.3 Relational Database Engine
 2.4 Hierarchical Memory
 2.5 Maintenance Processor

3. Relational Algebra Processing Algorithm
 3.1 Merge-Sort Relational Algebra Algorithm
 3.2 Merger Algorithm

4. Delta Command Set
 4.1 Command-Tree and Transaction
 4.2 Relational Algebra
 4.3 Set Operation
 4.4 Set Comparison
 4.5 Aggregate Function
 4.6 Arithmetic Operation
 4.7 Definition
 4.8 Update
 4.9 Transaction Control
 4.10 Input/Output
 4.11 Asynchronous commands
 4.12 Miscellaneous commands

5. Internal Schema
 5.1 Tuple Based Schema and Attribute Based Schema
 5.2 Clustering

6. Relational Database Engine (RDBE) Details

7. Hierarchical Memory Subsystem

8. Implementation Designs

9. Performance Estimation

10. Towards a Knowledge Base Machine

11. Conclusion

12. Acknowledgment

References

A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor

Shigeki SHIBAYAMA, Takeo KAKUTA, Nobuyoshi MIYAZAKI,

Haruo YOKOTA, and Kunio MURAKAMI

ICOT Research Center

Institute for New Generation Computer Technology,

Mita Kokusai Bldg. 21F, 1 - 4 - 28, Mita

Minato-ku, Tokyo 108

ABSTRACT

This paper describes the basic concepts, design and implementation decisions, standpoints and significance of the database machine Delta in the scope of Japan's Fifth Generation Computer Project. Delta is planned to be operational in 1985 for researchers' use as a backend database machine for logic programming software development. Delta is basically a relational database machine system. It combines hardware facilities for efficient relational database operations, which are typically represented by relational algebra, and software which deals with hardware control and actual database management requirements. Notable features include attribute-based internal schema in accordance with the characteristics found in the relation access from a logic programming environment. This is also useful for the hardware relational algebra manipulation algorithm based on merge-sorting of attributes by hardware and a large-capacity Semiconductor Disk for fast access to databases. Various implementation decisions of database management requirements are made in this novel system configuration, which will be meaningful to give an example for constructing a hardware and software combination of a relational database machine. Delta is in the stage between detailed design and implementation.

1. Introduction

Delta is a hardware-oriented relational database machine which is in the detailed design stage and will be operational in 1985. Delta is planned to be one of the software development tools in Japan's Fifth Generation Computer Project [MOTO 83].

Another principal new hardware which will also be finished in the first stage Project is the Sequential Inference Machine (SIM), which is constructed in order to support efficient logic programming language processing [UCHI 82]. Delta will be used as a backend database machine in a local area network environment. A number of SIM's will be connected to the Ethernet-like network as the hosts of Delta.

2. Delta Architecture

Delta's global architecture [SHIB 82] is shown in Fig.1. This figure, however, does not show the implementation details of an actual machine. In that sense, Fig. 1 shows the conceptual function distribution of a database machine. The concept behind the architecture is that functions needed for a database machine should be distributed to efficiently construct a full system. Important functions which are mapped on Delta subsystems are described in the following chapters by means of subsystem descriptions.

2.1 Interface Processor

The Interface Processor (IP) manages interfacing functions to connect a database machine to the overall computing environment such as host machines or local area networks. This portion should be independent of the control portion for flexible interface requirements. Some interruption type database commands could be analyzed here to avoid the rather time-consuming command analysis process performed in the control portion and make it responsive for users.

2.2 Control Processor

As the functionally distributed database machine is comprised of several separately working processors, a certain central control function which manages the coordination of each subsystem is needed. Distribution of control as well as distribution of operation is required for a regularly-structured highly-parallel machine. However, database operations include not only data intensive operations (join operation is the typical one) but database management facilities such as recovery functions, security control and transaction management to maintain data consistency. These should be fulfilled in a working database management system and also in a database machine once it begins to be actually used. Proposed highly-parallel architectures perform these functions poorly and in some cases do not even consider them. uery analysis or command compilation are also time-consuming jobs for a database machine if it receives a high-level user language such as S L [CHAM 76]. Considering these functions, database machines which are constructed only of data intensive operation-oriented resources are unlikely to tolerate real life database manipulations. To perform these tasks, a control portion, which is constructed upon a general-purpose computer, is needed. What kind of processor can complete these jobs efficiently is a subject for debate in future database machine research. This could be a conventional von Neumann machine or an inference-based new architecture machine.

2.3 Relational Database Engine

Relational database operations pose a heavy computational burden on conventional von Neumann architecture. This is particularly true when symmetrical accesses to database are required. Indexed internal schemata will perform well only when indexed keys (attributes) are handled. To perform full-range relational algebra operations in a reasonable time, some dedicated hardware is needed. But the use of dedicated hardware without considerations for storage portion architecture will hardly produce a good result. Relational database engine (RDBE), the dedicated hardware for performing relational algebra and other Delta commands, should be considered in conjunction with the Hierarchical Memory subsystem which supplies attribute data to RDBE in a stream at a high bandwidth. There are various kinds of hardware proposed for relational database operations [BANC 82],[SCHW 82],[TANA 82],[KITS 83],[DEWI 79]. We think that the following points are the properties required for a database engine:

- o Stream synchronous processing (does not disturb the data stream flow)
- o Processing a stream on-the-fly (pipeline processing)
- o Processing a stream in one scan (repeated stream transfer should be minimized.)
- o Regular architecture to exploit the use of VLSI technologies
- o Applicable for wide range of database commands

By on-the-fly we mean an operation in which overlapping of data processing and the data transfer is carried out. A database engine should be designed to exploit high bandwidth data transfer from the storage portion. Processing-bound database query execution should be avoided as much as possible, enabling the data to flow smoothly from storage. This means that we expect a high-speed data transfer between the engine portion and the storage portion (possibly from the cache and not from the moving-head-disk) and processing the transferred data without delay. An engine should perform better than the transfer rate if the storage subsystem becomes fast with architectural enhancement. This feature requires the capability of processing a data stream on-the-fly from storage, in combination with a high-bandwidth storage hierarchy. And the other factor to make query execution fast is that stream data should not be transferred repeatedly to fulfill an operation. For example, if a database engine can perform a single condition selection at a time, the S L query:

```
SELECT employee_name, age
FROM employee_list
WHERE age BETWEEN 25 AND 35
```

will need to scan the age attribute twice to get the result. There are two solutions to this problem. One is to make the database engine cascadable and flow one data stream into cascaded engines so that each of them can operate on the stream differently. The other is to make a single database engine perform multiple-condition operations like the example shown

above. In our database engine algorithm, multiple condition query can be handled, (see Chap.3 for details) succeeding in suppressing the data transfer frequency.

The other requirement for an engine is that it should be designed to exploit the benefits of future VLSI technology. This means that if the basic architectural features include some kind of regular structure, it can be manufactured in mass quantity, hence lowering the cost and justifying the use of a special purpose hardware. The last requirement is that RDBE should support a wide range of database manipulation operations. This is somewhat contradictory to the requirement detailed just above. As the engine becomes sophisticated enough to support a wide range of database manipulations, it will become complicated in design and difficult to be implemented with VLSI technologies. However, if a database engine performs aggregate operations poorly, for example, these should be done somewhere in the database machine, presumably at a slow speed. This will result in uneven execution times in the database machine's command set, which will be unpleasant for users. Therefore, to attain this feature, the database engine is expected to have some kind of general-purpose processing capability such as incorporation of microprogrammed architecture or a general-purpose processing portion. In our design, the hardware-intensive portion of the RDBE is implemented in a regular form to be implemented using VLSI, while operations which need flexible data manipulation are delegated to a general-purpose processor which also takes care of detailed control of the hardware portion.

2.4 Hierarchical Memory

The database storage portion is a most essential one in a database machine. Conventionally, a one-level collection of moving-head-disk devices constitutes a database storage. Currently, incorporation of layers in storage are considered to be promising for efficient database processing. The idea follows the technology of cache memories first considered and implemented to enhance the processing power of CPU by compensating a main memory access gap. The disk cache is the counterpart of the CPU cache, assuming the locality of reference using replacement algorithms derived mainly from LRU (Least Recently Used) strategy. In a database operation, however, data access patterns fit LRU poorly. A database has the sequential access property in its nature because database data often needs to be scanned for search. One somewhat artificial but possible example where LRU performs poorly is as follows:

i) Scan a (part of) relation which overflows the total cache size by one page

   By this, the cache is filled up with the relation, the first page of which is replaced with the last page.

ii) Re-scan the relation once again for another purpose

Cache control looks for the first page in the cache, recognizing the cache miss and then fetching it from a secondary device. The fetched page replaces the second page (next to be searched in

cache 1) with it.

Thus this process is repeated through the last page. In this situation, apparently, it does no good to have a cache, however large it might be. We have to introduce a cache or a hierarchy of caches to enhance the performance. The algorithm should be carefully chosen to fit the database access properties. One candidate for a wise cache replacement algorithm is the object-oriented cache [SCHW 83] or an algorithm which makes use of some kind of semantic information. That a (part of) relation is needed likely means the whole of it and not random subparts of it.

The lowest layer storage device should be constructed on moving-head-disks. This is today's most popular technology (it has the best storage/cost factor with acceptable access speed for on-line database use) and is widely accepted as a non-volatile data storage. Other cost-effective devices, such as optical disks, could be accommodated in a database machine storage, in the sense discussed above, when they have gained good reliability and become commercially available. Mass storage system (MSS) is a current technology and offers a huge amount of on-line storage using a tape-based technology. It can also be used as a low-end storage, but MSS is somewhat special with its properties of the amount of storage space and access time. When it is used, it should be used as a kind of loading device which is activated when the user needs a large new relation for a possibly new transaction. MSS will poorly perform if it is used like a disk.

As has been pointed out [BORA 83], the major bottleneck which exists in the database operation is disk access time. To overcome this, one idea is to have a fairly large amount of cache storage which can contain "all" the relations which are used during a transaction. This is virtually impossible for every query. However, by giving a large capacity fast storage space with a wise replacement algorithm, most database transactions can be carried out within the fast storage. This is an expectation and not a proven fact as yet. But we think that lowered storage cost will justify the use of a very large semiconductor cache in future.

2.5 Maintenance Processor

As Delta is comprised of several subsystems, some care has to be taken to maintain reliability and serviceability. A supervisory processor in a conventional large computer system has a similar role. In Delta we prepared a Maintenance Processor (MP) for this purpose. MP has communication channels to each of Delta's subsystems. When an error occurs in a processor, it reports the status to MP, and MP will judge the state of the entire system and determine whether to continue operation by isolating the error or to shut the system down.

3. Relational Algebra Processing Algorithm

3.1 Merge-Sort Relational Algebra Algorithm

There have been numerous algorithms to implement relational algebra by hardware. We have chosen a merge-sort algorithm for Delta's hardware algorithm. The hardware portion for the relational algebra execution is done in RDBE. The basic RDBE command set concept is related to the fundamental RDBE processing hardware algorithm. Unless the command set is closely associated with the hardware implementation algorithm, efficient processing of that command set will be difficult.

Merge-sort is a well-known algorithm to efficiently manipulate the strings of two or more pre-sorted strings to form a longer sorted string. Logically, there are two or more FIFO memories which contain pre-sorted shorter streams. In the ascending sort, a processor compares the top elements of those FIFO's and outputs the smaller element, at the same time advancing the FIFO which contains the smaller element. This algorithm only outputs each string top as the compared result to form a longer sorted string. If the output is properly controlled by another criterion, most relational algebra operations can be implemented using this algorithm. For example, in two-way merge-sort, if one string is a selection criterion and the other string output is controlled such that those values which match one of the upper strings are output, (note that the two strings are both sorted) selection to the criteria is accomplished.

As shown in this example, if proper output control is accommodated in the course of the merging process, various useful selection and join algorithms can be realized.

3.2 Merger Algorithm

The merger unit in the RDBE is the portion which performs the algorithm described in the previous chapter. The merger is, basically, similar to a merge sorter, as the algorithm inherently contains the merging process. The merger has two stream buffers which are placed before the comparator unit, and performs two-way merge-sorting. The comparator unit must have several fundamental commands to perform classes of relational algebra operations. We will briefly describe the principal merger functions in the following chapters.

3.2.1 Simple Selection

A simple selection is defined as a selection against a single criterion, for example, selecting all attribute items by the qualification of "greater than C." This operation is done by placing the constant value in the top of the first FIFO, flowing the stream through the second FIFO and outputting those items which satisfy the criterion. The time required to perform this is N*T where N is the length of the attribute and T is the transfer unit time. Criteria such as "less than", and "equal" are easily carried out by simple modification of the output control.

3.2.2 Range Selection

Range selection is a little more complex than the simple selection. Range selection is defined as selection of attribute items by the range specified by two values C1 and C2. First FIFO contains the two values in sorted order (C1 at the top if it is

smaller). The process is as follows:

(1) Flows the target attribute items in the second FIFO
(2) Discards the items until the top element exceeds C1
(3) Advances the first FIFO to make C2 appear at the top
(4) Outputs the stream until the top element exceeds C2

It is obvious that range selection which has the form ((X < C1) or (X > C2)), where C1 < C2, is easily done by modifying the control in a reversed manner.

3.2.3 List Selection

List selection is an operation in which qualified attribute items are selected against a list of constant values. Sorted constant values C1, C2,.., Cn are stored in the first FIFO. The operation is carried out by the following procedure:

(1) Target attribute items in the second FIFO are forwarded
(2) Items they are smaller than the top element in first FIFO are discarded
(3) If a match is found, the attribute item (at the top of FIFO) is output
(4) If the top element in the second FIFO exceeds the top element in the first FIFO, the first FIFO is advanced.

3.2.4 Synchronous Comparison

Synchronous comparison is an operation where both first the FIFO and second FIFO are advanced synchronously and the top two items are compared with each other. This is useful when selecting attribute items which have a value which is greater or smaller than the other attribute items. The two streams should be sorted beforehand by tuple-identifier (see Chap.5) to correctly correspond the attribute item values.

3.2.5 Join

Equi-join operation is performed by the following procedure:

(1) The first FIFO is filled with sorted attribute items
(2) The second FIFO is forwarded with the other sorted attribute items
(3) The top two items in both FIFO are compared
(4) If the first FIFO top is greater than the second one, the first one is advanced
(5) If the second FIFO top is greater than the first one, the second one is advanced
(6) If a match is found between two items, both are output or, just in the case of a natural join, only one of them is.

A general join can be performed by modifying the control of the FIFO advances. There is a problem here that when there are duplicate items in an attribute stream, all the possible combinations should be obtained in join operations. This is solved by the tag field in the attribute items. If there are duplicate values, the tag field of the duplicate items are set by

the sorter. The merger portion recognizes duplicates, controls FIFO advances and produces all possible combinations in join operations.

There are other commands in addition to the relational algebra operations. However, they all fall into one of these categories.

4. Delta Command Set

The Delta's command set is shown in Fig.2. It consists of several classes of commands. Host software is responsible for converting user queries to the Delta command sequence. We named the sequence a command-tree, because a set of relational database query commands forms a tree structure of Delta commands. In the next section, command-tree and the transaction concept provided in Delta are described. The following sections are brief descriptions of the categorized Delta commands.

4.1 Command-Tree and Transaction

A command-tree is a set of Delta commands which represents a meaningful query. A command-tree example is shown in Fig. 3. In this example, the permanent relation 1 and the permanent relation 2 are semi-joined to produce the temporary relation 1 from the permanent relation 1, while the temporary relation 2 is selected from the permanent relation 3. The temporary relations 1 and 2 are then natural-joined to produce the intermediate relation 1, which is the result relation of this command-tree. According to Delta's terminology, temporary relations are erased when the command-tree has produced a result relation

(intermediate relation). Intermediate relations are given a name and can be used across the command-trees. They are, however, erased at the end of the transaction.

There is a notion of transaction in Delta. A transaction is composed of a set of command-trees which are enclosed by a start-transaction command and an end-transaction command. The command pair groups a set of command-trees in between as a transaction. Delta will support update consistency by user's specification of these transaction control commands. This means that updates to the database can be undone by aborting the transaction (abort-transaction command) in which the updates took place, or can be committed also by committing the transaction (commit-transaction command). For read-only database accesses, committing a transaction means that the result relation can no more be used as an intermediate relation, inhibiting subsidiary queries. The only exception to this is when it is used for output. Before a commit-transaction command, intermediate relations can be used as source relations of the command-trees to follow.

4.2 Relational Algebra

Relational algebra-type commands come directly from the original relational algebra. Usually duplicate tuples are not eliminated automatically. Duplicate elimination can be explicitly specified by Unique command (see 4.12 Miscellaneous Commands). Relational division is not included in the command set because it can be done by a combination of other commands and the frequency of the divide command is not very large. We

provide three types of join commands; natural-join which is considered to be most useful, theta-join for general classes of join and semi-join for efficient use of Delta.

4.3 Set Operation

Set operations between two relations are fully implemented. We have decided to implement Cartesian Product command for optimization reasons. Relational algebra queries with some specific execution ordering are sometimes more elegantly represented using Cartesian Product operation. In Union command, resultant duplicates of tuples are automatically eliminated in contrast to the other commands. If duplicate elimination is not required in Union command, it is no more than an append operation of one relation to another. Specification of Union command is considered to be an explicit duplicate elimination intention.

4.4 Set Comparison

Set comparison commands are useful when checking two sets of objects. In Prolog language, meta-predicates such as set-of or bag-of are provided. These predicates are useful for interfacing logic programming language and relational database [KYKM 83]. Set comparison can operate on those set results. Contain command can also be used to perform least-fixed-point operations. By explicitly checking the set equality between source relation and result relation after a join operation, a least-fixed-point set can be obtained.

4.5 Aggregate Function

Aggregate functions are often needed in an actual database operating environment. They are also useful for obtaining statistical database characteristics. Of course, this can be done by selecting all the tuples, sending them to the host, and letting the host calculate the aggregate values. However, communication between host and Delta is done via a local area network, so it would be costly to send all the tuples every time when such aggregate operations are issued. Furthermore, the hosts are logic programming based inference machines, which are not very good at arithmetic speed. This is the reason why Delta supports aggregate type commands internally.

4.6 Arithmetic Operation

Arithmetic operations on numerical values are not essential for a database machine. However, there is certainly a class of update queries, for example, increasing the salary for all of the employees by 20 %, which needs calculation throughout the attribute values. Because of local area network transfer overhead, we also provided this command set. (Note that the arithmetic type update shown in the example requires the attribute transfer twice.)

4.7 Definition

Definition-type commands are classified into two categories. One deals with definition and deletion of relations and the other deals with attribute appending and dropping. The latter is added to easily manipulate relation schemata.

4.8 Update

Update-type commands are "delete", "insert" and "update" commands. The format and syntax of update commands are somewhat specific. The following is the delete command syntax:

[CN,del,TRN1,TRN2]

CN stands for command number, the identifier of a Delta command. The 'del' specifies the command and TRN1 specifies a target relation (the relation from which tuples are deleted) and TRN2 specifies another relation which contains the tuples (derived from TRN1) to be deleted. There is an alternative way to specify the criterion in the command to delete tuples like this:

[CN,del,TRN1,attribute1 < 300].

In this case, only those tuples which can be specified by the condition field of the command can be deleted. We decided to choose the former syntax to enhance the power for specifying the update criteria.

4.9 Transaction Control

Transaction control-type commands are used to control the update consistency as well as freezing the result relations. Freezing means that the transaction can no more issue commands to Delta except Input/Output commands, in effect, guarding the intermediate relation from further modifications. This is done by commit-transaction command as well as the update commitment.

Abort-transaction command means the rollback of updates done previously in that transaction scope. Concurrency control is also done behind the transaction control command. Delta adopts a two-phase lock method to maintain concurrency control. The user can specify a relation list to lock during the transaction in start-transaction command, or the system automatically locks the concerned relation when it is needed to keep the consistency of updates in a transaction. All the locks are undone upon the receipt of commit-transaction or end-transaction command.

4.10 Input/Output

Input/Output commands are used to transfer relations between hosts and Delta. These commands form a command-tree in one command. Only Input/Output commands are valid after commit-transaction command. (Of course, Input/Output commands are good anywhere in a transaction.) The "get" command has the effect of retransferrig the result more than once.

4.11 Asynchronous Commands

Asynchronous-type commands are used to interrupt Delta. These commands expect a fast response from Delta. One of these is the abort-processing command which terminates the current command execution. The other is the sense-status command to sense Delta's execution status. Normally, Delta commands are executed by the arrival order. However, asynchronous-type commands are executed as soon as they arrive at Delta. Abort-processing provides users a facility like Control-C or Break in a usual program run. However, if an update-type command has been executed in the transaction to which the

abort-processing is issued, the host has to command Delta to abort that transaction. This is because abort-processing terminates a command execution at a point where data might be inconsistently dirtied.

4.12 Miscellaneous Commands

Miscellaneous-type commands are a collection of utility-type database commands. One very important command here is the "sort" command, which is often used for output. The "unique" command is used in combination with other commands when the user wants a duplicate-free relation. Group-by type commands are provided to perform functions known by S L's group-by operation.

5. Internal Schema

5.1 Tuple-Based Schema and Attribute-Based Schema

The method of choosing physical or internal schema influences a database machine's performance as much as its internal configuration does. There are two major methods on how to physically store a relation. One is to store it in tuple- or row-based form and the other is to store it in attribute- or column-based form. In Delta we have chosen the attribute-based schema as the internal representation of relations. There are merits and demerits in each schema.

< Merits of tuple-based schema >

- o Fits original relational database (set theory) concept
- o Read-out of tuples is smooth from sequential storage device

< Demerits of tuple based schema >

- o Needs to access extra information such as unnecessary attributes in a query
- o Symmetrical access to attributes is difficult (secondary indices to many attributes are hard to make and maintain)

< Merits of attribute-based schema >

- o Only concerned attributes should be read
- o Symmetrical manipulation of attributes can be easily done

< Demerits of attribute-based schema >

- o Needs tuple reorganization (tuple reconstruction) for output
- o Needs tuple identifier (extra storage space) for tuple reconstruction

As mentioned in the first chapter, Delta will be used in a logic programming language environment. It is highly desirable to handle almost every attribute in a symmetrical way. In other words, a system can not guess a predetermined access path to a relation. This is closely related to the non-deterministic behavior of logic programming language execution, typically when backtracking of literals occurs. This fact made us select the attribute-based schema. The other strong reason why we chose it is that the schema is appropriate for our RDBE's merge-sort hardware algorithm. We did not want to flow unnecessary

attributes through RDBE, which would result in an increase of data transfer time between RDBE and HM. We also considered the difficulty RDBE has in recognizing the boundaries between a field to be sorted and compared and other trailing attribute(s) in RDBE. Later, however, the latter reason proved to be of little significance. There still exist reasons for RDBE to recognize the field boundaries even if we had chosen the attribute-based schema. One obvious example is the existence of tid (tuple identifier) field and the other is the fact that at times RDBE still has to handle some tuple-like streams called concatenated attribute class data which is considered as a part of relation projected on some attributes. In the case of tid field, the situation is better because there are only two distinct fields in a data stream (like binary relation) and it makes the hardware design much simpler. However, to perform set operations, the tuple should be reconstructed before processing by RDBE. In this case, a usual tuple-based relation is forwarded through RDBE. Unnecessary attributes, however, need not be transferred.

5.2 Clustering

A naive relational database machine design might scan a relation (or attributes in attribute based schema) every time a query is issued. Certain clustering is necessary for narrowing the search space before brute-force searching takes place. This is particularly true in an environment where a storage hierarchy is accommodated in a database machine system. If a cache (or a similar smaller but faster storage) is filled up with relations most of which are only staged up to be disposed, the significance of that expensive storage can not be justified. Much works has

recently been done on efficient clustering of relations [TANA 83]. In Delta, however, we decided to use an attribute-based schema. In a tuple-based schema, it might be difficult to access several attribute values evenly. (This is the reason why a smart indexing method is needed.) An attribute-based schema has only to consider two fields, namely attribute value field and tid field, for clustering. We therefore adopted a simple two-level clustering method. An example of this clustering is shown in Fig. 4. First, the attribute is sorted according to the attribute's value, divided in multiple first-level clusters according to the range of values. Secondly the first-level clusters are separately sorted according to tid values in each item. (An item, in our terminology, is a pair of tids and a value of attribute.) Thus one leaf of this schema (which corresponds to a physical page in storage) contains items the values of which are in a certain attribute value range and also tid values which are in a certain tid value range. Though the access patterns are dominant in deciding the effectiveness of a clustering scheme, we observed that this clustering is effective in symmetrical access pattern queries [MKSY 83].

6. Relational Database Engine (RDBE) Details

RDBE is the key component for the processing of relational algebra in Delta (Fig. 5). RDBE incorporates the basic merge-sort algorithm described in the former chapter. In a word, RDBE is a hardware implementation of the merge-sort software algorithm. However, to implement it to work in the actual processing stage in a database machine, various difficult problems must be solved.

As discussed in the architecture chapter, RDBE has a general-purpose processor portion which controls the hardware portion (Engine Core) and processes certain classes of Delta commands such as aggregate functions and arithmetic operations. Engine Core is a piece of new hardware which is comprised of a first stage input aligner, second stage pipeline sorter and last stage merger. The second stage sorter is further comprised of smaller stages called sorting stages. The sorter follows the idea of hardware pipeline merge-sorting by [TODD 78]. A sorting stage is cascaded to another one, forwarding a partially sorted stream to it. An N-th sorting stage (starting with 1) has two FIFO memories which can contain 2**(N-1) items. The first sorting stage sorts one item from one FIFO and one item from another, producing a two-item partial sort result into the second stage FIFO. The second sorting stage compares the two two-item streams, producing two four-item streams. This process is repeated until the last stage, in effect producing a 2**N-item sorted stream.

A sorting stage does not have to wait for the arrival of two completed streams. Upon the arrival of the first item in the second stream, the first item can be compared with the top item in the previously stored first stream. (The first item of the second stream is the smallest item in the second stream.) The sorting time of this sorter is (2*N+2**N)*T where N is the sorting stage count and T is the time unit to transfer an item. In actual attribute sorting, the length of an item varies in magnitudes of range (for example, 2B to 2KB). To handle varieties of length, each sorting stage has a fixed size of

memory and simulates the FIFO operation by the use of pointer registers.

The sorter has twelve sorting stages and the last sorting stage has two 64KB-memories. This sorter can sort either (1) 4K items if each item is smaller than 16B (64K/4K) or (2) 2*N items where N is 64K/item-length. The sorting stage comparator part will be implemented by discrete components initially and replaced with LSI implementation in the final configuration.

The input aligner portion rearranges the field orders before a stream of items is sent to the sorter portion. As a stream is composed of several fields such as value fields and tid fields, the fields should be rearranged so that the sorting key field will appear at the top of each item to match the sorting algorithm. The last stage merger, after performing the proper operation, arranges them back to the right fields order.

The merger is a unit which performs relational algebra and other Delta-command-related operations on a sorted stream. The fundamental structure of the merger is the same as the sorter. The merger has two FIFO memories and a comparator. The comparator, however, is more complex than the sorting stage. In addition, the FIFO controller has the capability to point to some item in FIFO in a more flexible way. For example, when merger performs a natural-join operation, the control procedure proceeds as follows (here, no duplicates are assumed in joined attributes for simplicity):

(1) Repeat until the first attribute is exhausted

(2) Send a bufferful of the first attribute to RDBE

(3) Sort the first attribute and load the merger's first FIFO with it

(4) Continue until the second attribute is exhausted

(5) Sort a bufferful of the second attribute and supply it to the merger's second FIFO

(6) First FIFO rewind (set to originally loaded state)

(7) Continue until a bufferful of the second attribute is exhausted

if top of second FIFO is greater, then pop second FIFO

if top of second FIFO is equal, then output it and pop second FIFO

if top of second FIFO is less, then pop first FIFO

The time required to perform this operation is as follows (no clustering effect is assumed here):

$$(M/B * (3 * L + \log(L) + N)) * T$$

where M is the size of the first attribute, N is the size of the second attribute, B is the merger buffer size, L is the FIFO size and T is the unit item transfer time. Note that if M is smaller than B, the join can be accomplished by flowing the second attribute once through the merger. This does not depend on the size of the second attribute, because the second attribute can be divided in the merger and the divided subparts can be continuously joined without disturbing the stream flow.

The merger unit is associated with a general-purpose processor for performing aggregate functions, arithmetic operations, complex condition selections and a subset of set operations. Dedicated pieces of hardware for performing these operations are required for higher performance. However, time did not allow us to design those ones. We decided to concentrate on the Engine Core to perform relational algebra. When RDBE is commanded to perform an operation which should be done in the general-purpose processor, the data stream is forwarded to the processor after the stream flows through the merger. In this case the stream flow speed is limited by the processor's processing speed.

7. Hierarchical Memory Subsystem

The Hierarchical Memory subsystem (HM) is responsible for storing, accessing, clustering and maintaining relations. HM's configuration is shown in Fig. 6. The lowest storage device is state-of-the-art moving-head-disks, the total capacity of which will be 20 GB in the final Delta configuration. HM is provided with a large capacity Semiconductor Disk (SDK), the capacity of which will be 128 MB also in the final configuration. SDK is a semiconductor RAM memory system which is protected against power failure and hence appears to be non-volatile. When power failure is detected, emergency power is supplied from the battery of the power supply system until dumping of SDK content to MHD is finished. The controller of HM is a general-purpose computer named HMCTL. HMCTL is the brain in HM and it performs the tasks assigned to HM. HMCTL's roles are as follows:

o Memory management in SDK and MHD

. SDK and MHD data staging and destaging

. CP directory area management

. SDK power failure management (non-volatility)

o Stream preparation for RDBE

. Buffer assignment in SDK

. Clustering search

. Stream arrangement (page fragment elimination)

o Stream transfer between HM and RDBE

. Channel activation

. Multiple stream management

o Transaction rollback

. Shadow management

. Update logging

o Tuple and attribute transposing

. Transposing to tuple

. Transposing to attribute

(For bidirectional tuple and attribute conversions)

A general procedure where HM prepares and forwards streams to RDBE is as follows:

(1) Buffers (output qualified buffers and result receiving input buffers for RDBE) are prepared.

(2) The stream (data in the output buffers) is forwarded to RDBE

(3) The input stream from RDBE (result of the RDBE processing derived from output stream) is received

(4) Buffers no longer usable are released

These are directed by CP or RDBE via HM subcommands. Usually HM subcommands are the principal command communication means between HM and other Delta subsystems. HM is almost always passive in the sense that it is directed by other subsystems. HM's most principal role is the stream preparation and transfer.

In the stream preparation process, HM is typically commanded by the prepare-qualified-buffer subcommand. This subcommand means that HM should prepare an attribute in a stream buffer using the qualification specified in the subcommand. The qualification example is value range specification like " greater than 1000 ". As is described in the clustering chapter, HM maintains a two-level clustering directory and looks for qualified pages when it receives the subcommand. It is not required that HM should filter out attribute items which do not satisfy the qualification. This qualification is a static one and HM fetches such pages which may contain attribute items satisfying the qualification.

Another important role of HM is the data recovery function. HM is the central portion for storing databases, it makes shadows of the update pages directed by CP for later possible transaction rollback specification. HM is further responsible for

transposing tuples to attributes and vice versa. Sending result tuples to the host is done by making buffers which contain necessary attributes to constitute the tuple, transposing them into a tuple buffer and transferring it via IP to the host. The transposing, in either direction, is done as much as possible in SDK to reduce the overhead associated with the attribute-based schema.

8. Implementation Designs [SHIB 83]

Delta has adopted new pieces of hardware in its architecture. The most notable ones are RDBE and non-volatile large capacity Semiconductor Disk. Other subsystems are basically built up with off-the-shelf components such as minicomputers or general-purpose computers. The Interface Processor will be implemented using a one-board minicomputer with on-board 512KB main memory. The Control Processor will also be implemented by the same minicomputer with 1 MB main memory. The Relational Database Engine (RDBE) is a new piece of hardware, our idea being based on an algorithm using the hardware supported merge-sorting implemented by the RDBE. Most hardware intensive efforts are being done for the implementation of RDBE, in the sense that it will become the first practical hardware relational database engine. RDBE will also use a minicomputer to control the RDBE's hardware resources. In order to manipulate various relational database processing requirements, for instance, floating-point data calculations, RDBE could not help but become a little sophisticated.

The Hierarchical Memory (HM) subsystem is implemented using a general-purpose computer as a controller (HMCTL), a large amount of fast semiconductor random access memory in the form of a semiconductor disk (SDK) and large capacity moving head disks. The SDK is used for temporarily storing classes of relations generated and managed in Delta. The SDK will be non-volatile (at least from a software point of view) to avoid disk accesses invoked by write-through storage management. We realize there remains a lot to be investigated and researched to make a real hardware-oriented HM. So we decided to simulate the HM using a general-purpose computer system for collecting performance data and making the points to be improved clear in this research. One of the most decisive factors in choosing a general-purpose computer as the HM is that it provides an operating system containing control software on state-of-the-art disks, the capacity of which is over 2GB per unit.

9. Performance Estimation

The performance of a database machine or a database management system is greatly affected by usage patterns, or how the database is accessed by users. Our preliminary assumptions for the accesses which will be made in a logic programming environment are as follows:

(1) Relations are of only a few attributes.

This is because the databases are closely associated with the "facts" of logic programming languages. Usually logic programming language programs contain only a small number

of arguments.

(2) Attributes are accessed in a non-deterministic manner.

This is a property of logic programming languages. Especially when backtracking takes place, we can not tell which one of the arguments is instantiated. There are classes of arguments which are only subsidiarily accessed. However, compared with usual key-based accesses to databases, there may be a lot of attributes from which the accesses are made. This is one of the reasons why we adopted an attribute-based schema.

(3) Databases are divided into two usage categories.

As the environment is research oriented, the usage categories include personal database usage as well as shared database.

(4) High degree of concurrency may manifest when the total system (SIM's, local area network and Delta) are used in a TSS-like manner.

(5) Select, Join and Project are the high frequency commands.

Because they are frequently found in our interfacing method between logic programming languages and relational databases.

Apparently, the local area network is not a very high-bandwidth one. We will have a transfer rate around 10M bit per second. This can be a bottleneck if heavy query traffic is always flowing through this path. We will not discuss the local area network problem further here. It has to be taken account of in the total system configuration.

9.1 A Delta Performance Estimation

Delta's command-tree execution steps are divided into the following substeps neglecting the host-to-IP command-tree and response transfer time:

(T1) IP software overhead to invoke IP-to-CP command-tree transfer
(T2) IP-to-CP command-tree transfer
(T3) CP software overhead to invoke command-tree analysis
(T4) Command-tree analysis, generating subcommand sequence
(T5) Subcommand sequence execution
(T6) CP software overhead to invoke response transfer to IP
(T7) CP to IP response transfer
(T8) IP software overhead to invoke response transfer to host

Among these, (T5) is the actual database access operation. So (T5) is further divided into the following procedures:

(T5.1) RDBE subcommand execution
(T5.2) HM subcommand execution
(T5.3) CP to RDBE subcommand transfer overhead and transfer time

(T5.4) CP to HM subcommand transfer overhead and transfer time

There is no HM to RDBE data transfer time included here. This is because it is hidden in RDBE subcommand execution time, a processing scheme we adopted. Besides the transfer time, some of these substeps are partially done in parallel. So the mere amount of execution times of these substeps will give us an underestimate performance figure. But as the details of these substeps are not sufficiently clarified, it will suffice to have a simple sum for rough estimation.

9.2 A selection example

We now consider an example selection query in S L form:

```
SELECT a1, a2,..., an
FROM A
WHERE ai IN [value list]
```

where A is a relation composed of 10 attributes, having 10000 tuples. The a1, a2 and so on are the attributes among the 10 attributes. We assume 10B for each attribute here.

The execution time characteristic for selectivity according to a deterministic simulation is shown in Fig. 7. This figure assumes a high semiconductor disk hit ratio. For certain ranges of selectivity, there are different dominating factors. For the selectivity range between 0.01% to 1%, the dominating factor is the increasing tid join time. Buffer preparation in semiconductor disk is not so time-consuming, because the

intra-semiconductor disk transfer is fairly fast compared with tid joins. For selectivity factors between 1% and 10%, the estimation curve shows a plateau. In this area, the processing time is dominated by the full tid joining. All the attributes should be scanned for tid join in this area. For selectivity range between 10% and 100%, along with the tid join time which form the plateau, tuple reconstruction time becomes influential. The effect of tuple reconstruction rapidly becomes great.

This result, though still not sufficiently quantitative, indicates that the incorporation of a large capacity semiconductor disk is effective for a high performance database machine. The effort to increase the hit ratio by the wise replacement algorithm is the key to effectively utilize the semiconductor disk. We assume that the performance of Delta in the high hit ratio range will be around several hundred milliseconds in the selection example.

10. Towards a Knowledge Base Machine

The final goal towards which Delta's research line aims is knowledge base machine construction. The Fifth Generation Computer Project has adopted logic programming language as a kernel programming language. This will be the basis for all the research to be carried out during the Project. We think that the knowledge representation problem is the key to construct a knowledge base machine. The relational model has a good affinity to logic programming language because both of them have logic in their foundation. Some attempts have been made to combine relational database and logic programming language [CHAK

82],[TAHO 83]. We have presented a compiled approach which defers the evaluation of literals in Prolog when the literal has alternatives in the external database (database machine) in the form of facts [YOKU 83]. Besides this approach, there may be various other approaches towards attainment of the final goal of a knowledge base machine. Our standpoint is based on the amalgamation of logic programming language and relational database. We do not think that the relational model is sufficient for a general class of knowledge base machine achievement. However, the combination of a relational database machine and inference mechanism is a good candidate for a future knowledge base machine [MKMS 83]. To more closely combine the database and logic programming language, rules (unit clauses with variables and non-unit clauses) should be manipulated in database machines. This implies that the database machines should be able to handle structured data in general and have the mechanism to perform a unification operation as a relational algebra level database operation [YOKO 83]. A unification engine, which operates on a data stream which represents a set of structures instead of an attribute, is one method of performing efficient knowledge base unification. This is easily applicable in Delta's architecture. All that is needed is to place a unification engine along the RDBE. In this case also, some knowledge clustering technique should be accommodated for the efficient use of knowledge space, which is still in the elementary research stage. We think that, like the relational database case, semantic information should be used in the clustering of knowledge bases. For research purposes, Delta will be connected to a sequential inference machine via a shared communication

memory. This will provide a powerful research tool for the pursuit of a knowledge base machine.

11. Conclusion

We have described the architecture of Delta, its commands, subsystem roles, and to some extent RDBE and HM detailed considerations. Delta's new architecture and major decisions such as pipeline relational algebra processing, incorporation of a large capacity non-volatile Semiconductor Disk, and semantics-based clustering will enable it to operate efficiently in relational algebra based commands processing, particularly when symmetrical accesses to the database are frequent. From a database management viewpoint, Delta has many features which current software database management systems have. More detailed implementation decisions are now being made in the course of the detailed software design and manufacturing steps. When completed, experiment and collection of performance data in actual usage, not only as a database machine but also as a research tool for a knowledge base machine, will be the next research step.

12. Acknowledgment

The authors would like to express their appreciation to Mr. K. Fuchi, director of ICOT, who provided the opportunity to conduct this research, and offer special thanks to the Toshiba and Hitachi researchers and engineers, who are engaged in the development of Delta. Many decisions in this paper were made through discussions with them.

[References]

[BANC 82] Bancilhon, F., et al., "VERSO : A Relational Back-End Data Base Machine", Proc. International Workshop on Database Machines, August, 1982.

[BORA 83] Boral, H., Dewitt, D., "Database Machines: An Idea Whose Time has Passed? - A Critique of the Future of the Database Machines", Proc. Int'l Workshop on Database Machines, August, 1983.

[CHAK 82] Chakravarthy, U., et al., "Interfacing Predicate Logic Languages and Relational Databases", Proc. First Int'l Logic Programming Conference, Sept., 1982.

[CHAM 76] Chamberlin, D., et al., "SE UEL 2: A Unified Approach to Data Definition, Manipulation, and Control", IBM Journal of Res. and Develop., 1976.

[CODD 70] Codd, E., "A Relational Model for Large Shared Data Banks", Commun. ACM 13, 377, June, 1970.

[DEWI 79] Dewitt, D., "DIRECT - A multi-processor organization for supporting relational database management systems", IEEE Trans. Comput., Vol.C-28, No.6, 1979.

[KITS 83] Kitsuregawa, M., et al., "Application of Hash to Data Base Machine and its Architecture", New Generation Computing, Vol.1, No.1, pp. 63 - 74, 1983.

[KMSY 83] Kakuta, Miyazaki, Shibayama, Yokota, Murakami, "A Relational Database Machine "Delta" (I), (II), (III)", 26th National Conference, Information Processing Society of Japan, 4F-6, 4F-7, 4F-8, 1983.

[KUYO 83] Kunifuji, Yokota, et al., "Interface between Logic Programming Language and Relational Database Management

System (1) -- Basic Concepts --", 26th National conference, Information Processing Society of Japan, 5C-9, 1983.

[KYKM 83] Kunifuji, S., Yokota, H., "PROLOG and Relational Databases for 5th Generation Computer Systems", ICOT Technical Report TR-002, 1982. (revised version 1983)

[MKSY 83] Miyazaki, N., et al., "On Data Storage Schemes in Database Machines", Proc. 27th National Conference on information Processing, 2K-5, Oct., 1983. (in Japanese, English version to appear as ICOT Technical Memorandum)

[MKMS 83] Murakami, K. et al., "A Relational Database Machine: First Step towards a Knowledge Base Machine, ICOT Technical Report TR-012, 1983.

[MOTO 83] Moto-oka, T., "Overview to the Fifth Generation Computer System Project", Proc. 10th Int'l Sympo. on Computer Architecture, June, 1983.

[SCHW 82] Schweppe, H., et al., "RDBM-A Dedicated Multiprocessor Systems for Data Base Management", Proc. Int'l Workshop on Database Machines, August, 1982.

[SCHW 83] Schweppe, H., "Some Comments on Semantical Disk Cache Management for a Knowledge Base System", to appear as ICOT Technical Report, Oct., 1983.

[SHIB 82] Shibayama, S., Kakuta, T., Miyazaki, N., Yokota, H., Murakami, K., "A Relational Database Machine "Delta"", ICOT TM-0002, 1982.

[SHIB 83] Shibayama, S., et al., "On RDBM Delta's Relational Algebra Processing Algorithms", Proc. 27th National Conference, Information Processing Society of Japan, 2K-5, Oct., 1983.

[TAHO 83] Tanaka, Y., Horiuchi, K., and Tagawa, R., "Combining Inference System and Data Base System by a Partial Evaluation Mechanism", Proc. First Knowledge Engineering Symposium, Tokyo, March, 1983. (in Japanese)

[TANA 82] Tanaka, Y., "A Data Stream Database Machine with Large Capacity", Proc. Int'l Workshop on Database Machines, August, 1982.

[TANA 83] Tanaka, Y., "Adaptive Segmentation Schemes for large Relational Database Machines", Database Machines, Proc. Int'l Workshop on Database Machines, Springer Verlag, Oct., 1983.

[TODD 78] Todd, S., "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM Journal of Res. and Develop., 22, 1978.

[UCHI 82] Uchida, S., et al., "The Personal Sequential Inference Machine - Outline of Its Architecture and Hardware System", TM-0001, ICOT Technical Memorandum, Nov., 1982.

[YKKM 83] Yokota, H., Kunifuji, S., et al., "How Can We Combine a Relational Database and a Prolog-Based Inference Mechanism ?", Proc. Meeting of WGAI, Information Processing Society of Japan, Tokyo, Nov., 1983. (in Japanese)

[YOKO 83] Yokota, H., et al., "An Investigation for Building Knowledge Base Machines", ICOT Technical Memorandum TM-0019, 1983. (in Japanese)

[YOKU 83] Yokota, H., Kunifuji, S., et al., "Interface between Logic Programming Language and Relational Database Management System (2) -- Implementation --", 26th

National conference, Information Processing Society of Japan, 5C-10, 1983. (in Japanese)

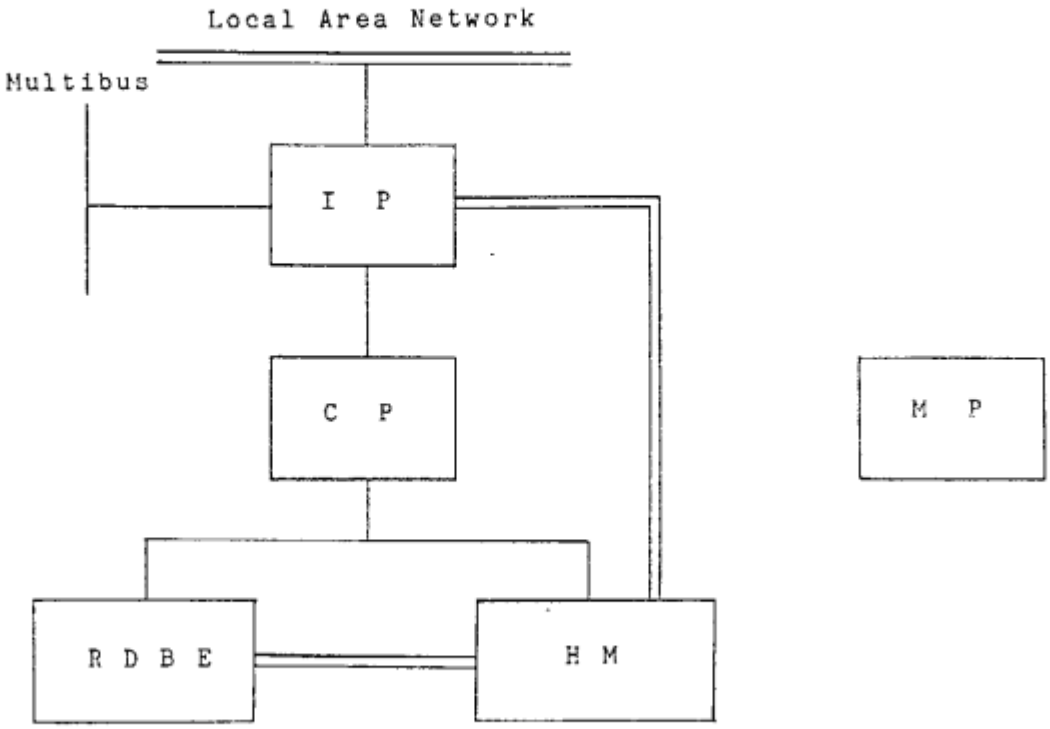


Fig. 1 Delta Global Architecture

Relational Algebra Commands	Aggregate Function Commands
Projection Restriction Compare-Attribute Natural-Join Theta-Join Semi-Join	Count Summation Maximum Minimum Average
Set Operation Commands	Set Comparison Commands
Union Difference Intersection Cartesian-Product	Equal Contain
Update Commands	Input/Output Commands
Delete Update Insert	Get Get-Next Get-End Put Put-Next Put-End
Definition Commands	Transaction Control Commands
Create-Relation Purge-Relation Rename-Relation Attribute-Append Attribute-Drop	Start-Transaction End-Transaction Abort-Transaction Commit-Transaction
Arithmetic Commands	Miscellaneous Commands
Add Subtract Multiply Divide	Sort Unique Group-by Select-Group Copy Classify
Asynchronous Commands	
Sense-Status Abort-Processing	

Fig. 2 Delta Command Set

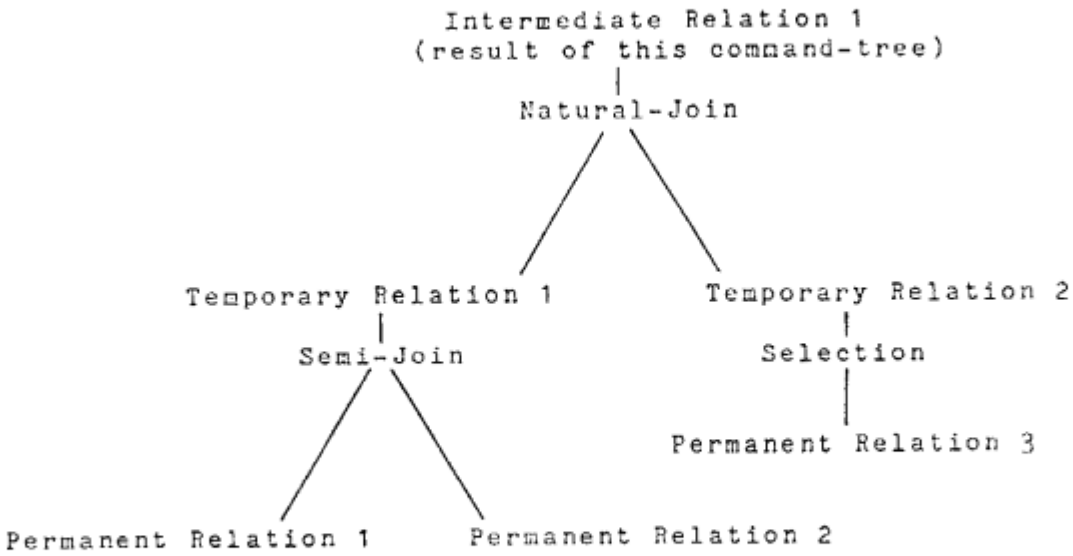
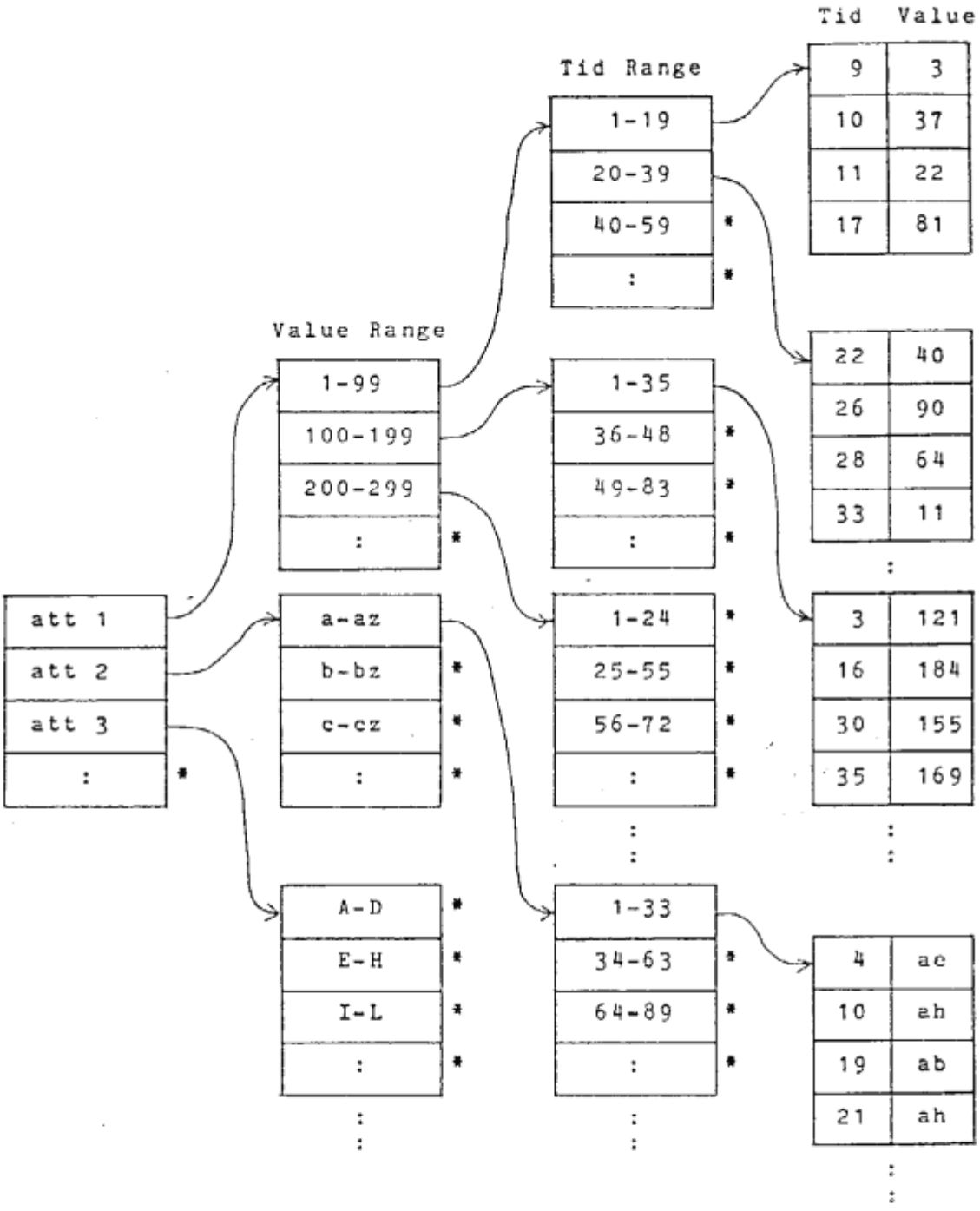


Fig. 3 A Command-Tree Example



Note: Asterisks indicate pointer abbreviations.

Fig. 4 Internal Schema of Delta

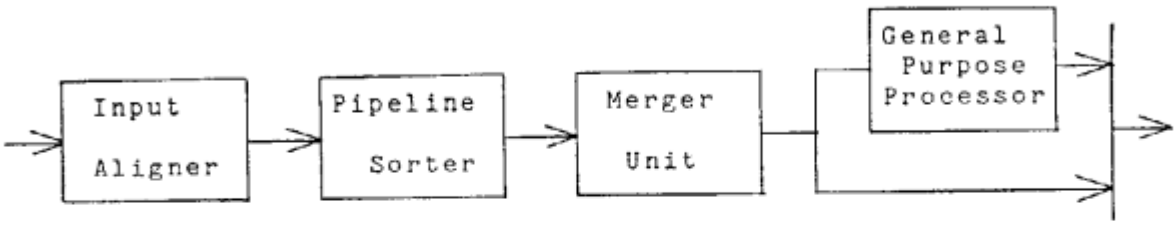


Fig. 5 RDBE Schematic Configuration

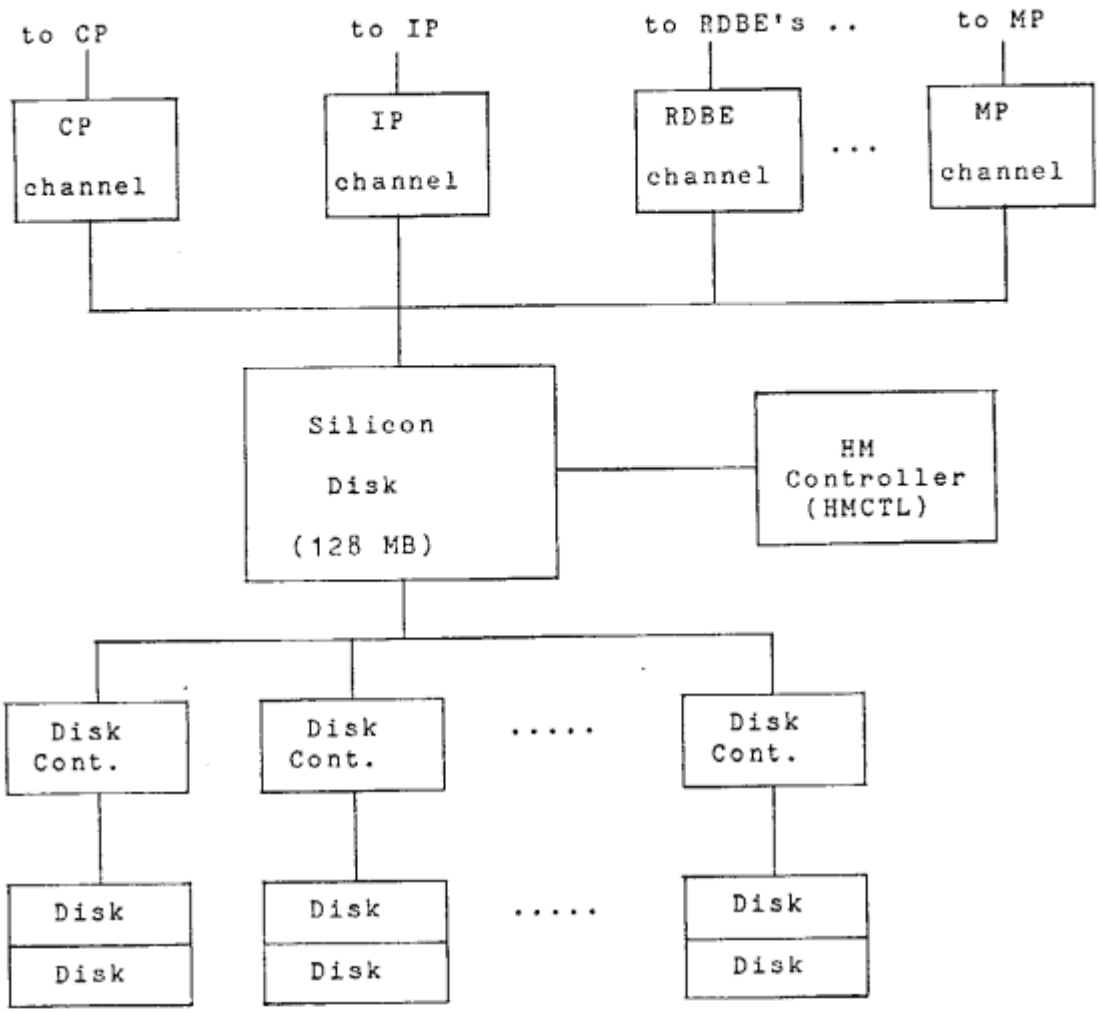
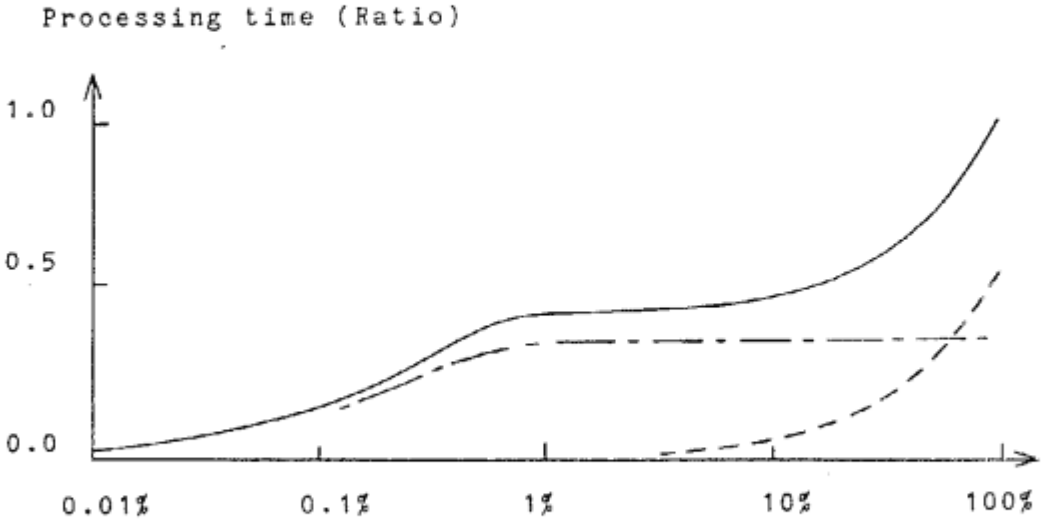


Fig. 6 HM Schematic Configuration



Solid Line : Total Processing Time
Broken Line : Tuple Reconstruction Time
Chain Line : Tid Join Time

Fig. 7 A Simulation Result