TR-089

The Design and Implementation of Relational Database Machine Delta

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

November, 1984

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

The Design and Implementation of Relational Database Machine Delta

Takeo Kakuta, Nobuyoshi Miyazaki, Shigeki Shibayama,
Haruo Yokota, Kunio Murakami
Institute for New Generation Computer Technology
Mita Kokusai Building, 21F,
1-4-28 Mita, Minatoku, Tokyo, 108 JAPAN

ABSTRACT

Delta is a relational database machine under development at ICOT. It has specially designed hardware components to perform relational database operations and a large semiconductor memory to be used as disk cache area. The machine will be used in an experimental local area network environment along with Personal Sequential Inference Machines which are also being developed at ICOT. This paper describes design decisions concerning Delta's architecture and processing algorithms, as well as its overall functions. Delta is expected to be operational with a data storage capacity of 20 GB by March, 1985.

1. Introduction

Japan's Fifth Generation Computer Systems (FGCS) research and development aim to build a prototype of a knowledge information processing system capable of efficiently performing knowledge-based problem solving and inference. Toward this end, a ten-year period has been assigned to the FGCS project, and this period has been divided into three stages.

The knowledge base machine and parallel inference machine are the most important hardware components of the FGCS. In the FGCS prototype to be completed as the final product of the project, the two machines will be integrated through a close link. In the initial stage, however, research and development are proceeding separately for each machine, since the initial stage mainly aims to conduct research and development of individual component technologies to establish the basic technology for the hardware, called the knowledge base subsystem and inference subsystem to be built in the intermediate stage.

In the initial three-year stage, a relational database machine is being developed in order to research and develop the basic techniques necessary for developing a knowledge base machine in the intermediate stage of the project and for investigating a prototype database machine capable of parallel relational and knowledge operations [Murakami83].

Development of the ICOT relational database machine (called Delta) has two purposes. One is to create an experimental environment in which various knowledge base functions and their implementation can be investigated. The other is to connect the machine via a local area network (LAN) [Taguchi84] with the personal sequential inference machine (called PSI) being developed separately [Yokota83]. PSI is a software development tool that makes use of a logic programming language, Kernel Language Version-0 (KL0) [Chikayama83].

This paper gives an overview of the architecture, functions, processing algorithms, and the implementation of Delta. Details of its architecture, query processing flow, and implementation have been reported elsewhere [Shibayama84a,b],[Sakai84]. Various database machines (DBM) has been reported and implemented

[IEEE79],[Bancilhon82],[Schweppe82],[Hsiao83]. Some are software oriented and others are hardware oriented. Hardware oriented database machines are designed to improve the performance of database operations. Delta is a hardware oriented database machine that adopts set-oriented internal operations and specialized hardware to realize these operations. It is one of the first hardware oriented DBMs implemented to store and manipulate large scale databases.

In chapter 2, some fundamental design decisions concerning the architecture are discussed. Functions that are made available to host computers and database administrators are described in chapter 3. Processing algorithms and methods for several basic operations are discussed in chapter 4, and implementation considerations are presented in chapter 5. Performance estimation is described in chapter 6.

2. Architecture

2.1 Fundamental Design Decisions

We have to solve two key problems in the design of a high performance DBM:

(1) Fast relational operations

(2) Efficient access to database storage.

Most DBMs so far proposed adopt parallel processing or specialized hardware to solve the first problem. Our solution is to perform all relational operations on sorted sets and to use a specialized preprocessing hardware to sort data. It is well known that join, a very time-consuming operation, can be executed extremely fast if both operand relations are sorted by their join attributes. Basic set

operations are also processed quickly if operand relations are sorted. Thus, relational operations can be performed efficiently if we can sort relations fast enough. The best software algorithms can sort N items in O(NlogN) time, but there are several algorithms to sort them in O(N) time using specialized hardware. We designed a relational database engine (RDBE) based on a pipelined merge-sort algorithm [Todd78] [Sakai84]. Because most RDBE operations can be done in O(N) time, its processing can be synchronized with the data transfer between itself and the memory subsystem. This is called data-stream processing and can be regarded as an extension of "on-the-fly" processing. Delta has four RDBEs, which can be used in parallel or independently.

There are several methods we may use to solve the access problem.

(a) Parallel I/O devices to reduce access time.

(b) Use of search filters attached to storage devices to reduce the amount of data to be processed by the upper layers

(c) Large cache (buffer) memory to reduce access time

(d) Clustering and indexing techniques to reduce the amount of data to be processed.

Delta adopts a combination of (c) and (d), because they provide more flexibility, and a large low-cost memory is available. This decision may seem somewhat conservative, because several devices that incorporate methods (a) or (b) have already been proposed and implemented. However, because the usefulness and feasibility of these devices remains unproved, we believe our method is more realistic, given the relatively short time allotted to RDBM development.

2.2 Internal Schema

Conventional database management systems (DBMSs) store a relation as a file in which a tuple is treated as a record and an attribute as a field. To rapidly obtain tuples satisfying specific criteria, indexing and hashing techniques are applied. These methods are more useful if the number of attributes frequently used in the criteria is kept small. 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 an unconventional access characteristics, because a logic programming language is used as PSI's language, and because the system is used for knowledge information processing. Access to the database stored in Delta is predicted on the following characteristics, based on the usage of Prolog programs:

(1) Relatively few attributes in typical relations
(2) Uniform distribution of attributes used in conditions
(3) Relatively uniform frequency of access to tuples.

Delta adopts an attribute-based schema to efficiently process these kinds of requests. Instead of storing all the attributes of a tuple together, it splits a relation into a collection of attributes and stores all occurrences of each attribute together. A TID (tuple identifier) is stored along with an attribute value to identify the tuple to which it belongs. To reduce the amount of data to be processed by the RDBE, a two-level indexing method is used, as shown in Figure 1 [Shibayama84a]. The merits of Delta's attribute-based schema are as follows:

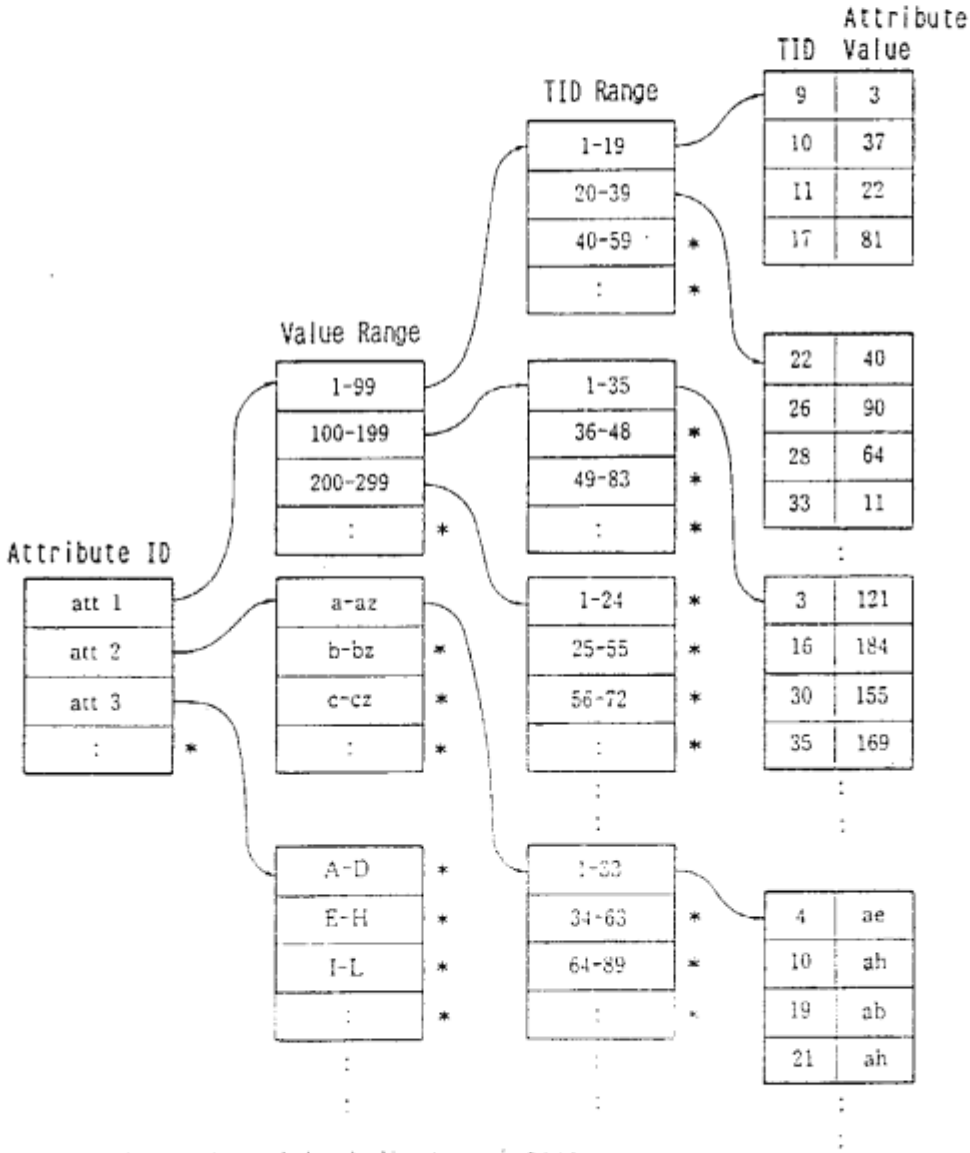
(1) Attributes that are not necessary for a given request need not be read from the secondary storage to work buffer area.
(2) Attributes are treated uniformly.
(3) Unnecessary attributes need not be transferred between RDBE and the memory subsystem.

There are several disadvantages as well.

(1) Transformation between tuple format and attribute-based format is necessary.

(2) Tuple identifiers occupy additional storage space.

To prepare tuples for output (called tuple reconstruction) is a formidable task for conventional DBMSs because it is an operation similar to join. Delta can make effective use of the attribute-based schema because it can efficiently reconstruct tuples using the RDBE.



Note: Asterisks indicate pointers

Figure 1. Internal Schema of Delta

2.3 Architecture of Delta

The Delta architecture was designed based on functional decomposition. Delta principally consists of three kinds of units: RDBEs, a Hierarchical Memory (HM), and a Control Processor (CP). A front-end processor (Interface Processor: IP) is used to connect Delta to the outside world. One more unit, a Maintenance Processor (MP), is included mainly for system supervising, as described later. Thus, Delta consists of five kinds of functional units, as shown in Figure 2. A logical request specifying the operation of an individual unit by another unit is called a subcommand.

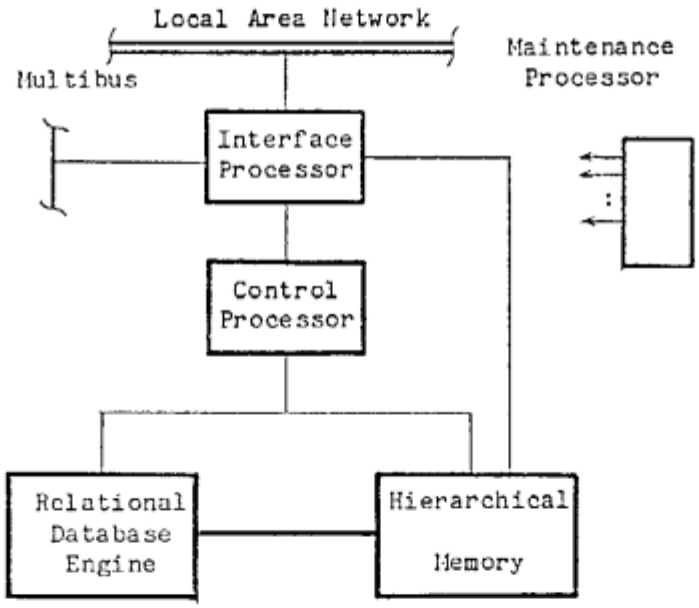


Figure 2. Delta Functional Architecture

RDBE performs basic data-manipulation operations such as selection, join, and sort. RDBE subcommands resemble relational algebra operators, except that their operands are usually attributes instead of relations. Operands of RDBE subcommands are called streams which are usually arrays of the form <TID, attribute-value, optional fields>, or sometimes arrays of combined attributes. Most subcommands

are of the form [output-stream := OPERATION (input-stream1, input-stream2, options)], where input-stream2 does not appear for one operand operation, such as selection (restriction). Input-streams are read from HM and the output-stream is written back to HM. Examples of operations are "join", "restrict", "sort", "unique", and "union". There are four RDBEs which can be used in parallel or independently.

HM is a hierarchically structured memory subsystem, which serves as Delta's system work area as well as secondary storage. Physically, it has two layers. The upper layer is a large semiconductor memory, called the database memory unit (DMU). The lower layer consists of large-capacity moving-head disk units (MHDs). Logically, HM has three layers because the DMU is divided into two areas: a buffer area and a cache area. The buffer area is used as a system work area, which stores streams (for RDBE) and sets of tuples. The cache area serves as a large disk cache to reduce access to the disks. It adopts the write-after strategy to reduce access further. HM has a battery back-up system to protect database against power failures.

HM provides the other units with a high-level interface for access to buffers and data. For instance, buffer-IDs are used instead of memory addresses for buffer access. Moreover, other units can specify the conditions of the permanent data (an attribute) to be read into a buffer from secondary storage (or cache area). HM maintains and manages the two-level attribute indices mentioned previously and uses them to find pages which contain data satisfying the given conditions. Thus, HM performs a kind of pre-screening of data to be used by other units. It also provides a low-level physical addressing interface for the storage area that contains the system directory.

A Shadow-page mechanism [Lorie77] and logs are also supported in HM to be used for transaction roll-back and data recovery.

Most database functions can be performed by a combination of RDBE and HM functions. Therefore, the main functions of the CP are to compile requests from the host into sequences of RDBE and HM subcommands, and to control their execution. The CP uses a system directory stored in HM to bind names to internal IDs. Other functions are as follows:

(1) Transaction scheduling and concurrency control

(2) Resource management

(3) Transaction commitment control and data recovery

(4) Management of dictionary and system directory.

3. Functions

Delta acts as a database server to a number of PSIs (Personal Sequential Inference Machines) in a local network (LAN) [YokotaM83] [Taguchi84]. A LAN may be a little slower than other interfaces, but it is currently the best available method of connecting a number of hosts to Delta. A faster direct interface is also available for experimentation with PSI. These experiments include interfacing the PSI logic programming language with Delta [YokotaH84]. Figure 3 shows the environment in which Delta will be used. It has been suggested that we incorporate as many database management and on-line I/O functions as possible in a special-purpose back-end computer to free the front-end general-purpose computer from database management chores [Hsiao80]. This might be true for a database backend that served a few host computers. However, Delta could prove to be a system bottle-neck if we assign it all the DBMS functions, because it will have to serve several dozen high performance PSIs. Therefore, some

high-level functions are not implemented in Delta; some of them will be incorporated in PSI's software.

Delta functions can be classified into three groups:

(1) database access functions

(2) database (access) controls

(3) system supervising

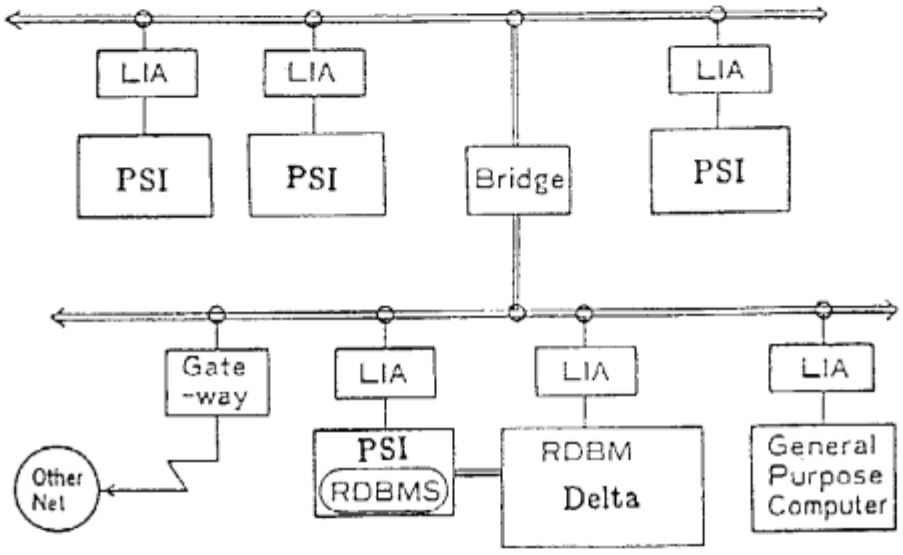


Figure 3. Total System Environment

3.1 Database Access Functions

Delta is based on the relational model and provides "relationally complete" but not "fully relational" DBMS functions according to Codd's definition [Codd82]. The relational model was selected, for (1) its compatibility with logic programming languages, (2) greater freedom in the design of internal schemas and processing algorithms made possible by its higher level of abstraction, and (3) fewer interactions with the host enabled by the set-based interface. The Delta access language is based on relational algebra; requests

expressed in user-level query languages are compiled into the Delta access language by the host. Relational algebra has the expressive power equivalent to the relational calculus, and is easier to compile into internal operations. Moreover, adding special operators such as "sort" and "set comparison" is easy in a language based on relational algebra. The Delta access language uses attribute-numbers instead of attribute-names because it is designed for use with logic programming language as the host language. Another difference between the Delta access language and relational algebra is that it allows complex conditions to be specified in conjunctive normal form in the selection operator in order to reduce the number of operators. A list of the primitives called Delta commands that are available in the Delta access language has been provided in [Shibayama84a].

Delta provides the following access functions:

(1) Relationally complete access to relations

(2) Aggregate functions

(3) Data definition and updating

(4) Arithmetic operations for updating and in retrieval conditions

(5) Special operators, such as "sort" and "set comparison"

(6) Support of null values

On the other hand, the following functions are not supported in the current system:

(1) Views

(2) Predefined requests

(3) Least-fixed-point operations

Views are not included to reduce the load on Delta. Predefined requests are not supported because the anticipated frequency of repetitive requests does not seem to justify implementation of this

function. Least-fixed-point operation are very desirable in a logic programming environment because they reduce the number of interactions required; we may later include this function in Delta [YokotaH84].

3.2 Database Controls

Delta supports the following database (access) control functions:

(1) Support of the (atomic) transaction concept

(2) Transaction concurrency control

(3) Data recovery

(4) Data dictionary

(5) Special functions for database administrators (DBAs).

The transaction concept is essential to every DBM and DBMS. The host can direct Delta to commit or abort a transaction at any time during its progress. Although a host may issue multiple update requests in a transaction, the result is treated as if the transaction were a single (atomic) operation.

Because Delta has to serve many PSIs simultaneously, it should be able to process concurrent transactions. Delta automatically locks and unlocks necessary relations so as to preserve their consistency. The granularity of locks is a relation, although finer granularity may be necessary for high throughput. It is fairly difficult to design concurrency control algorithm using finer granularity for DBMs such as Delta that have a functionally distributed architecture and set-oriented basic operations, because finer granularity must also be supported by recovery algorithm. A combination of deferred update and short-term locking methods is applied to the data dictionary so as to improve performance of transactions that involve dictionary update.

Data recovery is another function essential to every DBM and DBMS in case of system or media failure. Delta rolls-back all transactions in progress after a major failure. All transactions committed, or in the process of being committed, are preserved. Moreover, back-up dumps and magnetic tape logs can be used to restore database, if roll-back processing fails due to destruction of magnetic disk contents. In this case, roll-back processing is performed after restore processing.

The data dictionary consists of schema information for database relations. It is defined as a set of special relations whose consistency with user-defined relations is preserved by Delta so that hosts can look them up just as they can normal relations. A host may access the dictionary each time it accesses a relation to determine its schema. It may temporarily preserve the current contents of the dictionary in order to reduce interactions with Delta and improve overall performance. Thus, there could be schema information for the same relation in both the host and Delta, and we have to make sure this information is consistent. Beside the concurrency control mentioned above, Delta attaches a kind of time-stamp to schema information to indicate the latest update; this time-stamp is checked each time a host accesses a relation.

There are several special functions available for the convenience of database administrators. The bulk loading and unloading facility using magnetic tapes is useful for exchanging data with other systems or efficiently inputting large amounts of data. This facility can also be used to restructure the database so as to improve performance. Other examples of functions are the commands for storage management.

Delta does not currently have some functions that may be found in other systems:

(1) Security management

(2) Integrity control

(3) Support for the database schema design

It was decided to incorporate security management in hosts because this function should be unified with the security functions of PSI's operating system, although it can be implemented easily on either side of the system. However, part of this function may be supported by Delta when the overall strategy for system security management is decided upon.

The other two functions were excluded from Delta, since there exist no fully-developed matured methods of implementing them. Some ICOT researchers are studying integrity control mechanisms for logic programming environment, and their methods will be applied to the PSI software that provides the interface to Delta.

3.3 System Supervising

In conventional computer systems, the system supervising functions are provided by the operating system rather than by the DBMS. If a DBM is a back-end, tightly-coupled with its host, some of the system supervising functions may be incorporated in the host's operating system. However, these functions must be provided in DBM if it is an independent database server such as Delta. Delta's system supervising functions include the following functions:

(1) System console (Operator or DBA interface)

(2) Control of system status

(3) Status report to host upon request

(4) Modifying parameters (size of the buffer area, etc.)

(5) Maintenance of log tapes, etc.

(6) Failure detection and system reconfiguration (disconnecting faulty equipment, etc.)

(7) Diagnosing faulty equipment

(8) Collection of statistical data for evaluation.

Most of these functions were designed and implemented especially for Delta, although a few made use of existing facilities.

Functions for registering users and maintaining their records are not supported, because Delta does not support security by itself. An accounting function is probably necessary for some DBMs; Delta does not need one because it is to be used in a closed research environment.

4. Basic Processing Algorithms

Delta processing algorithms are described in this chapter. The sequence of request processing is as follows. IP receives a request from a host and passes it to the CP. The CP compiles the request into a sequence of internal operations. A compiled request consists mainly of RDBE and HM subcommands. Then, CP controls the execution of the compiled request by RDBE and HM. The path between IP and HM is used to transfer tuples to and from host; the CP does not process them directly.

4.1 Retrieval

Processing algorithms are best explained by some examples. For readability, host requests are written in an SQL-like syntax instead of in the Delta access language. Some trivial operations are omitted.

Example 1: simple selection

```
Request: Select A,B
         from   R
         where  B>"10" and B<= "20"

Compiled request: (modified for readability)
 1:    HM:    Temp1 := R.B where (range ["10"<, <="20"])
 2:    HM:    allocate Temp2
*3:    RDBE:  Temp2 := restrict (Temp1, range ["10"<,<="20"])
 4:    HM:    allocate Temp3
 5:    RDBE:  Temp3 := sort_by_tid (Temp2)
 6:    HM:    Temp4 := R.A where (tid in Temp3)
 7:    HM:    allocate Temp5
 8:    RDBE:  Temp5 := restrict (Temp4, equal_tid [Temp3])
                /* this performs tid-join */
 9:    HM:    allocate Temp6
10:    RDBE:  Temp6 := sort_by_tid (Temp5)
11:    HM:    allocate Temp7
12:    HM:    Temp7 := transpose_to_tuple (Temp6, Temp3)
          /* result in Temp7 */
13:    HM:    release Temp1, Temp2, Temp3, Temp4, Temp5, Temp6
```

* Data transfers between RDBE and HM are directed by RDBE. For instance, RDBE issues two HM subcommands, start_stream_in (Temp1) and start_stream_out (Temp2), when it executes step 3. These subcommands do not appear

in the compiled sequence, because they are automatically issued by RDBE and CP is not involved.

In step 1, HM gets items of attribute B that may satisfy the condition "10"<B<="20" using clustering indices. Then, RDBE extracts only those items that actually satisfy the condition (step 3). Next, HM gets those items of attribute A that may correspond to the same tuples as the result of the previous operations (step 6), and RDBE extracts the exact items (step 8). This operation is called a tid-join because it resembles to a join. At this stage, the result is obtained, but it is still split between two buffers although both are sorted by tids. Thus, the final step is conversion to the tuple format.

In this example, steps 2, 4, 7, 9, 11 simply involve just the buffer allocation. Different buffers are used for different data for simplicity in this example, but buffers may be reused in actual operations. If they are reused, some allocation steps shown here are not necessary. All buffers except the buffer containing the result are released at or before the last step. Allocation and release of buffers are not shown in the other examples because they are trivial operations.

Steps 5 and 10 prepare for the subsequent steps by sorting data. They may be skipped if the result of previous operation does not exceed a specific limit (64KB or 4K items), because they are already sorted in previous steps or can be automatically sorted at the next steps in such cases.

In Delta operations, projections do not usually appear explicitly in internal operations. There may be more than two attributes in relation R, but the internal operation is the same as above. Thus, those attributes which are not output and are not conditions are not accessed in Delta.

Example 2: semi-join

```
Request: select A, B, C
         from   R
         where  C in
                (select C
                 from   S
                 where  D = "d")
```

Compiled request:

```
1:   HM:      Temp1 := S.D where (equal "d")
2:   RDBE:    Temp2 := restrict (Temp1, equal ["d"])
3:   HM:      Temp3 := S.C where (tid in Temp2)
4:   RDBE:    Temp4 := restrict (Temp3, equal_tid [Temp2])
5:   HM:      Temp5 := R.C where (equal Temp4)
6:   RDBE:    Temp6 := restrict (Temp5, equal [Temp4])
                  /*  this performs semi-join */
7:   HM:      Temp7 := R.A where (tid in Temp6)
8:   HM:      Temp8 := R.B where (tid in Temp6)
9:   RDBE:    Temp9 := restrict (Temp7, equal_tid [Temp6])
10:  RDBE:    Temp10 := restrict (Temp8, equal_tid [Temp6])
11:  HM:      Temp11 := transpose_to_tuple (Temp9, Temp10, Temp6)
```

In this example, several trivial operations (allocate, release, and sort) are not shown. The semi-join can be performed by RDBE as a restriction, with the condition being given in a stream. Note that semi-join and tid-join are done by almost identical operations in Delta.

Example 3: join

```
Request: select R.A, R.B, S.C
         from   R, S
         where  R.A = S.A
         and    S.D = "d"

Compiled request:
 1:   HM:   Temp1 := S.D where (equal "d")
 2:   RDBE: Temp2 := restrict (Temp1, equal "d")
 3:   HM:   Temp3 := S.A where (tid in Temp2)
 4:   RDBE: Temp4 := restrict (Temp3, equal_tid [Tepm2])
 5:   HM:   Temp5 := R.A where (equal Temp4)
 6:   RDBE: Temp6 := join (Temp4, Temp5, equal)
 7:   RDBE: Temp7 := join (Temp5, Temp6, equal_tid)
           /* tid-join for result relation */
 8:   HM:   Temp8 := R.B where (tid in Temp6)
 9:   HM:   Temp9 := S.C where (tid in Temp6)
10:   RDBE: Temp10 := join (Temp8, Temp6, equal_tid)
11:   RDBE: Temp11 := join (Temp9, Temp6, equal_tid)
12:   HM:   Tepm12 := transpose_to_tuple (Temp7, Temp10, Temp11)
```

The result of the join (step 6) is an array of triplets (new_tid, R's_tid, S's_tid) and does not include the values of R.A or S.A because of hardware limitations. Thus, [new_tid, R.A] := join ([R's_tid, R.A], [new_tid, R's_tid, S's_tid], equal R's tid) must be

done at step 7. Steps 10 and 11 are tid-joins, but are a little different from examples 1 and 2, because new tids are attached.

4.2 Updating

There are three kinds of update operations in Delta: insert, delete and update. An update is basically performed by a combination of delete and insert of necessary attributes. The example below shows the basic of update operations.

Example 4: Updating

```
Request: update R
         set    A = A + "a"
         where  B = "b"

Compiled request:
  1:     HM:   Temp1 := R.B where (equal "b")
  2:     RDBE: Temp2 := restrict (Temp1, equal "b")
  3:     HM:   Temp3 := R.A where (tid in Temp2)
  4:     RDBE: Temp4 := restrict (Temp3, Temp2, equal_tid)
  5:     RDBE: Temp5 := add (Temp4, "a")
  6:     RDBE: Temp6 := delete (Temp3, Temp2, equal tid)
           /* Temp6 := Temp3 - Temp2 */
  7:     HM:   R.A := update pages corresponding to Temp3 by Temp6
           /* R.A := R.A - Temp2 */
  8:     HM:   R.A := insert (R.A, Temp5)
```

Selection of the qualified tuples is performed from step 1 to step 4. Calculation of new values is done in step 5 by RDBE. Actual updating takes place after these preparations. In steps 6 and 7, old value items of attribute A are deleted from the database. New value

items are inserted in step 8.

When items are deleted from an attribute table, they are actually erased from the pages of the table. However, the items are inserted in the overflow pages. Thus, newly inserted items are not indexed and are read as possible candidates for whatever conditions are specified for subsequent retrievals. Indices must be restructured if the retrieval performance is degraded by these update operations. This restructuring is called reclustering; it is performed automatically by special system transactions whenever a specific trigger condition is satisfied.

5. Implementation

Delta consists of an RDBM Supervisory and Processing (RSP) subsystem, which takes charge of overall control, monitor and operation processing, and a Hierarchical Memory (HM) subsystem, which is responsible for storage, retrieval, and updating of relation data. Each functional unit identified in chapter 3 is being implemented by a separate piece of hardware, as shown in Figure 4. Delta is s loosely-coupled, functionally-distributed multiprocessor. The RSP and the HM are connected through a total of 11 channel interfaces. Figure 5 shows the specifications for the Delta systems.

5.1 RSP Subsystem Configuration

The RSP subsystem consists of the CP (Control Processor), IP (Interface Processor), MP (Maintenance Processor), and RE (Relational Database Engine), each of which consists of hardware and software.

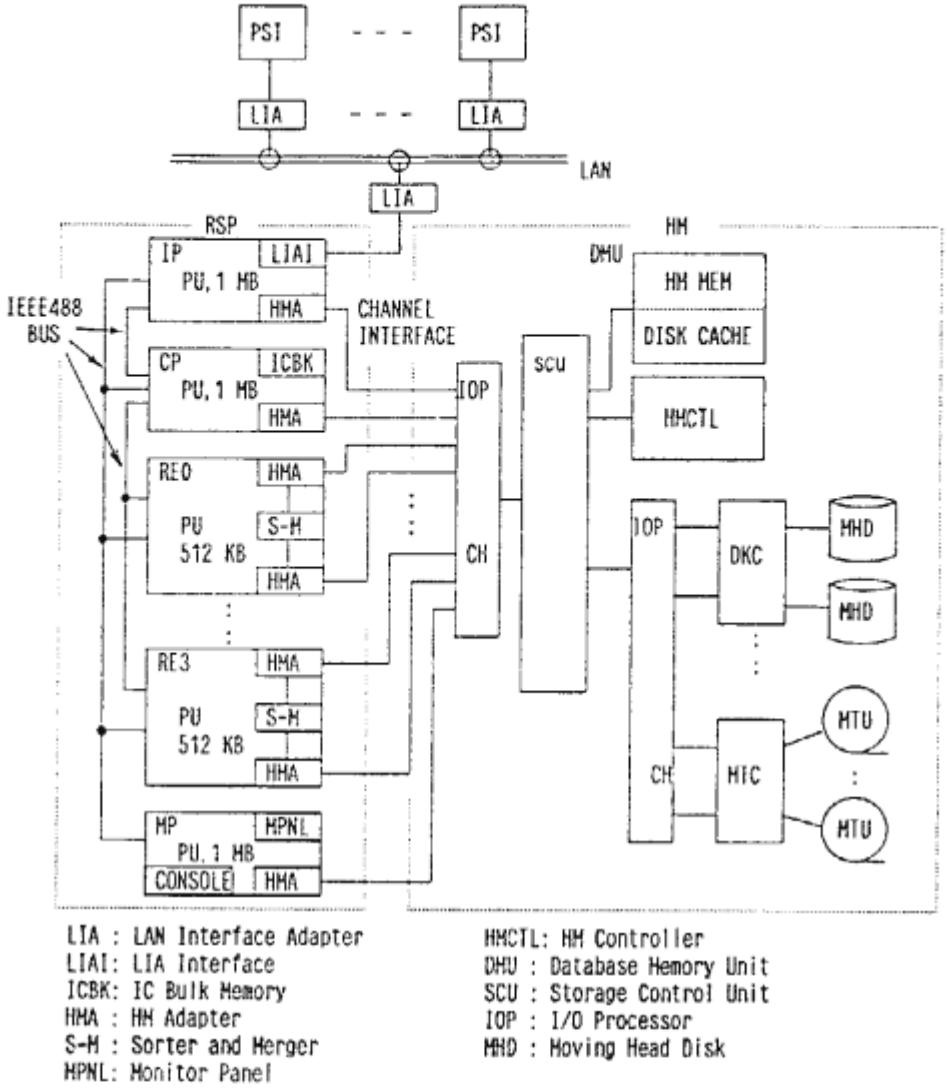


Figure 4. Delta Hardware Configuration

	Configuration of May, 1984	Final configuration
(1) RSP subsystem		
Control Processor	1	1
Relational DB Engine	1	4
Interface Processor	1	1
Maintenance Processor	1	1
(2) MM subsystem		
MM Controller	1	1
DB Memory Unit	16 MB	128 MB
Moving Head Disk	2.5 GB ×2units	2.5 GB ×8units
Magnetic Tape Unit	2	4

Figure 5. Delta system specifications

5.1.1 RSP hardware configuration

Each unit basically consists of a 16-bit processor with a main memory size either 512 KB or 1 MB. The RE is provided with dedicated hardware for sorting and merging, and the CP is provided with a 15 MB semiconductor disk storage for increasing memory capacity. CP's control program temporarily stores the directory and other tables in the semiconductor disk storage. The MP is connected to the Delta system status monitoring display, the desk console, and the MTU for collecting RSP log information. Each RSP unit is connected to the others via three IEEE 488 buses. Each CP, IP, and MP unit is provided with one HM adapter (HMA), which serves as interface hardware for the HM. The RE is provided with an HMA for input and another for output.

(1) Relational database engine

The RE comprises of a sorter and a merger, which perform relational operations for pre-sorted results. Although various kinds of sorting algorithms have been studied [Knuth73], a two-way merge-sort algorithm has been adopted for the sort, a major component of the RE.

(a) Two-way merge-sort algorithm

This algorithm rearranges input record values in either ascending or descending order. An array of input records is regarded as a collection of sorted arrays, each of which has a length of 1. Sorted arrays are merged in pairs so that the length of each sorted array is doubled at each sort stage. When this operation is carried out in the pipeline processing configuration, the use of the $\log_2 n$-stage sorter enables a sort output with a cycle time of $2n+(\log_2 n)-1$ [Todd78] (n=number of

records).

(b) Configuration of the RE

The RE consists of a IN module, a sorter consisting of 12-stage sorting cells, a merger, two HMAs, and the processor and RE control programs, which controls the above units. Figure 6 shows the configuration of the RE.

RE operations, based on relational algebra, sort item groups consisting of attribute values and TIDs while transferring them as a stream, perform relational operations from the head of sort output, and output combinations of items based on the results of the operation [Sakai84].

The IN module transforms input stream data item into an internal format suitable for the sorter and merger : field ordering, which replaces the head of each item to be sorted, and data type transformation. The sorting cell is made of two FIFO buffers, a comparator, and a multiplexer for merging. Each FIFO buffer has a capacity of 16 bytes at the first stage and 32K bytes at the twelfth stage.

The merger consists of an operation section, and an output control section. The operation section comprises of a comparator and two 64 KB memories having FIFO function, and performs relational algebra operations by comparing two sorted stream data. The output control section comprises of two 16 KB buffers, two field-ordering, field-selection and data-type transformation circuits, a selector and an output sequence controller. This section performs field reordering and selecting of an output data item, and adding a new TID to it, and then transferring output data items to the HM via an HMA.

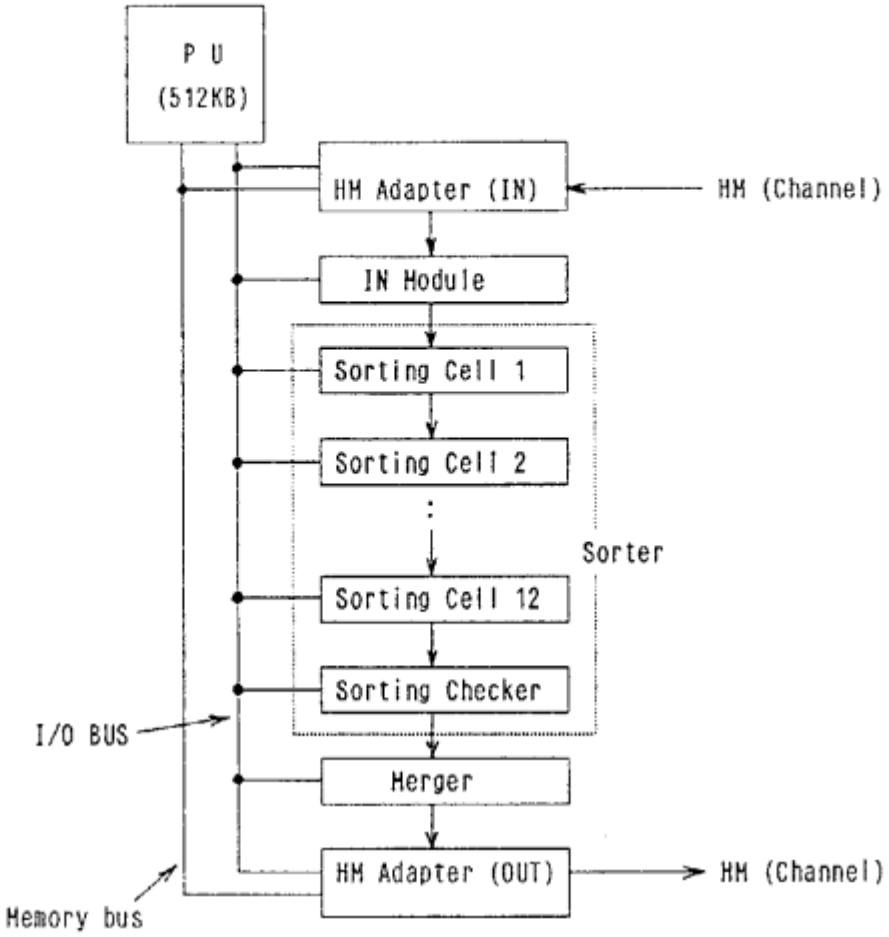


Figure 6. RE Configuration

5.1.2 RSP Software Configuration

Some pieces of software, such as the operating system and the IEEE 488 bus driver, are common to IP, CP, and MP. The IEEE 488 bus driver is used for inter-RSP subsystem communications and also used for the physical level interface with the LAN Interface Adapter (LIA).

The following is a list of software functions assigned to the RSP units.

(1) IP

(a) LAN interface control

(b) Delta command and data extraction from packet-typed data

(c) Parallel reception of command tree

A command tree consists of one or more Delta commands, and is the unit in which any sequence of meaningful processing is carried out. This software identifies and controls a command tree for each transaction in the received packet data array.

(d) Data format conversion

When data is input from the host, it is provided each tuple with a tuple identifier (TID). The TID is removed when data is sent to the host.

(e) Data transfer with the HM

After the CP completes an instruction execution that reserves a buffer for data transfer between the IP and the HM, the IP performs data transfer with the HM.

(2) CP

(a) Transaction management

The CP is responsible for transaction execution management, Delta command analysis, generation and execution of HM, RE and IP subcommands, and Delta resource management.

(b) Dictionary/directory management

Dictionary is used for the host user reference and directory is used for the Delta command analysis, subcommand generation and concurrency control. Dictionary includes two meta relations, " Relations" relation and "Attributes" relations, but directory does not include the unnecessary part of dictionary information. Dictionary is stored in HM based on attribute-based schema, but directory is stored in HM in special data structures for the efficiency in look-up. The

semiconductor disk storage in the CP stores the directory for repeated use. If necessary, the CP obtains directory pages from the HM by directory-access subcommands.

(c) Concurrency control

Concurrency is controlled by locking the relations based on the two-phase locking method. Relations to be locked during a transaction can be explicitly locked using a start-transaction command, or they can be automatically locked before command-tree execution. All the relations are unlocked at the end of a transaction whether it is a normal or an abnormal termination.

(d) Recovery management

When an abort-transaction command is received from the host during command-tree execution or hardware failure in Delta and media failure are detected, relations are restored in a status before the transaction has been started.

(3) RE

(a) RE control

The RE software analyses subcommands from the CP, controls the sequential operations of RE hardware and controls I/O data transfer with the HM.

(b) Relational operation support

The RE software performs arithmetic operations that cannot be processed by the merger hardware.

(4) MP

(a) Delta system status monitoring, and system configuration control

(b) Delta system start-up and shut-down

(c) Database loading and unloading

(d) Operator command and message management

(e) Statistical information collection

5.2 HM Subsystem Configuration

The HM subsystem consists of HM hardware and software designed to efficiently manage the storing of large amount of data to and the retrieving of data from storage area, as instructed by the RSP subsystem.

5.2.1 HM hardware configuration

In the final configuration, the HM hardware comprises of a non-volatile high speed Database Memory Unit (DMU) having a memory capacity of 128 MB, magnetic disk units having a maximum capacity of 20 GB, disk controllers, a 32-bit processor (named HM controller, HMCTL), which controls HM execution, and magnetic tape subsystem for system logging and database loading and unloading.

5.2.2 HM software configuration

HM software functions are listed below.

(1) HM subcommand processing

The following processing is performed by HM subcommands specified by the RSP.

(a) Attribute definition and operation

This processing generates and deletes attribute definitions, transposes tuples to attributes, and vice versa.

(b) Update processing

This processing inserts, deletes, and updates attributes.

(c) Clustering operation

Clustering tables are generated and updated in accordance with the value of attributes and TIDs by HM subcommands instructed from the CP.

(d) Data transfer management

Stream data transfers between the HM and the RE, and packed data between the HM and the IP or CP.

(e) RSP buffer management

RSP buffer inside the HM is reserved and released by HM subcommands.

(2) Memory and MHD space management

The data transfer of attributes and directory information is performed between secondary storage and the DMU, and memory resource is managed for storing such information.

(3) Data recovery processing

Data recovery of attributes information is performed by recovery subcommands from the CP.

Delta can be classified as a hardware-oriented DBM because it incorporates a new piece of hardware in its kernel, and its processing algorithms for primitive relational operations are totally different from those of conventional DBMSs. However, some basic operations and additional functions are implemented in software. We expect to learn more about which functions to implement in hardware by evaluating Delta in actual operation.

6. A Delta Performance Estimation

We now consider an example selection query in SQL form for a Delta performance estimation :

```
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 10 Bytes for each attribute here.

The execution time characteristic for selectivity according to an estimation is shown in Figure 7. This figure assumes a high hit ratio of disk cache in DMU. For certain range of selectivity, there are different dominating factors. For the selectivity range between 0.01 % and 1 %, the dominating factor is the increasing TID join time. Buffer preperation in DMU is not so time-consuming, because the intra-buffer data transfer in DMU 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 contents of the attributes should be scannd for TID join in this area. For the selectivity range between 10 % and 100 %, along with the TID join in this time which forms the plateau, tuple reconstraction time becomes influential. The effect of tuple reconstruction rapidly becomes great.

This result, though still not sufficiently quantative, indicates that the incorporation of a large capacity DMU is effective for a high performance database machine. The effort to increase the hit ratio by the wise replacemnt algorithm is the key to effectively utilize the DMU.

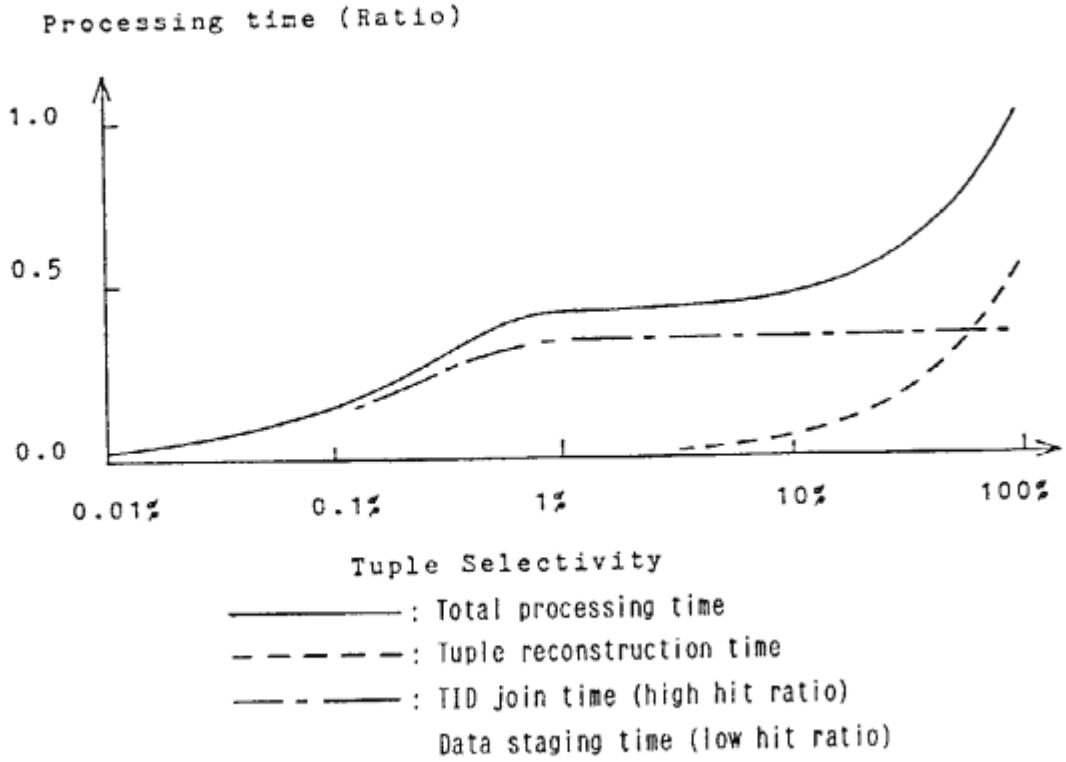


Figure 7. Estimated Delta Performance Characteristics

7. Conclusions

We have presented an overview of relational database machine Delta. Part of its hardware, along with 5 GB of storage, was installed at ICOT in April 1984, and has been used to debug and test the control software. The design of most of the control software has been finished and it is currently being tested. The remaining hardware will be installed in December 1984, and we expect Delta to be fully operational by March 1985.

The functions described here and the performance of the machine will be tested in ICOT. Our goal is not just to build a good database machine, but to pave the way for future knowledge base machines. We hope these tests will prove our most crucial decision correct: that developing an RDBM is the first logical step toward a true knowledge

base machine.

Acknowledgements

The authors express their appreciation to the Toshiba and Hitachi researchers and engineers who have participated in the development of Delta.

REFERENCES

[Bancilhon82] F. Bancilhon et al., VERSO : A Relational Back-End Data Base Machine, Proc. of Int'l Workshop on Database Machines, Aug. 1982.

[Chikayama83] Takashi Chikayama et al., Fifth Generation Kernel Language, Proc. of the Logic Programming Conference'83, Mar. 1983, Tokyo Japan.

[Codd82] E. F. Codd, Relational Database: A Practical Foundation for Productivity, Comm. of ACM, Feb. 1982.

[Hsiao80] David K. Hsiao, Data Base Computers, in (ed.) M.C. Yovits, Advances in Computers, Vol. 19, Academic Press 1980.

[Hsiao83] David K. Hsiao (ed.), Proc. of Int'l Workshop on Database Machines, Aug. 1982, and also revised version : Advanced Database Machine Architecture, Prentice-Hall, 1983.

[IEEE79] Special Issue on Database Machines, IEEE Transactions of Computers, Vol. c-28, June 1979.

[Knuth73] D. E. Knuth et al., Sorting and searching, The Art of Computer Programming, vol.3, Addison-Wesley Publishing Co., 1973.

[Lorie77] Raymond A. Lorie, Physical Integrity in a Large Segmented Database, ACM TODS, 2-1, Mar. 1977.

[Murakami83] Kunio Murakami, et al., A Relational Data Base Machine: First Step to Knowledge Base Machine, Proc. of 10th Symposium on

Computer Architecture, Stockholm, Sweden, June 1983.

[Sakai84] Hiroshi Sakai, et al., Design and Implementation of the Relational Database Engine, Proc. of Int'l Conf. on Fifth Generation Computer Systems 1984, Nov. 1984, Tokyo Japan. (to appear)

[Schweppe82] H. Schweppe et al., RDBM - A Dedicated Multiprocessor System for Data Base Management, Proc. of Int'l Workshop on Database Machines, Aug. 1982.

[Shibayama84a] Shigeki Shibayama, et al, A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor, New Generation Computing, Vol.2, No. 2, June 1984, Ohmsha, Ltd. and Springer-Verlag.

[Shibayama84b] Shigeki Shibayama, et al., Query Processing Flow on RDBM Delta's Functionally-Distributed Architecture, Proc. of Int'l Conf. on Fifth Generation Computer Systems 1984, Nov. 1984, Tokyo, Japan. (to appear)

[Taguchi84] Akihito Taguchi, et al., INI: Internal Network in ICOT and its Future, Proc. of ICCC, Australia, Oct. 1984. (to appear)

[Todd78] S. Todd, Algorithm and Hardware for a Merge Sort Using Multiple Processor, IBM J. of Research and Development, Vol.22, No.5, Sep. 1978.

[YokotaH84a] Haruo Yokota, et al., An Enhanced Inference Mechanism for Generating Relational Algebra Queries, Proc. of 3rd ACM Symposium on Principles of Database Systems, Apr. 1984, Waterloo, Canada.

[YokotaM83] Minoru Yokota, et al., The Design and Implementation of a Personal Sequential Inference Machine: PSI, New Generation Computing, Vol. 1, No. 2, 1983, Ohmsha, Ltd. and Springer-Verlag.