ICOT Technical Report: TR-066

TR-066

Design and Implementation of a Two-Way Merge-Sorter and its Application to Relational Database Processing

by

Kazuhide Iwata, Shigeo Kamiya, Hiroshi Sakai,
Susumu Matsuda (Toshiba Corporation)
Shigeki Shibayama and Kunio Murakami

May, 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

Design and Implementation of a Two-Way Merge-Sorter and its Application to Relational Database Processing

KAZUHIDE IWATA, SHIGEKI SHIBAYAMA, SHIGEO KAMIYA

HIROSHI SAKAI, SUSUMU MATSUDA and KUNIO MURAKAMI

Abstract

A new hardware sorting unit having both internal and external sort capabilities is described. The main elements of the unit consists of an IN module, sorting cells, a sorting checker, a merger, a data input adapter and a data output adapter. A general-purpose CPU controls the hardware components. The sorter is responsible for internal sorting and the merger is responsible for external sorting. The sorting algorithm is a straight two-way merge-sort. The sorting hardware is accommodated in a database machine for relational database processing. The merger is also provided with relational database operation facilities, for example, join operation commands. The CPU also controls complex relational database operations. The decisions made in the design phase focused on the practical aspects of use in a relational database environment. The performance estimation indicates that the hardware can sort 4K 16-byte tuples in 56 milliseconds, including data transfer time.

Index terms:
two-way merge-sorter, internal sort, external sort, merging, relational database machine, join operation, set operation, pipeline processing

1. INTRODUCTION

Sorting is an essential operation with a great deal of practical significance in data processing. In particular, it is one of the most important operations in database processing. Various kinds of sorting algorithms have been developed from the beginning of computer science research and some of them, for example, quick and heap sorts have practical applications in conventional computer systems [1]. However, as sorting is a time-consuming operation and imposes a heavy burden on conventional computer systems, the intensive research has also been done on hardware implementations of various sorting algorithms [2]-[6]. This research is summarized briefly below.

K. Iwata, S. Kamiya and H. Sakai are with the Research and Development Center, Toshiba Corporation, Kawasaki, Japan.

S. Matsuda is with the Ome Works, Toshiba Corporation, Ome, Japan.

S. Shibayama and K. Murakami are with the Research Center, the Institute for New Generation Computer Technology (ICOT), Tokyo, Japan.

This work is part of a major research and development project on Fifth Generation Computer Systems (FGCS), conducted under a program set up by Japan's Ministry of International Trade and Industry.

Winslow and Chow [7] have classified parallel sorting architectures into five categories based on the nature of the data input and output and compared existing sorting hardware designs in order to design new sorting circuits. Thompson [8] has analyzed the area-time complexity of thirteen sorting circuits using an updated model of VLSI implementation. However, some of the sorting circuits described in these literatures aimed at the theoretical verification of sorting algorithms and are, in some respects, impractical for data processing using currently available technology.

Given these circumstances, we designed a hardware sorter for research on relational database processing, rather than on hardware sorting algorithms. It is based on the straight two-way merge-sort algorithm analyzed by Todd [9] and is implemented using currently available technology. This implementation can be classified as a sequential input/sequential output sorter (SISO) according to the five categories by Winslow and Chow [7]. Our sorter is designed and implemented as part of Delta*1, a backend relational database machine. The following list summarizes the properties of this sorter as implemented for use in the database machine environment.

(1) The sorter consists of a linear array of 12 processing elements, called the sorting cell, and one processing element, called the sorting checker; these arrange input data elements so that they are in a specified linear order (ascending or descending). The sorting operation is efficiently performed by pipeline processing synchronized with the data transfer rate of 3MB/sec.
(2) The sorter processes only absolute values represented in standard binary notation. However, the pre-processing module, called the IN module, connected to the input section of the sorter, has the function of transforming some data types into absolute values on-the-fly.
(3) The sorter processes the entire tuples as well as the key fields, in order to perform set operations, such as intersection, and relational algebra operations, such as join.
(4) The sorter performs stable sort operations on equal values, i.e. it maintains the original relative order of the input sequence of tuples having the same values. This property is useful for the group-by operation, which is often used in database processing.
(5) The sorter processes null values by recognizing the tag fields.
(6) The sorter provides a function for detecting duplicate values. This capability enables the manipulation of duplicate values in the hardware implementation of our proposed relational database processing algorithm.
(7) The sorter provides a function for checking sort results. This function increases reliability of the sorter.
(8) The sorting cell has two operation modes: the sort mode and the pass mode. The former merges two sorted sequences of tuples into one. The latter does not merge, but transfers input data directly to the next cell.

These properties represent important factors in the design of a practical sorter and are discussed in detail in later sections. In Section 2, we present an overview of Delta and its key component, RDBE (relational database engine), and summarize their main functions to

---

*1 Delta is under development at ICOT Research Center (Institute for New Generation Computer Technology supported by Japan's Fifth Generation Computer Project) [10].

clarify the background of the sorter design. Section 3 describes design considerations for the sorter. Section 4 describes sorter design and implementation. Applications and performance estimation are described in Sections 5 and 6, respectively. Section 7 is the conclusion.

2. BACKGROUND

The relational data model is useful not only for conventional database systems, but for logic database systems as well [11], [12]. At ICOT Research Center, from the latter point of view, research on the connection of an inference machine to a relational database machine is being conducted for the purpose of developing a knowledge base machine. In this section, we outline the architecture of Delta and its key component, RDBE, in order to provide some background on the sorter design.

A. Overview of Delta Architecture

Delta's global architecture is shown in Fig. 1. In this figure, the dotted line shows the final configuration. Delta consists of the following components:

(1) An interface processor (IP), which connects Delta to a local area network (LAN) and the Multibus.
(2) A control processor (CP), which provides database management functions, such as concurrency control and database recovery.
(3) A relational database engine (RDBE), which is the key component for processing relational database operations in Delta. The RDBE is implemented by the combination of a general-purpose processor with a specialized processor. The sorter is accommodated in the specialized processor for performing high-speed relational database operations.
(4) A maintenance processor (MP), which provides functions that enhance Delta's reliability and serviceability.
(5) A hierarchical memory (HM), which provides functions for storing, accessing, clustering and maintaining relations. The HM is implemented using a general-purpose processor as a controller, a high volume of semiconductor memory and large-capacity moving head disks. HM is connected to other components through high-speed channels. HM can be seen as a large semiconductor buffer with a very short latency time and a fast transfer speed from the RDBE.

The general Delta command processing sequence is as follows.

The IP receives relational algebra level commands, called Delta commands, from a host connected to the LAN, and sends them to CP. The CP translates these commands into a sequence of internal subcommands, which are then issued to RDBE and HM to make them cooperatively perform the specified database operation. After execution of the Delta commands the IP transfers the result stored in HM to the host via the LAN.

B. Overview of the RDBE Architecture

The RDBE configuration is shown in Fig. 2. It is designed to achieve high-speed relational database processing by means of a pipelined sorting and merging. The RDBE is implemented using the following modules:

(1) A general-purpose CPU, which is used as the RDBE controller.
(2) Two HM adapters, which serve as interfaces between RDBE and HM.
(3) The IN module, which transforms the format of input data into an internal format suitable for the sorter and merger modules. Among these transformations are:

* Field ordering, which shifts a key field to the head of the tuple
* data type transformation
* generation of null value bit signals.

(4) A sorter, which generates sorted tuples. This module is described in detail later.
(5) A merger, which performs external merge-sorting and relational database operations using a processing algorithm based on a merge operation. This module is also described in detail later.
(6) Two input/output controllers (IOCs), which control the internal bus connection to CP and MP in Delta.

In Fig. 2, DT, PT, NL and DP stand for data lines, parity lines, a null line and a duplication line, respectively. The null line is used to denote that there is a tuple with a null value key on the data lines. The duplication line is used to denote that there is a tuple having the same key value as the next on the data lines.

These modules are controlled to run simultaneously in such a way that pipeline processing is synchronized with the data transfer rate.

The main data path is from the HM adapter (IN) to the HM adapter (OUT) through the engine core indicated by the dotted line in Fig. 2. If an RDBE operation involves two relations, as in a join operation, the relation is first stored in the engine core from the HM adapter (IN). Then, the engine core compares the tuples of the relation from the HM adapter (IN) with the stored relation, and outputs the result to HM via the HM adapter (OUT).

If the CPU itself is required to manipulate the data, the result from the engine core is sent to the CPU's main memory via the HM adapter (OUT). After the CPU has finished the manipulation, the final result is sent to HM via the HM adapter (OUT).

RDBE offers various kinds of operations necessary for relational database processing. They are classified into the following categories:

(a) Relational algebra operations, such as join, projection and selection

(b) Sort operations in both ascending and descending order

(c) Set operations, such as intersection, union and difference

(d) Arithmetic operations

(e) Aggregate operations over an entire relation and also over a nonintersecting partition of a relation

(f) Miscellaneous operations specific to the way in which Delta manages data.

These operations are performed cooperatively by the engine core and the CPU software to achieve high cost-effective database processing.

3. DESIGN CONSIDERATIONS

A. Incorporation of Sorting into Relational Database Processing

Most of the latest research on sorter design is focused on algorithms for high-speed sorting of large volumes of data. From the point of view of current technology, such designs require very difficult modifications of the software and the memory system. For these reasons, there have been very few implementations of relational database machines equipped with sorters until now [13]. Therefore, we have designed a practical sorter for application to relational database processing. The central concept of our design is to unite the sorting operation and the relational database operations in order to decrease the time required to transfer sorted data between the sorter and the memory system. For example, in the join operation, which is an important and time-consuming aspect of relational database processing, one of the two target relations is usually limited by a restrict operation and becomes a scaled-down relation. The join operation with the other full-scale relation is performed using the sorter's capability of separately processing the fragments of the full-scale relation. To implement this idea, we connected the merger to the output section of the sorter. The merger is a new hardware unit that performs relational algebra operations, as well as external merge-sorting, on data processed by the sorter. The merger used a new processing algorithm that takes null and duplicate values into consideration.

Based on this idea, the following problems have been taken into account in our implementation.

B. Input Data Structure

With the regard to their input data structure, sorter designs are classified into those using the key sort method and those using the tuple sort method. In the key sort method, a sequence of keys is received and a sequence of numbers (C1, C2, .., Cn) is output, indicating that the i-th key is the (Ci+1)th, the smallest (in case of ascending order) in the input key sequence [6]. In the tuple sort method, a sequence of tuples is received and a sorted sequence of tuples is output according to the value of the key field. Although the key sort method has the advantages of decreasing the amount of sorting hardware and the amount of data transfer, it has drawbacks, in that a key field must be selected from a tuple and tuples must be rearranged in the CPU or memory system using an output sequence of numbers. On the other hand, although the tuple sort method increases the amount of sorting hardware required, it not only lightens the load of the CPU or memory system, but can also perform set operations, such as intersection. We, therefore, adopted the tuple sort method, which processes the entire tuple.

As a result of adopting the tuple sort method, it is necessary that the key field be placed at the head of the tuple, so that the sorter can perform a pipelined two-way merge-sort from the head to the tail of the tuple. In our sorter, the key field must be positioned at the head of the tuple, because other fields must not be sent to the next cell until the key comparison is complete. The value of the key field is the only relevant component for sorting. We decided to order the fields of the tuple in a pre-processing module, called the IN module, connected to the input section of the sorter. Fig. 6(a) shows the scheme for field-ordering of the tuple with five fields whose key field is B. We decided to rotate the fields for ease of ordering and later reordering. Field reordering, if necessary, is performed in the merger connected to the output section of the sorter.

C. Treatment of Null and Duplicate Values

The tuple has an additional field, called a tag, that indicates whether the value is null; the sorter function that checks the tag field is essential to the database processing which deals with null values. However, the addition of this function to each sorting cell increases the complexity of the cell configuration considerably. Therefore, we prepared a null bit for each two-byte "word" in the memory of each sorting cell and a null bit line (NL) between cells; these are analogous to parity bits and parity bit lines (PT). The null bit line indicates that there is a tuple with a null value key on the data lines. Using this line, each cell can easily output a sorted sequence of tuples that include null value keys. In other words, the tuples with null value keys are considered to be duplicate tuples and the original relative order of the input tuple sequence is thus maintained. In our implementation, the tuples with null value keys are placed after the tuples with normal key values, regardless of ascending or descending sorts.

The detection of duplicate values is an important function in relational database processing. There are two kinds of duplicate values; duplicate keys and duplicate tuples. Duplicate keys are used in the hardware implementation of our relational algebra processing algorithm, which is described later in detail. Duplicate tuples are used to perform set operations, such as intersection, and to remove the duplicate tuples (unique operation) that are frequently produced by the projection operation. The addition of this detection function to each sorting cell also increases the complexity of the cell configuration. Therefore, we assigned the function of duplicate value detection to a post-processing module, called sorting checker, connected to the last sorting cell. The result of this detection is sent to the merger through the duplication line (DP) and is used as a signal for controlling relational database operations.

D. Input data type

Various kinds of data types must be handled in database processing. However, it is impractical to require the sorting hardware to sort data types in their original representation. For ease of implementation, we decided that the sorting hardware would compare only absolute values represented in standard binary notation. It is, therefore, necessary to transform some data types to absolute values in order to apply the sorter to database processing. Such transformation imposes a considerable burden on the CPU if the sorter does not have this capability. We decided, therefore, that this transformation would be performed by the IN module. According to the

data type transformation function of the IN module, the sorter can process the following data types:

(1) Unsigned integers
(2) Signed integers
(3) Single precision normalized floating point numbers (IBM format)
(4) Alphabetic Characters (ASCII code)

E. Data Length

The data length for sorting varies widely, depending on the application. It is desirable to be able to handle variable-length data. It is very difficult, however, to implement this function in hardware. Therefore, we determined the specifications for data length as follows.

All tuples within a relation must be the same length and the equal number of fields. As the number of data lines between RDBE components is set at 16 (two bytes wide), the length of tuples is restricted to multiples of two bytes. The maximum tuple length is 4096 (2**12) bytes including a 2-byte tag field and the maximum number of fields is 256.

In order to satisfy the above specifications, we designed a sorter that serially processes even-byte data using a two-byte wide hardware scheme.

4. DESIGN AND IMPLEMENTATION OF THE SORTER

The straight two-way merge-sorter is characterized by the following parameters:

N: sorting cell count
M: capacity of one of the FIFO memory of the last sorting cell (in bytes)
T: common clock interval to process and transfer one byte of data

N determines the maximum internally-sortable tuple count within the limits of FIFO memory capacity. A merge-sorter having N sorting cells can sort up to 2**N tuples in one scan. M determines the length of tuples that can be sorted. If N is 12, for example, the sorter can sort 2**12 (4096) tuples. However, if M is 64K bytes, the maximum length of a tuple that can be sorted is 16 bytes (64K/4K) when a 4096-tuple sort is specified. (Longer tuples can be sorted in a sort of a smaller number of tuples by fully utilizing the FIFO memory capacity.) Therefore, the capabilities of the merge-sorter are determined by N and M.

The maximum numbers of tuples that can be sorted is

min(2N, M/L)**, where L is the length of one tuple.

The maximum tuple length that can be sorted is

M/(2[logC])**, where C is tuple count not greater than 2**N and [logC] denotes an integer not less than logC. Base-2 logarithms are used throughout this paper.

A. Algorithm

Here, we introduce some terms used throughout the paper. We call an entire sequence of tuples a stream, a sub-sequence of a stream a substream, and a sorted sequence of tuples between sorting cells a string. Each tuple consists of a key field and other fields, called satellite information or satellite attributes [1].

We have adopted an algorithm based on a straight two-way merge-sort algorithm [9]. By using this algorithm, the sorter has been provided with properties that make it applicable to relational database processing. That is to say, the sequential nature of the input and output of this sorter makes it easy to connect a pre-processing module and a post-processing module to the sorter's input and output sections, respectively, and to perform pipeline processing at the module level, as well as at the function level within each module. Module-level pipeline processing means that several operations, such as a date type transformation, sorting and relational database processing are overlapped. Function-level pipeline processing means that several operations within each module in a transfer cycle, such as memory write, memory read and comparison are overlapped.

The straight two-way merge-sort algorithm performs a sort operation by repeating the merge process that combines two sorted strings into a single sorted string. That is to say, the first sorting cell generates strings of two tuples by merging two one-tuple strings, each of which consists of a single tuple; the second sorting cell merges these strings of two tuples into strings of four tuples, and so on. The i-th sorting cell generates a sorted string of 2**i tuples.

In our sorter, each sorting cell is performed by a specialized hardware unit called a sorting cell, which consists of two first-in/first-out(FIFO) queues implemented in RAM, a comparator and a control circuit. These sorting cells are controlled to run synchronously. In general, the i-th sorting cell generates a sorted string of 2**i tuples by merging two output strings of 2**(i-1) tuples from the (i-1)th cell. Each sorting cell starts to perform the merge operation when it has received one complete string and the first two bytes of the next string from the previous cell.

As an example of an ascending-order sort operation, we consider the following input stream in which numerals and letters denote key fields and satellite information, respectively.

3E,7W,8S,0L,4G,9T,4A,2U

In sorting cell 1, the tuples of the above input stream are alternately stored in two FIFO queues to produce single-tuple strings. The strings are read from each queue and merged into the following two-tuple strings.

(3E,7W)(0L,8S)(4G,9T)(2U,4A)

In sorting cell 2, the input strings from the previous cell are alternately stored in two FIFO queues. The merging of these two-tuple strings into four-tuple stings yields

(0L,3E,7W,8S),(2U,4G,4A,9T)

Similarly, in sorting cell 3, the two four-tuple strings are merged to produce the final eight-tuple sorted string.

(0L,2U,3E,4G,4A,7W,8S,9T).

This example shows that a sorter having N sorting cells can sort 2**N tuples and that the i-th sorting cell requires two 2**(i-1)-tuple queues.

B. **Implementation Details**

The sorter configuration is shown in Fig. 3. It consists of 12 sorting cells and a sorting checker. Twelve cells form a linear array. The input section of cell 1 is connected to the IN module and the output section of the sorting checker is connected to the merger. All the cells and the checker are connected to the I/O bus of the CPU and communicate control parameters and status information. Among the control parameters are tuple length, key field length, number of tuples and operation mode.

Fig. 4 is a block diagram of the sorting cell. It consists of an L-register, a U-register, a comparator, a selector, an output register, a FIFO memory, an address control circuit, an interface controller and a cell controller. The FIFO memories are logically divided into two queues, called a U-memory and an L-memory, which are used to alternately store string tuples from the previous cell. The address control circuit provides three kinds of addresses, WAR, RAU and RAL. WAR is a write address for storing input register data into the memory; RAU and RAL are read addresses for reading data from each memory into the U-register and the L-register, respectively. The interface controller communicates control parameters and status information between the CPU and the cells. The cell controller provides six signals TLC, KLC, SMC, STC, STU and STL for controlling the execution in the cells. Each signal is a carry bit of the corresponding counters. The counters in the cell controller are loaded with the control parameters containing the length information. They are then counted down in synchrony with the 2-byte processing loop (described below) to produce the control signals as carry bits.

TLC: end of the tuple
KLC: end of the key field
SMC: end of the stream
STC: end of the input string from the previous cell.
STU: end of the string read from U-memory
STL: end of the string read from L-memory

The control structure in the sort mode of the sorting cell contains three nested loops: a 2-byte processing loop, a tuple-processing loop and a string/stream-processing loop. The 2-byte loop has three states. In the first and the second states, 2-byte data is read from the U-memory and L-memory, respectively. In the third state, data in an input register is written into the memory, the two tuples in the U- and L-registers are compared, and one of the two tuples is transferred to an output register via a selector according to the result of the comparison. This represents the desired sorting order. These states are controlled by a three-phase clock. The state-transition time is 220ns. Therefore, the repetition interval of the 2-byte processing loop is 660ns. This interval is synchronized with the data transfer rate of 3MB/sec. The tuple-processing loop controls the 2-byte processing loop by means of TLC and KLC. This loop has also three states: the comparison state, the U state and the L state. The comparison state is maintained until the sort cell determines the tuple to be sent to the next cell according to the comparison result. In the U state, the U-memory tuple is transferred to the next cell and the pointer address of the L-memory tuple in current use is reset for next comparison state. In the L state, the L-memory tuple is transferred to the next cell and the pointer address of the U-memory tuple in current use is reset.

The state-transition condition is as follows:

<comparison state to U state>
- o In ascending sort, t(U-memory) =< t(L-memory)
- o In descending sort, t(U-memory) >= t(L-memory)
- o t(L-memory) = null
- o both t(L-memory) = null and t(U-memory) = null

<comparison state to L state>
- o In ascending sort, t(U-memory) > t(L-memory)
- o In descending sort, t(U-memory) < t(L-memory)
- o t(U-memory) = null and t(L-memory) \= null

where t(U-memory) or t(L-memory) denotes the tuple at the head of each memory.

The uppermost string/stream loop controls the lower tuple-processing and 2-byte loops by detecting the string or stream end using SHC STC, STU and STL.

By using these control loops, stable sorts containing null values are achieved.

In the pass mode, the sorting cell merely forwards 2-byte data directly to the input tuples via the input register, the L-register and the output register. The time required to forward 2 bytes is 660ns, corresponding to the sort-mode 2-byte processing loop.

This mode is used when the tuple length is greater than 16 bytes or the number of tuples are small. It is not necessary to activate cells that are not required to perform the specified sort.

5. APPLICATION

Sorting is often a very effective type of preprocessing for database operations. A natural-join, for example, can be performed efficiently by merging two sorted relations. A unique operation or duplicate elimination is virtually impossible without first sorting the target tuples. If the characteristics of the manipulated attributes are known in advance, an optimization algorithm can be applied to streamline database operations. Software-backend database machines use built-in access paths to match predefined access characteristics. It is, however, a well-known fact that those software-backend database machines perform poorly when unexpected accesses occur, as in a join of two attributes that are not assumed to be join keys. If these characteristics are not known, in the worst case, every attribute has to be handled symmetrically; a different processing scheme is required to process queries in a reasonable amount of time. Database operation based on merging is considered a good algorithm for this purpose.

In an environment in which database schemata are fixed and there are only predetermined patterns of access to the databases, incorporating a special (asymmetrical) internal schema is effective. Satellite attributes, if they are known, can be stored with a some kind of pointer from their key and fetched as needed. However, to exploit the full power of the relational model, database management systems (including software DBMS and database machines) require more flexible attribute-handling capabilities. For example, in information systems such as decision-support systems, queries tend to be of a conversational nature. Such queries are not usually made by programmers; often the user is an executive. Therefore, it must be easy to select, join, and group attributes, for instance. This can be done by providing indices to every attribute that can be used as an access key. However, the maintenance of such indices, as the number of links grows, can rapidly becomes a difficult and time-consuming task.

In this circumstance, some essentially symmetrical treatment of attributes becomes necessary. We decided that merge-based database operations are good for symmetrical database manipulation. The reasons for this are as follows:

1) with a hardware sort/merge support, the time required to perform database operations can be minimized, in the sense that the operations are performed concurrently with data transfer,
2) merge-based database operations on each attribute can be performed at a reasonable speed.

The sorter, in our implementation, is a piece of hardware that receives a data stream as input and outputs a data stream that is a permutation of the input stream. To sort a long stream that cannot be sorted in one scan, some external merging of sorted substreams is necessary. The maximum length of the substreams is, naturally, the length the sorter can handle in one scan, in other words, the sorter's capacity. We provided a merger for performing external sorting by merging. The sorter's capacity is limited; however, by providing a merger in addition to the sorter, arbitrary-length external sorts can be performed by merging. Thus streams of any length can be sorted.

The basic idea that the merger is necessary for performing external sorting. By modifying the output control of the merger, various relational database operations can be accommodated by the merger.

A block diagram of the merger is shown in Fig. 5. The merger consists of an operation section and an output control section.

The operation section contains two 64K-byte memories (U-memory and L-memory). Each has a FIFO function, a comparator and a control-ROM table. This section performs the following steps:

(1) Storing two sorted streams from the sorter into the memories
(2) Reading a tuple from each of two memories simultaneously and providing them to the comparator and the tuple memory in the next section.
(3) Comparing the keys of each tuple and detecting output tuples satisfying the condition of the command.

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

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

(1) Reordering the fields of an output tuple.
(2) Selecting fields of an output tuple.
(3) Undoing the transformation of the key field.
(4) Adding a new TID to an output tuple.

Examples of these functions are shown in Fig. 6. Fig. 6(a) shows reordering of the fields of an output tuple. That is to say, tuple (1) with five fields (A, B, C, D, E; B is a key field) is rotated to tuple (2) by the IN module, so that the key field is positioned at the head of the tuple, and tuple (2) is rearranged to the original tuple (3) by the merger.

The selection of fields of an output tuple is shown in Fig. 6(b). That is to say, tuple (4) is projected to tuple (5) or tuple (6) by the assignment of the two pointers, P1 and P2. Fig. 6(c) shows the addition of a new TID (NTID) to an output tuple.

In the remainder of this section, we describe algorithms for external sorting and the join operation as examples of relational database processing.

A. External Sorting

We assume that the substreams to be externally merged are represented as S1 and S2, and that S1 and S2 have been previously sorted. S1 consists of S11, S12, S13,..., S1n; S2 consists of S21, S22, S23,..., S2n. Sijs are segmented beginning with the top of each substream, so Sij is a sorted substream and every tuple in Sij is less than or equal to an arbitrary tuple in Sij+1. All Sijs are of the

same length (the capacity of the sorter's internal sort). U-memory and L-memory have the same 64K-byte capacity.

1) let i = j = 1
2) load S1i into the U-memory
3) load S2j into the L-memory
4) merge the top tuples in the U- and L-memories as soon as the first tuples of S1i and S2j appear at the top of the queue
5) if U-memory is exhausted, increment i by one tuple-length and if i $\leq$ n, load S1i into U-memory, then go to 4. If i gets greater than n by incrementing, go to 7.
6) if L-memory is exhausted, increment j by one tuple-length and if j $\leq$ n, load S2j into L-memory, then go to 4. If j becomes greater than n by incrementing, go to 7.
7) output the remaining tuples (S1 or S2).

B. **Join operation**

An example of JOIN-EQ command processing is shown in Fig. 7. The JOIN-EQ command is typically used when an equi-join of two relations is performed. Fig. 7(a) shows two input streams (S1 and S2) sorted in ascending key-order (A1 and B1); these are stored in the U- and L-memories, respectively. UADR and LADR provide sequence numbers for explaining the address control scheme of each memory. Fig. 7(b) shows the output tuples and Fig. 7(c) illustrates the execution process.

The processing algorithm for the JOIN-EQ command is as follows:

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

Here, DP stands for the duplication line and LADR# points to the first tuple of the duplicate tuples.

This algorithm is more efficient than a simple merge algorithm, because the duplication controls the selection of the tuple to be processed. Other merger commands are also executed by an algorithm similar to the above example.

6. PERFORMANCE ESTIMATION

As the sorting algorithm proceeds in a deterministic way, the time required to sort a stream of tuples can be estimated precisely. The following list sums up the parameters describing the two-way merge sorter:

N: sorting cell count
M: merger's memory capacity

C: stream tuple count
L: tuple length
T: time to transfer a byte

To estimate the time required to sort a stream using the sorter, the length of the stream must be determined. A sort that is performed by scanning the whole stream once, in other words, a complete stream sort that uses only the sorter section, is called an **internal sort.** A stream larger than the maximum internally-sortable stream has to be merged after the internally-sortable substreams are sorted. We call a sort in which the merging of presorted substreams is necessary an **external sort.**

The merger's memory capacity is twice that of the last stage sorting cell memory. So M is 2**(N + 1) * Lmax, where Lmax is the maximum tuple length, under the restriction that the stream of length 2**N can be sorted by the N-stage sorter. If M is 64K bytes and N is 12 stages, Lmax is 64K bytes/2**12 = 16 bytes. That is to say, 2**12 (4096) tuples of up to 16 bytes can be sorted internally by this sorter. The time required to sort C tuples whose length is Lmax is determined according to the external merge count. We define the effective sorter capacity, E, as:

$$E = \min (2^{**}N * L , M).$$

E represents the maximum sorter capacity that can be effectively used for an internal sort. If L is 1 byte, for example, although the sorter's capacity is one M byte, only 2**N bytes can be used to internally sort the stream.

Then, we define the merge count factor, F, as:

$$F = [\log(CL/E)].$$

F determines how many times the stream has to be merged. If F equals 0 (i.e., CL/E is less than or equal to 1) the stream can be internally sorted without using the merger. If F equals 1, merging of sorted substreams is necessary. In this case, however, the time required to load a substream can be reduced by flowing the first sorted substream directly into the merger memory, so the stream does not have to be read twice. If F is greater than or equal to 2, however, the sort strategy changes. As the sorter and merger combination can not sort a stream longer than 2E in one scan, the stream is divided into substreams of length 2E. First, the substreams are sorted as in the F = 1 case. Then, the sorted substreams are externally merged.

The time required to sort a stream is determined according to the F values.

Case 1: F = 0

$$T_{sort} = (2CL + L\log(C))T + T_{overhead}(F)$$
approximately, $2CLT + T_{overhead}$

Case 2: F = 1

$$T_{sort} = (3E + \log(E) + 2(CL-E) + \log(CL-E))T + T_{overhead}(F)$$
$$= (2CL + E + \log(E) + \log(CL-E))T + T_{overhead}(F)$$
approximately, $2CLT + ET + T_{overhead}$

Case 3: $F \geq 2$

Approximately,

$$Tsort = (CL/2E)5ET + 1.5(CL/E)ET[\log(CL/2E)] + Toverhead(F)$$
$$= 2.5CLT + 1.5CLT\log(CL/2E) + Toverhead(F)$$

Here, (CL/2E)5ET means that the stream of length CL is divided into (CL/2E) substreams of length 2E, each of which requires a time of 5ET to sort. (CL/E)ET is the time required to transfer the stream of length E (CL/E) times. The coefficient 1.5 means that the mean time required to merge two substreams requires 50% more time than is required by the substream transfer. Log(CL/2E) is the mean merge count for a substream to form the final sorted result.

The implementation values are as follows:

N: 12, the sortable tuple count is 2**12 = 4096
M: 64KB
Lmax: 64KB/4096 = 16B
T: 1/(3MB/sec) = 0.33 * 10**(-6)

The sorting time versus tuple count when tuple length is 16 bytes is shown in Fig. 8.

7. CONCLUSION

We have described our design considerations and the implementation of the sorter used in the RDBE for relational database processing. Our sorter is characterized by the following features:

(1) High performance, achieved through pipelined processing synchronized with the data transfer rate (3MB/sec)
(2) Sufficient data types, including null values, are provided by an IN module
(3) High reliability, achieved by using a sorting checker
(4) Relational database operation capability, achieved by using a merger

The sorter was implementated using currently available technology. The time elapsed from the initial design stage to the implementation in RDBE was about a year and a half. The RDBE is incorporated in the Delta database machine and is currently undergoing system testing.

Activities planned for the future include a gate-array implementation of the sorting cell with the exception of the memory element.

Acknowledgement

We would like to express our sincere appreciation to the members of ICOT's KBM (Knowledge Base Machine) group for valuable discussion, and to Dr. K. Mori, director of Toshiba information systems laboratory, who provided the opportunity to conduct the present research. We would also like to thank the Toshiba development group, in particular K. Oda, T. Oka and A. Tanaka, for their cooperation in designing and implementing the RDBE.

References

[1] D.E. Knuth, The Art of Computer Programming, Vol.3, Sorting and searching, Reading, MA, Addison-Wesley, 1973.

[2] F.P. Preparata, "New Parallel Sorting Schemes," IEEE Trans. Comput., Vol. C-27, pp.669-673, July, 1978.

[3] Y. Dohi, A. Suzuki and N. Matsui, "Hardware Sorter and its Application to Data Base Machine," Proc. 9th Annual Symp. on Computer Architecture, pp.218-225, April, 1982.

[4] F.Y. Chin and K.S. Fok, "Fast Sorting Algorithms on Uniform Ladders (Multiple Shift-Register Loops)," IEEE Trans. Comput., Vol. C-29, pp.618-631, July, 1980.

[5] Y. Tanaka, Y. Nozaka and A. Masuyama, "Pipeline Searching and Sorting Modules as Components of Data Flow Database Computer," Proc. IFIP'80, pp.427-432, Oct., 1980.

[6] H. Yasuura, N. Takagi and S. Yajima, "The Parallel Enumeration Sorting Scheme for VLSI," IEEE Trans. Comput., Vol. C-31, No.12, pp.1192-1201, Dec., 1982.

[7] L.E. Winslow and Y.C. Chow, "The Analysis and Design of Some New Sorting Machines," IEEE Trans. Comput., Vol. C-32, pp.677-683, July, 1983.

[8] C.D. Thompson, "The VLSI Complexity of Sorting," IEEE Trans. Comput., Vol. C-32, pp.1171-1184, Dec., 1984.

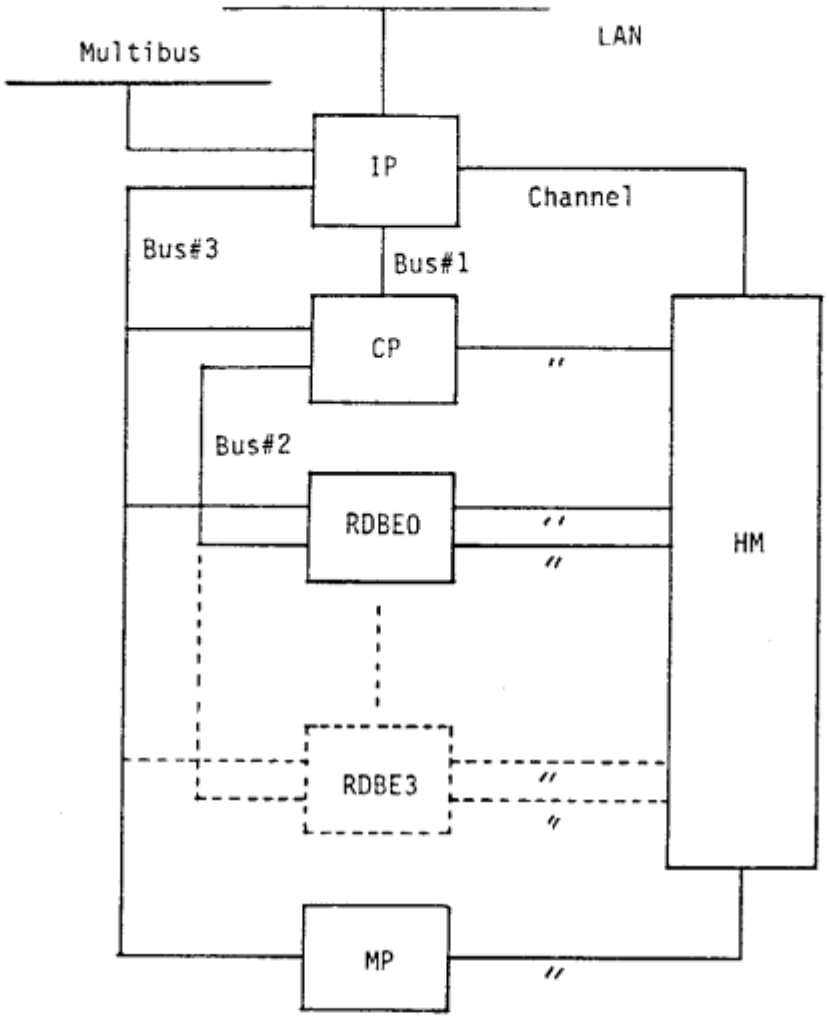
[9] S. Todd, "Algorithm and Hardware for a Merge Sort Using Multiple Processors," IBM Journal of Res. and Develop., 22, 1978.

[10] S. Shibayama, T. Kakuta, N. Miyazaki, H. Yokota and K. Murakami, "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor," ICOT Technical Report TR-055 and also to appear in New Generation Computing, Vol.2, No.2, March, 1984.

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

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

[13] D.K. Hsiao(ed.), Advanced Database Machine Architecture, Prentice-Hall, 1983.



LAN : Local Area Network
IP : Interface Processor
CP : Control Processor
RDBE: Relational Database Engine
MP : Maintenance Processor
HM : Hierarchical Memory

Fig. 1 Delta architecture

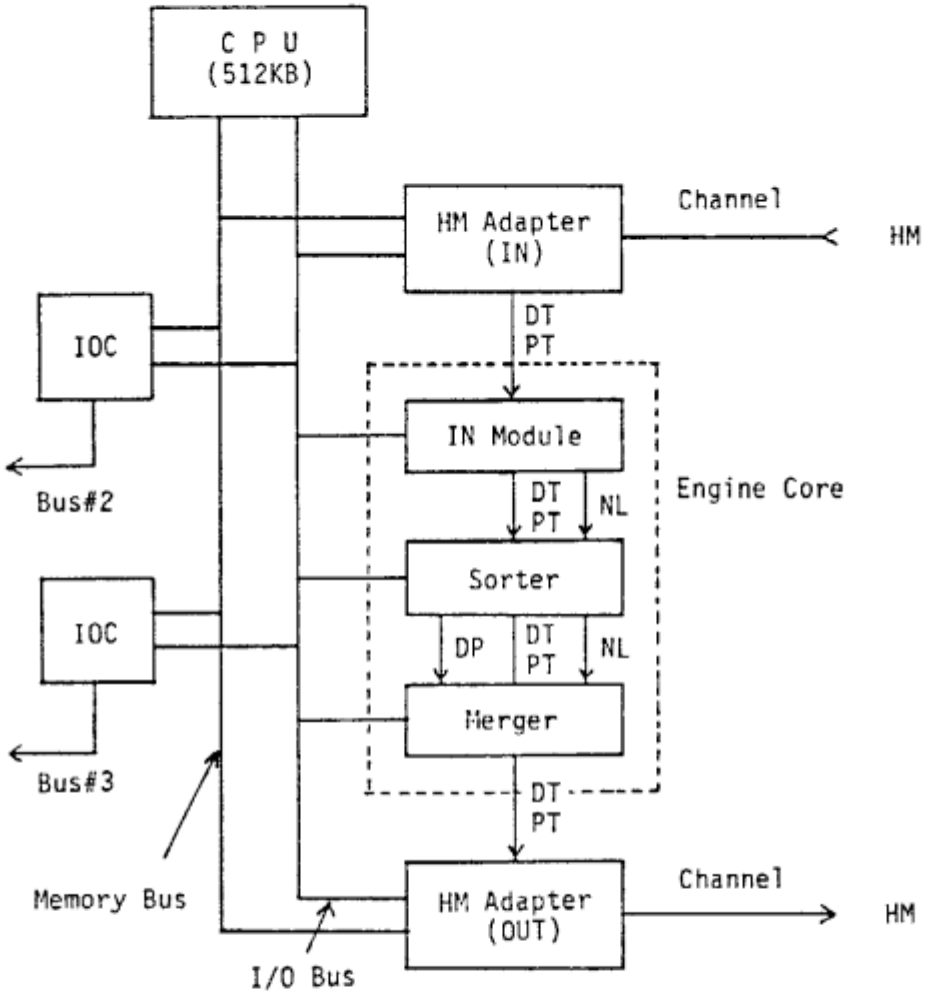


Fig. 2 RDBE configuration

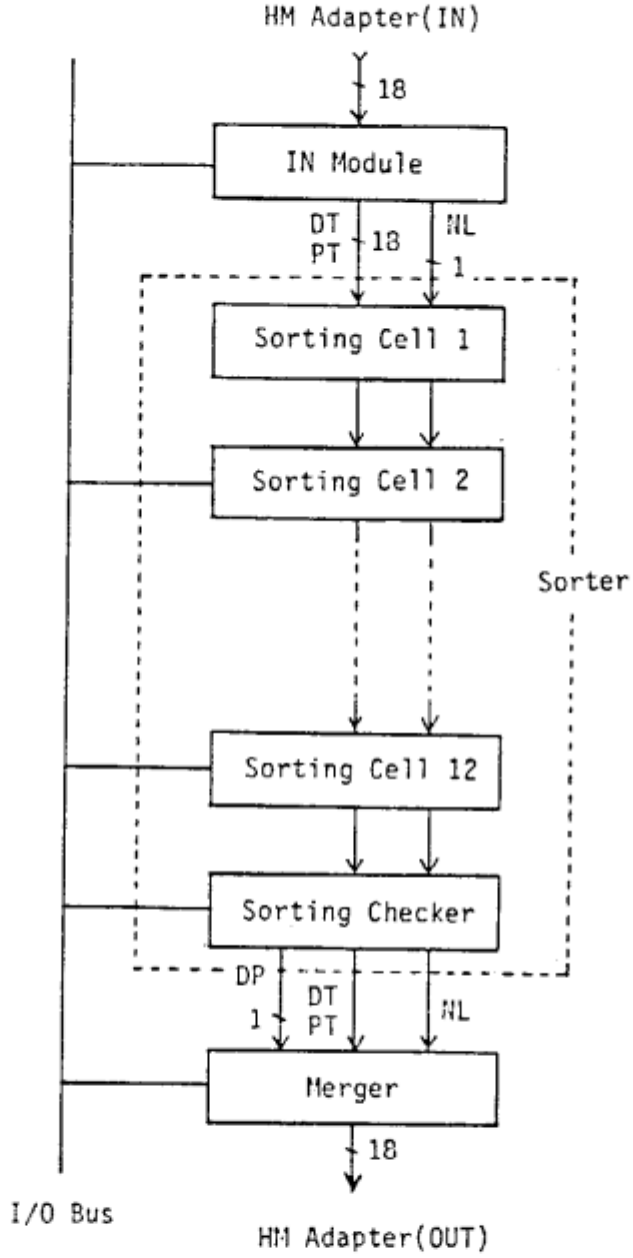


Fig. 3 Sorter configuration

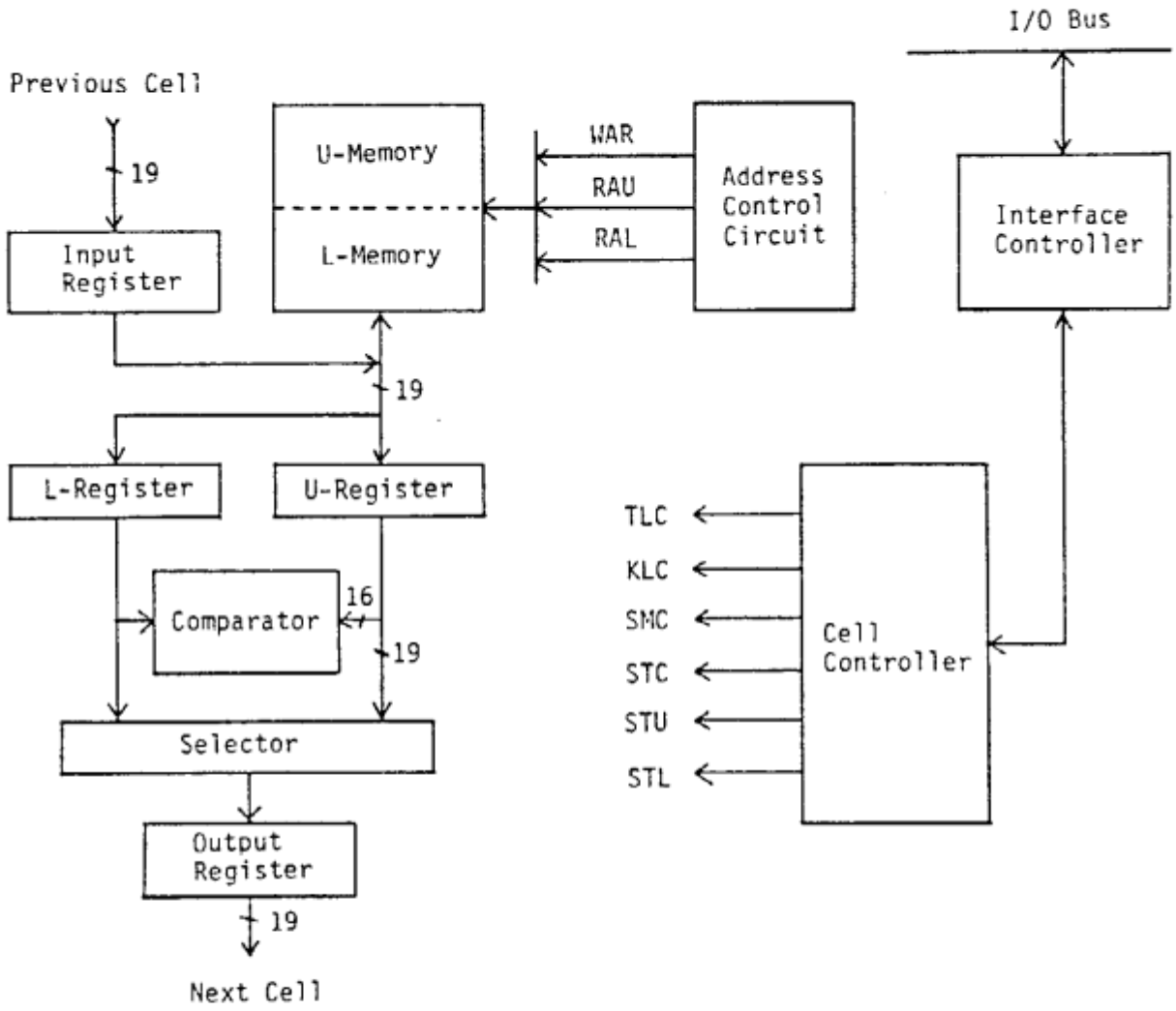


Fig. 4 Block diagram of sorting cell

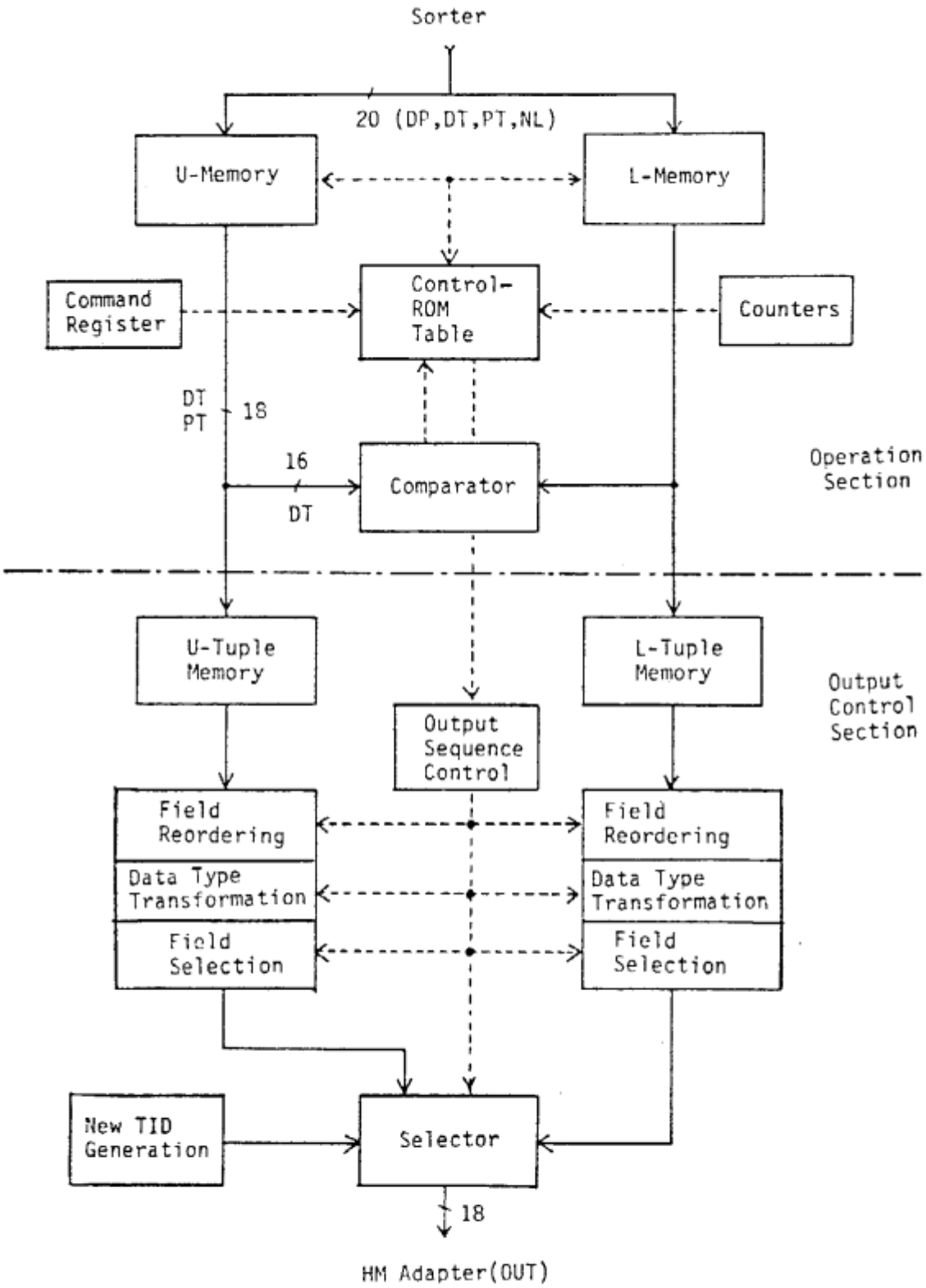


Fig. 5 Block diagram of the merger

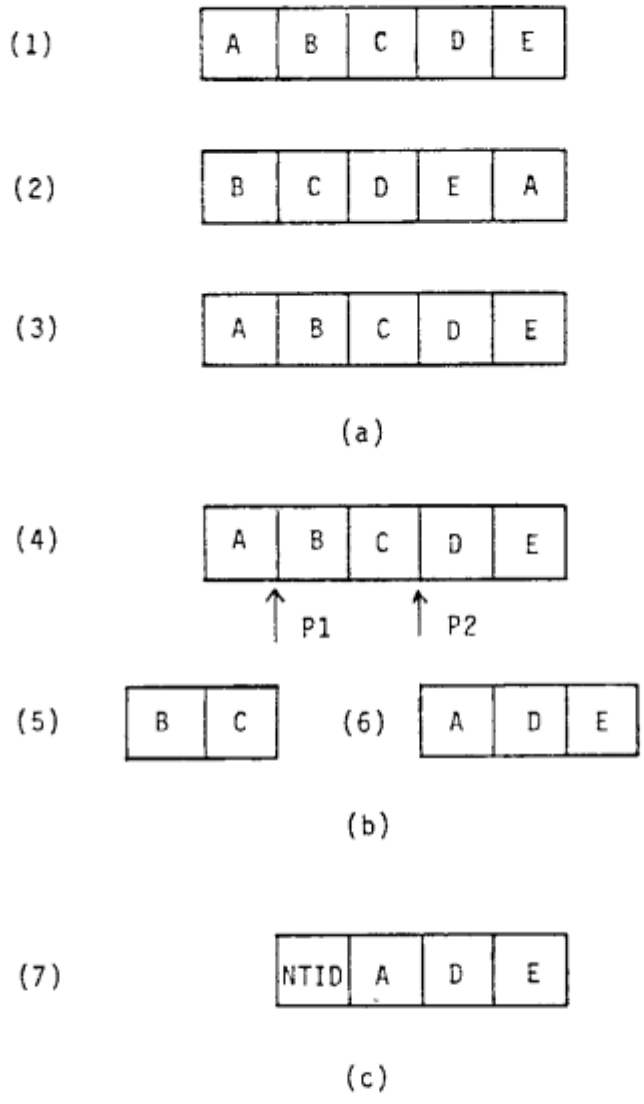
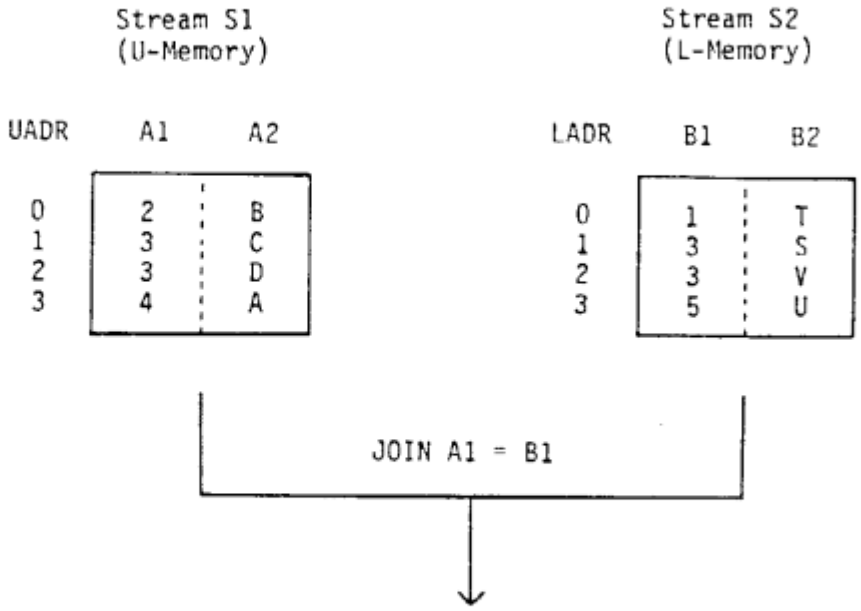


Fig. 6 Functions of the output control section of the merger

(a)



(b)

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

(c)

UADR	LADR	U-TUPLE	L-TUPLE	RESULT	OUTPUT
0	0	2B	1T	>	
0	1	2B	3S	<	
1	1	3C	3S	=	3C3S
1	2	3C	3V	=	3C3V
2	1	3D	3S	=	3D3S
2	2	3D	3V	=	3D3V
3	3	4A	5U	<	
END	3				

Fig. 7 Example of JOIN-EQ command processing

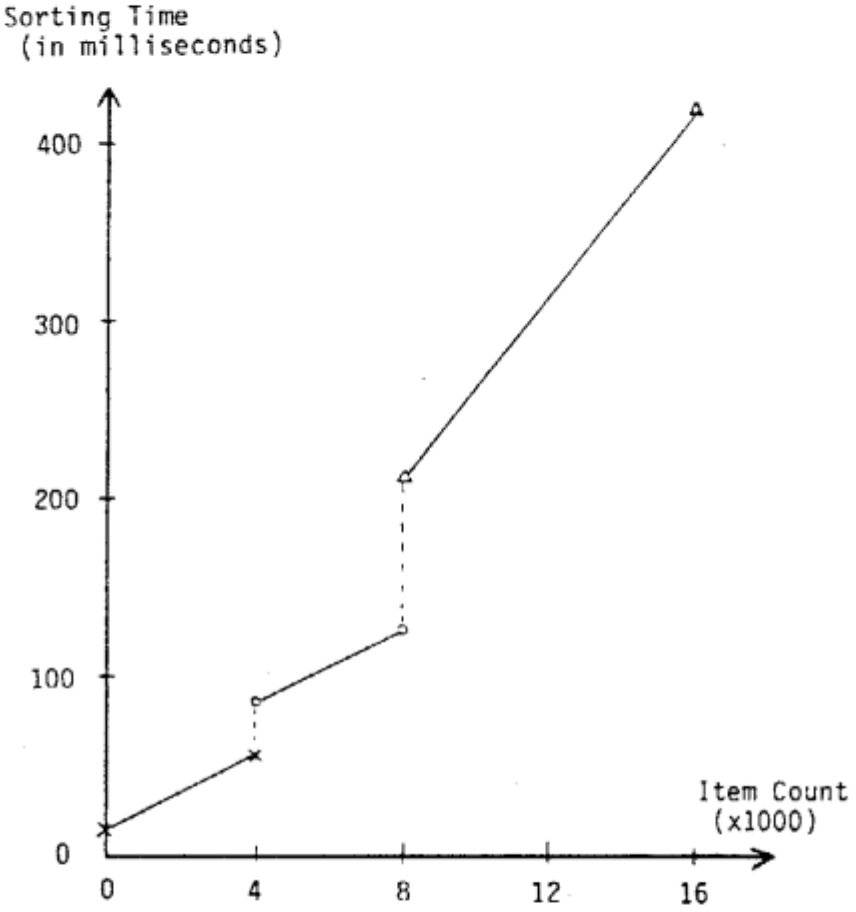


fig. 8 Performance estimation of the sorter