TR-182

Parallel Control Techniques for Dedicated Relational Database Engines

by
H. Itoh. M. Abe. C. Sakama (ICOT)
and Y. Mitomo (Japan Systems Corp.)

June. 1986

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

Parallel Control Techniques for Dedicated Relational Database Engines

Hidenori Itoh*, Masaaki Abe*, Chiaki Sakama*, Yuji Mitomo†

ICOT Research Center

Tokyo, Japan

May 31, 1986

Abstract

In this paper, we assume a back-end type relational data base machine equipped with multiple dedicated engines for relational database operations. Response characteristics are evaluated, and some parallel control techniques are considered for improved response time by simulating the database machine in executing relational database operations using these engines in parallel.

1 Introduction

The Fifth Generation Computer Systems(FGCS) project in Japan aims to develop a high level knowledge information processing system including inference and knowledge base mechanisms. In the first three-year stage (1982-84) of the project, personal sequential machine *PSI* was developed from research on inference mechanism [Yokota 83] . For knowledge base mechanism, a back-end type relational database machine *Delta* [Kakuta 85] , compatible with logic programming languages such as Prolog, was developed as the first step towards a knowledge base machine. Delta possesses the following characteristics.

1. Facts from logic programming languages are stored in relation.
2. The logical command interface with the host machines is based on relational algebra level commands.

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

†Japan Systems Corporation, Nomura Building, 4-4-8 Chiyoda-ku, Tokyo 102 Japan.

3. Assuming mass data processing, dedicated engines for relational database operations were provided for rapid execution of high-load operations such as join.

4. Hierarchical memory is provided, consisting of a disk drive for mass storage, and a semiconductor memory for inter-unit transfer buffer and workspace buffer volumes.

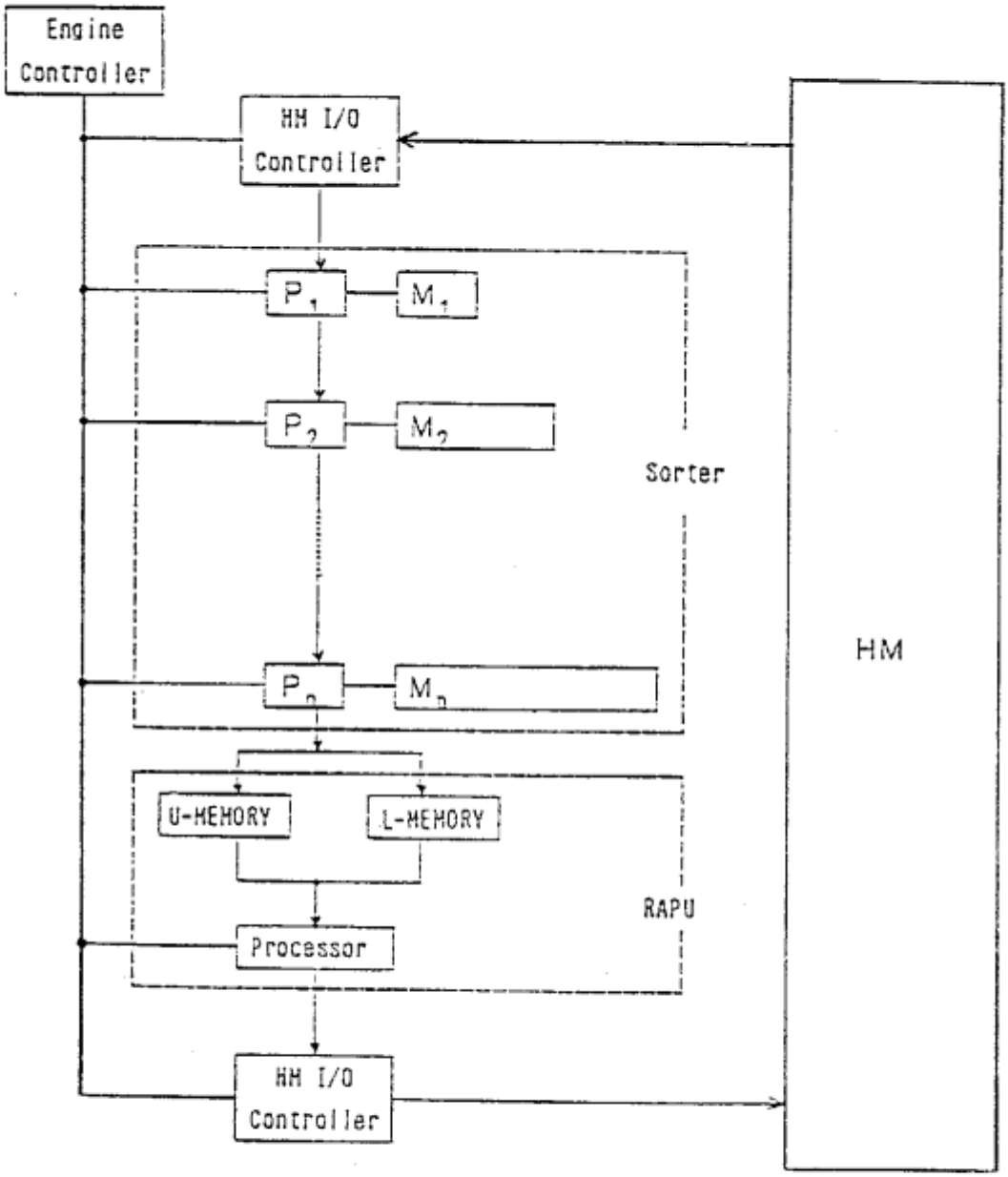
In this paper we assume a back-end type relational database machine equipped with multiple dedicated engines for relational database operations. Chapter 2 describes the configuration of the dedicated engine for the relational database operations(referred to as RE below), Chapter 3 and 4 describe the techniques of the relational database operations used in the RE, Chapter 5 discusses parallel processing techniques where there are multiple RE utilized, Chapter 6 presents the relational database machine model developed for research purposes and considers parallel processing controll strategies for the RE, and Chapter 7 evaluates parallel engine utilization to resolve relational operations through simulations using actual engine parallel operation, and hierarchical memory system(referred to as HM below) measured and experimental data. Finally, response characteristics and parallel control strategies to improve response are considered based on these experimental results.

2 A Dedicated Engine for Relational Database Operations

Assuming mass data processing, operations such as join, selection and projection present high processing loads when used on a relational database. The implementation of the join operation in particular, which combines two relations, has been cited as a major problem in relational database research [Tanaka 84] [Kitsuregawa 84] . For the efficient execution of relational algebra operations, it is often advantageous to sort the object relations by key attributes in advance, reducing the range of comparison and simplifying downstream processing : an approach which is especially effective in join and similar operation.

Based on this concept, we developed the RE indicated in Figure 1 to provide increased speed for relational algebra processing. The RE is composed of a sorter and a RAPU(Relational Algebra Processing Unit) [Sakai 84] , where the sorter sorts the data as a stream by controlling stream transfer time and overlap, and the RAPU is placed downstream of the sorter to execute relational algebra operations on the sorted stream without delay. Here the data transfer speed for the RE is $3Mbyte/sec$.

A pipelined 2-way merge sort algorithm is used for sorting [Todd 78] . When the number of



P_i : *ith Processor of Sorter* M_i : *Memory Unit of ith Processor*

RAPU : *Relational Algebra Processing Unit* *EM* : *Hierarchical Memory*

Figure 1 RE Hardware configuration

records to be sorted is N, this algorithm can reduce the sorting order $O(N \times log_2 N)$ to $O(N)$. The sorter is composed of 12 levels of processors, each with dedicated memory space. The volume of memory size, $size(M_i)$, increases with each higher level as :

$$\begin{aligned} size(M_1) &= 32bytes \\ &\vdots \\ size(M_i) &= 2(size(M_{i-1})) \\ &\vdots \\ size(M_{12}) &= 64Kbytes \end{aligned}$$

This means that the sorter is able to sort up to $64Kbytes$ of data at one time. N_{max}, the maximum number of records the sorter is able to process, is given by the following equation :

$$N_{max} = min(2^{12}, \quad [size(M_{12}) \,/\, L])$$

where L is the record length of relation R. $(max(L) = 16bytes)$

The RAPU consists of two memories(U-memory and L-memory) for storing sorted data, and a processor(comparator) which selects output data satisfying the condition. Each of these memories is $64Kbytes$ in size. In addition to the above RAPU and sorter, the RE includes an I/O controller for relation storage in HM, and an engine controller for overall control operations.

3 Relational Algebra Processing Techniques

This Chapter discusses the techniques for relational algebra processing used in the RE.

3.1 Join operation

When R_1 and R_2 are relations with length l_1 and l_2, θ-join on attributes A_1, A_2 of each relation is denoted by

$$R_3(l_3) = R_1(l_1) \underset{A_1 \theta A_2}{\bowtie} R_2(l_2)$$

or simply

$$l_3 = l_1 \bowtie l_2$$

where R_3 is the resultant relation with length l_3.

If the two relation record counts are N_1 and N_2, then a join operation without sorting is $O(N_1 \times N_2)$, but a join operation preceded by a sort operation can executed in $O(N_1 + N_2)$.

Expressing the sorting of a relation R with length l as $\mathbf{S}(R(l))$,or simply $\mathbf{S}(R)$, and the resultant relation as **R**, the join operation is executed in the following steps :

*step*1: Load relation R_1 into the sorter.

*step*2: Perform $\mathbf{S}(R_1)$ in the sorter.

*step*3: Store $\mathbf{R_1}$ into RAPU U-memory.

*step*4: Load relation R_2 into the sorter.

*step*5: Perform $\mathbf{S}(R_2)$ in the sorter.

*step*6: Store $\mathbf{R_2}$ into RAPU L-memory.

*step*7: Once the first record of $\mathbf{R_2}$ has begun to be stored into the L-memory, the $val(A_2)$ of that record, and the $val(A_1)$ of the first record of $\mathbf{R_1}$ stored in the U-memory are compared by the processor. (Here, $val(A_i)$ express the value of A_i.) Records satisfying the condition $val(A_1)\theta val(A_2)$, the tuples are combined, or else the required attributes are picked up and output. The above procedure is repeated in FIFO order for each record in the U- and L-memories.

3.2 Selection operation

It is possible to interpret the selection operation as a join operation between the value in the condition (equivalent to relation R_1 of a single attribute), and the selection object relation R_2, with no output of R_1 records. Processing is the same as for the join operation.

4 Mass Data Processing Techniques

This Chapter discusses the processing techniques used for mass data exceeding $64Kbytes$ in length.

4.1 Sort operation for mass data

Where the relation with length l is $64Kbytes < l \leq 128Kbytes$, the initial $64Kbytes$ of data are sorted by the sorter, and then stored into the RAPU U-memory. The remaining $(l - 64)Kbytes$

of data are sorted by the sorter, and then stored into the RAPU L-memory. The records of the U- and L-memories are then merged in the comparator to sort the entire data set.

Where $l > 128Kbytes$, sorting is accomplished as follows :

*step*1: The data readied in the HM buffer $BUF0$ is received from the HM in $64Kbytes$ multiplexed blocks, and sorted by the sorter in the same $64Kbytes$ units. The output is merged by the RAPU as 64×2 $Kbytes$ units, with output alternately to $BUF1$ and $BUF2$.

step$S(S = 2, 3, \ldots, \lceil log_2(l \,/\, 128Kbytes) \rceil - 1)$: At the $(S-1)th$ *step*, the data sorted in $64 \times 2^{s-1}$ $Kbytes$ units is stored in buffers $BUF1$ and $BUF2$. The engine receives a pair of $64 \times 2^{s-1}$ $Kbytes$ data blocks from $BUF1$ and $BUF2$, merges them in the RAPU to sort $64 \times 2^s Kbytes$, and then outputs them in units of that size to buffers $BUF3$ and $BUF4$, alternately.

step$(\lceil log_2(l \,/\, 128Kbytes\ ) \rceil)$: The data stored in the two buffers in the preceding step are merged, with output to $BUF5$, which is the normal output buffer.

An example of processing flow for 64×8 $Kbytes$ is indicated in Figure 2.

The total buffer capacity required for sorting a data with length l is, input buffer $BUF0$, working buffers $BUF1$ and $BUF2$ ($BUF3$ and $BUF4$),and output buffer $BUF5$, that is $3lKbytes$ in all.

4.2 Join operation for mass data

Where both relations exceed $64Kbytes$ in length, the following two techniques are possible.

(a) Both relations are sorted into $64Kbytes$ units, and join processing is executed in round-robin pairings of the $64Kbytes$ units with the following algorithm.

for $i = 1$ **to** $\lceil l_1 \,/\, 64Kbytes \rceil$

begin

S_{1i} is input from the HM, sorted in the sorter, and stored in the RAPU U-memory.

for $j = 1$ **to** $\lceil l_2 \,/\, 64Kbytes \rceil$

begin

S_{2j} is input from the HM, sorted in the sorter,

and stored in the RAPU L-memory. Once the first record has

begun to be stored, U- and L-memories records are compared in

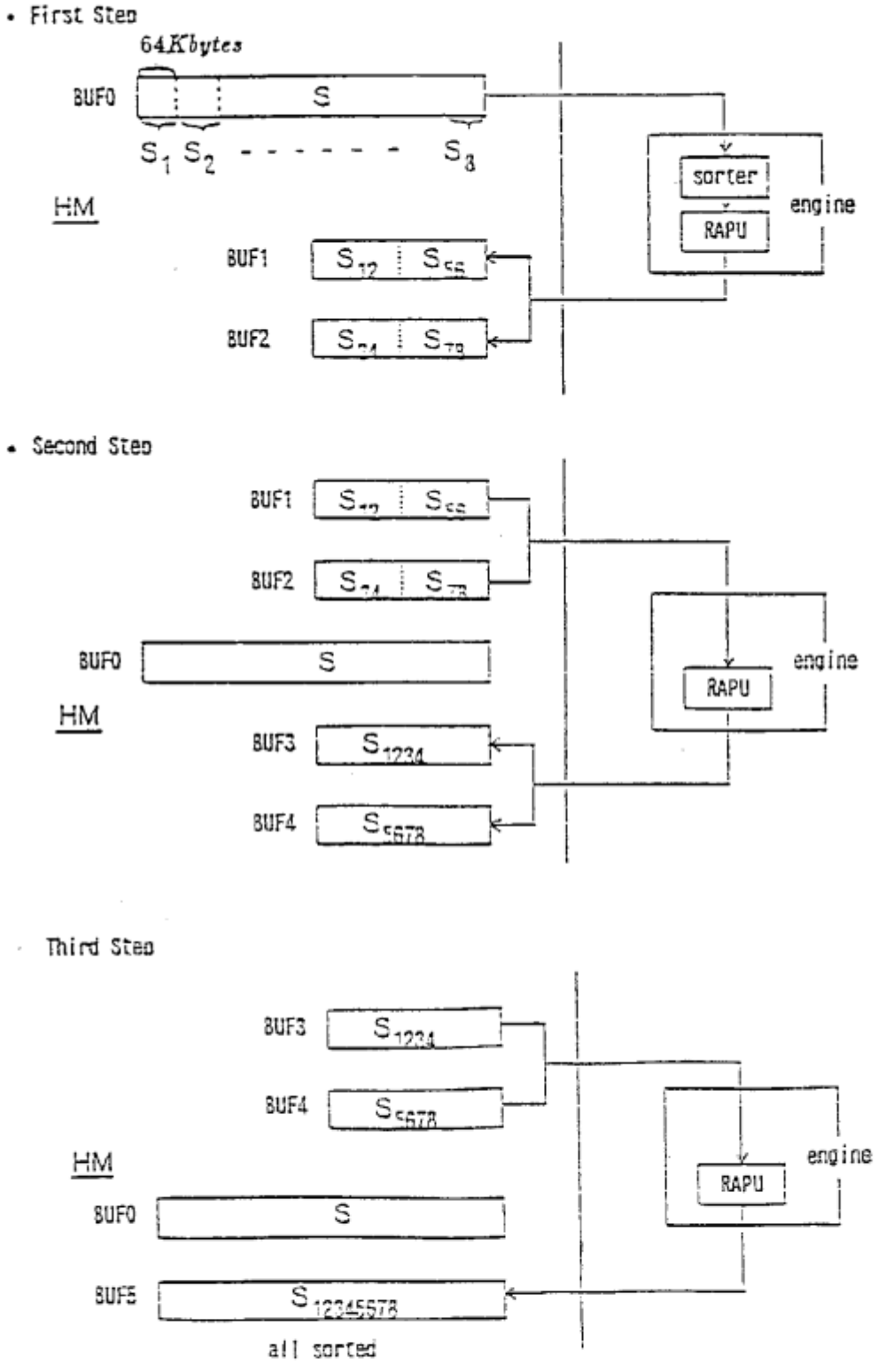


Figure 2 Execution example for $64 \times 8Kbytes$ sort operation

FIFO order, and those fulfilling the condition are output.

end

end

In this algorithm, S_{ij} represents the j-th sub-relation of relation R_i $(i = 1, 2)$, which is divided into $64Kbytes$ units.

(b) Both relations are sorted all over at first, then joined in the RAPU through comparison.

In the above techniques, (a) is $O(N^2)$ whereas (b) is $O(N)$. Figure 3 indicates comparisons of these two techniques in actual implementation processing, where the both relations record sizes were $16bytes$. As is apparent from the figure, (a) offers superior efficiency when both relations are relatively short, but above about $380Kbytes$, (b) is more efficient. And in (a), if the length of one relation l_1 is $64Kbytes$ or less, l_1 is sorted and stored into the RAPU U-memory, after which the other relation l_2 is sorted in 64 $Kbytes$ units, stored into the RAPU L-memory, while comparing with l_1 and output.

For this reason, when join operation is executed, ordering of the sequence of operation, such as placing selection operation first, in order to reduce at least one relation to $64Kbytes$ or less enables significant efficiency improvement. The evaluations of the join operations presented below were all performed using technique (a).

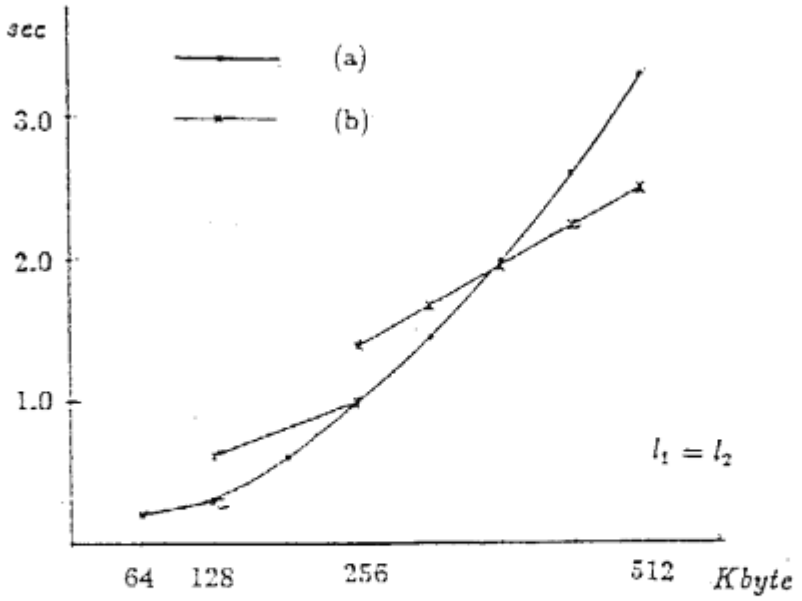


Figure 3 Comparisons of two techniques in join operation

4.3 Unit performance

The performances for sort and join operations in the RE actually implemented are shown in Figure 4. For these trails, object relations were stored at random in the HM, and all record sizes were 16*bytes*. Times were measured from the RE command interpretation to response generation, using an RE time measurement module. Figure 4 indicates that in the sort operation the doubling of the data volume intervals at each level is due to the sort operation described in Section 4.1, whereas the increase in the join operation in 64*Kbytes* units is due to the fixation of one data set to 64*Kbytes* while the other mass data set is sorted in 64*Kbytes* units. In both cases, $O(N)$ was assured per engine processing unit.

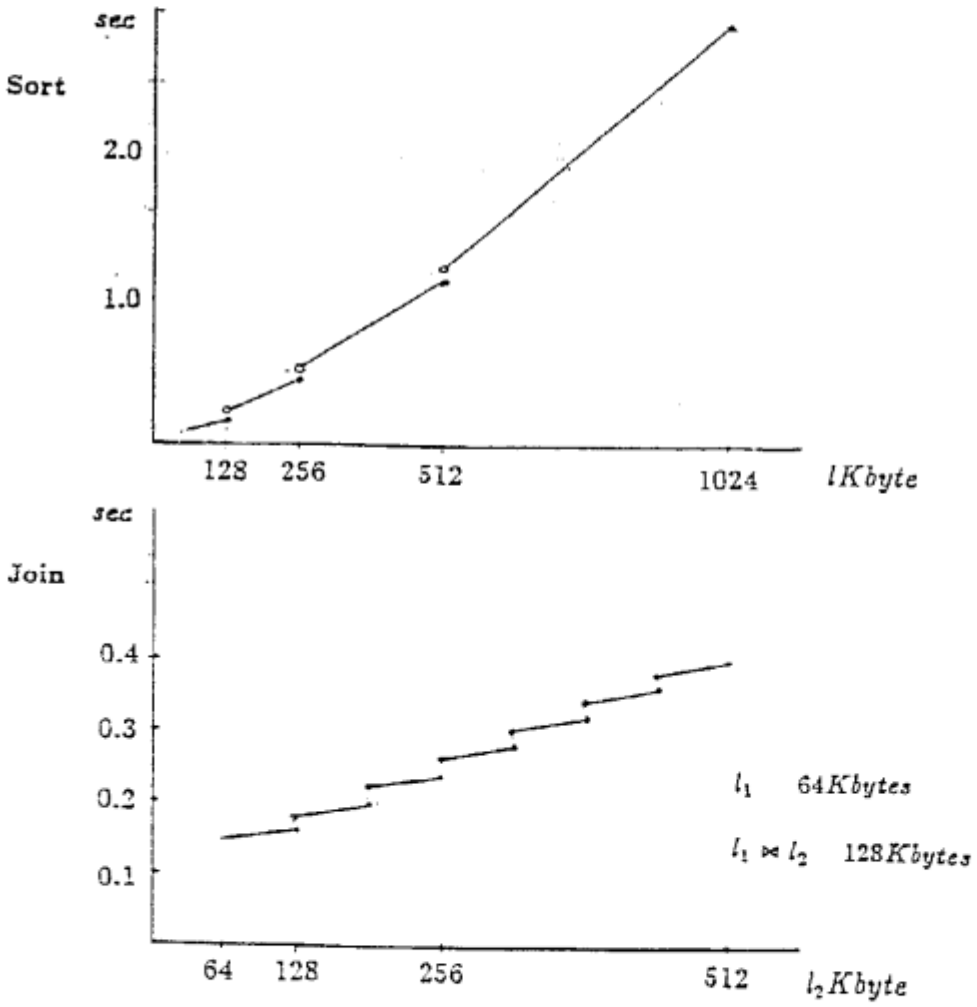


Figure 4 Unit performance

5 Data Division Relational Algebra Processing Techniques

This Chapter discusses the parallel execution techniques by which multiple RE are used to execute relational database operations in parallel.

5.1 Data division sort operation

When mass data is sorted, major decreases in execution time can be achieved by first dividing the data into a number of data segments equal to the number of inactive engines, with each segment then being assigned to an engine to be executed in parallel. This procedure is referred to as the *data division sort operation*, and is executed in the following steps :

step1: Where m be the number of inactive engines and l be the length of the data to be sorted, then the data is divided into m data segments and each is stored into one of m buffers. The size of each buffer is l/m.

step2: The data segments in the buffers created in *step1* are assigned to m engines respectively, then sorting is executed in parallel, with the results output to the m buffers.

step3: Sorted data segments are combined two-at-a-time into P pairs, followed by merging on P engines, which outputs P sorted data segments. This procedure is repeated until data length l is sorted.

In general, when data of length l is sorted in parallel by m engines, the required HM buffer size is $3 \times l$, the same as for a single engine. The number of engines operating at each merge level is half the number of the preceding level, and input data is twice the quantity of the preceding level. This situation is indicated in Figure 5, where the operating engines are shaded.

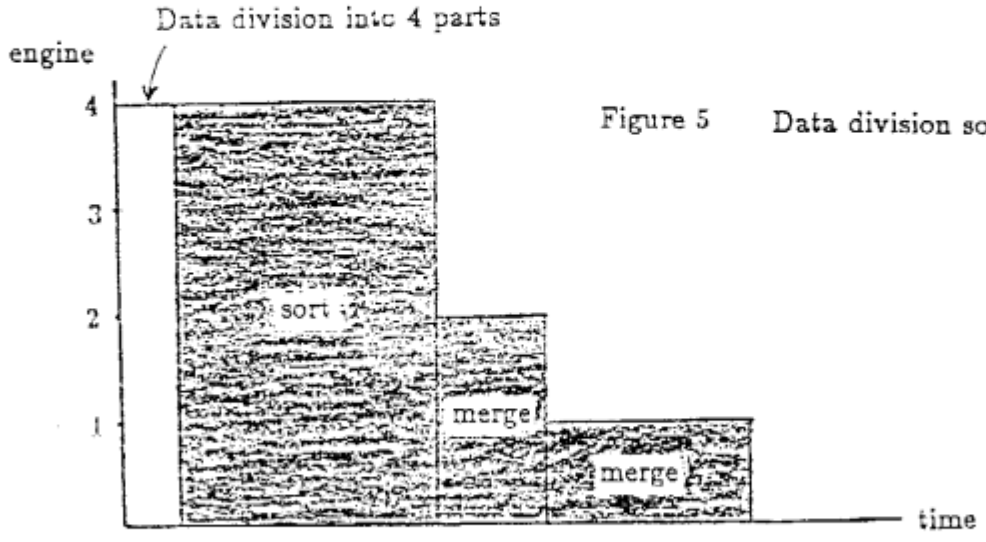


Figure 5 Data division sort operation

5.2 Data division join operation

For two relations with lengths l_1 and l_2, the join operation $l_1 \bowtie l_2$ can be executed in parallel by m engines by dividing one of the relations into m data segments, and $l_1/m \bowtie l_2$ is executed by a single engine. This procedure is referred to as the *data division join operation* and is executed in the following steps :

*step*1: The relation with length l_1 is divided into m data segments, and each data segment is stored into one of m HM buffers. The data length of each buffer is l_1/m.

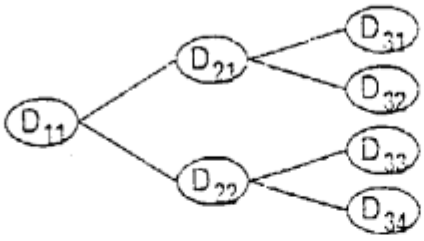
*step*2: The join operation $l_1/m \bowtie l_2$ is executed in parallel by m engines.

*step*3: The execution results from each engine are output to the output buffer.

5.3 Data division subcommand tree

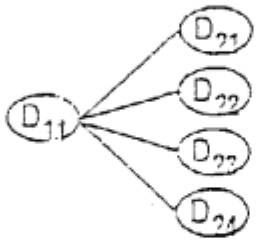
The data division sort and data division join operations are converted into subcommands with tree structures and then executed. We call these trees *data division subcommand tree*. The data division subcommand tree for data divided into m data segments is abbreviated as m-DDST or simply DDST. The j-th node at depth i is expressed as D_{ij}.

For example of Figure 2, a binary tree is formed by the 4-DDST of the sort operation S($8Mbytes$), as indicated below :



Nodes $D_{31} \sim D_{34}$ correspond to S($2Mbytes$), and nodes D_{21} and D_{22} correspond to the merge processing for the already-sorted pairs of $2Mbytes$ data bundles. D_{11} corresponds to the merge processing for the already-sorted pair of $4Mbytes$ data bundles.

The 4-DDST of the join operation $l_1 \bowtie l_2$ is given by the following tree.



D_{11} is a dummy node for synchronization, and $D_{21} \sim D_{24}$ correspond to $l_1/4 \bowtie l_2$ processing.

The m-DDST operation is executed from leaf to root, and parallel processing of nodes at equal depths is possible. Execution at a node is not initiated until the processing is complete for all nodes at levels above that node. In general, m-DDST depth for the sort operation is $log_2 m$ and for the join operation is 1.

6 Relational Database Machine Model

The relational database machine taken as the basis for this paper is a back-end type for a host machine. This Chapter discusses the internal execution control mechanism of the database machine in responce to the host machine inquiries, and engine parallel control strategies.

6.1 Relational database machine control model

The conceptual diagram of the relational database machine is indicated in Figure 6. It consists of the interface unit, control unit, schedule unit, HM control unit, and m engines. The interface unit, the control unit and the HM control unit assume to be a single CPU, with engines attached processors. In addition, the HM control unit will include an external disk drive (HM disk) for mass storage of relations. The individual units are described below.

1. Interface Unit (IU)

   The IU receives a query command from the host machine, and sends it to the control unit queue. If the IU receives the processing complete notice for that command from the control unit, the IU directs the output of resultant relations to the HM, and then outputs them from the HM to the host machine.

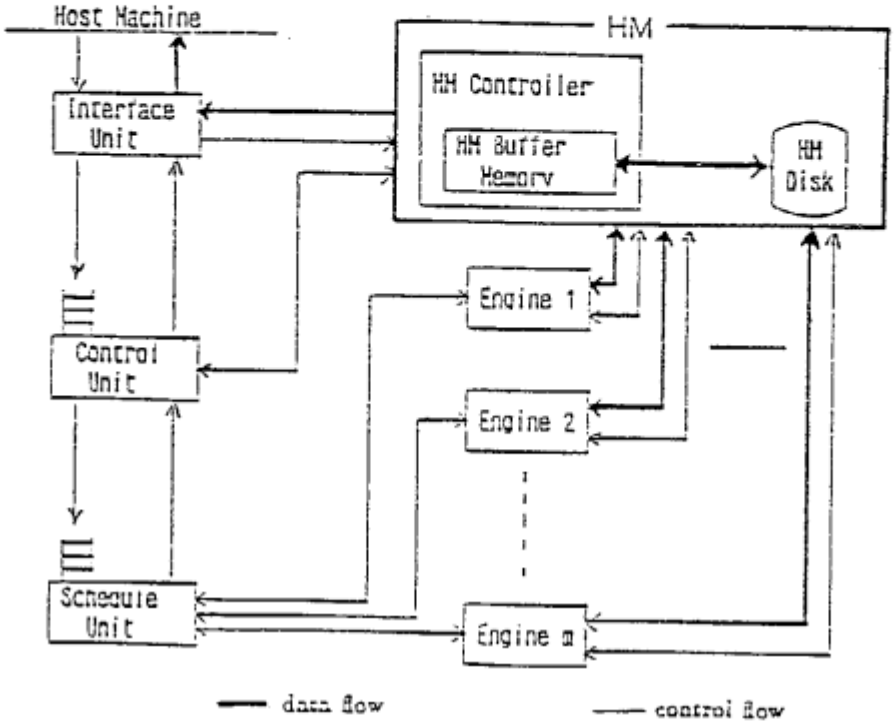


Figure 6 Block diagram of relational database machine

2. Control Unit (CU)

The CU takes commands from the queue in FIFO order, interprets them, and converts them into relational algebra internal commands. It outputs these commands to the schedule unit queue to instruct the staging of object relations from the HM disk to buffer memory, and the HM to prepare the output relation buffer. If all processing complete notices are received from the schedule unit, then a processing complete notice is sent to the IU.

3. Schedule Unit (SU)

The SU takes internal commands from the queue in accordance with the engine parallel control strategy. By monitoring the state of active and inactive engines, the SU determines whether to divide data or not for the internal command taken from the queue. If division is required, it determines the number of segments for each command, directs the HM to

divide the data, and converts data into DDST. The SU determines which selected commands and execution-enabled nodes will be assigned in what sequence to which engines. It sends execution directions to the assigned engines accordingly. If the SU receives the root processing completion notice for the internal command from an engine, it passes the notice on to the CU.

4. RE

The engines execute the internal commands and the nodes in DDST under direction of the SU. During execution, I/O inquiries for the object relations and the results are sent to the HM.

5. HM Control Unit

The HM control unit handles the staging of object relations from the disk to the buffer space, division processing, and preparing buffer memory for resultant relations. The replacement algorithm between the disk and the buffer memory utilizes the LRU (Least Recently Used) algorithm [Knuth 73].

6.2 Engine parallel control strategy

We consider three engine control strategies for the schedule unit as discussed below. At a given time t, the number of inactive engine is m_t, and the total number of engines is m_0.

a. Data non-division / allocation

All commands are executed by one engine each, without data division. m_0 commands are taken from the SU queue on an FIFO basis, and performed by m_0 engines in parallel.

b. Data m_t division / allocation

At time t, least recent command C is removed from the SU queue, converted to m_t-DDST, allocated to m_t engines, and parallel-processed in nodes corresponding to m_t-DDST leaves.

c. Data m_0 division / allocation

The same as *b*. above, except that C is converted to m_0-DDST instead.

6.3 Staging timing

The HM control unit functions include the staging of object data from the disk drive to the buffer memory, and the allocation of data to engines. Two staging timings are considered :

(i) Allocation to engines after staging to buffer memory completed.

(ii) Allocation to engines prior to staging to buffer memory.

The former method gives priority to the engines, aiming at increased engine utilization efficiency, while the latter takes increased memory utilization efficiency as its goal.

6.4 Objective functions

The following objective functions are considered :

1. Average response time

   The average time interval from the arrival of a query command C from the host machine at the IU until the response is output.

2. Average engine availability ratio

   The percentage of time that m_0 engines are operating.

3. Average HM memory utilization volume

   Average memory volume needed in the HM for engine command execution.

7 Evaluations and Considerations

This Chapter discusses the response characteristics and multiple-engine effects for parallel engine processing on the above-described relational database machine.

7.1 Multiple-engine effect

The multiple-engine effects for join and sort operations are indicated in Figure 7. The actual results are utilized for up to 4 engines, while results for 8 ~ 16 engines were determined experimentally through processing described above in Sections 5.1 and 5.2. The processing time indicated is the processing time of both RE and HM, and does not include overhead for the IU, CU, or SU. It is clear from the graph that there is a major effect for up to 8 engines, after which the effect drops off quickly..

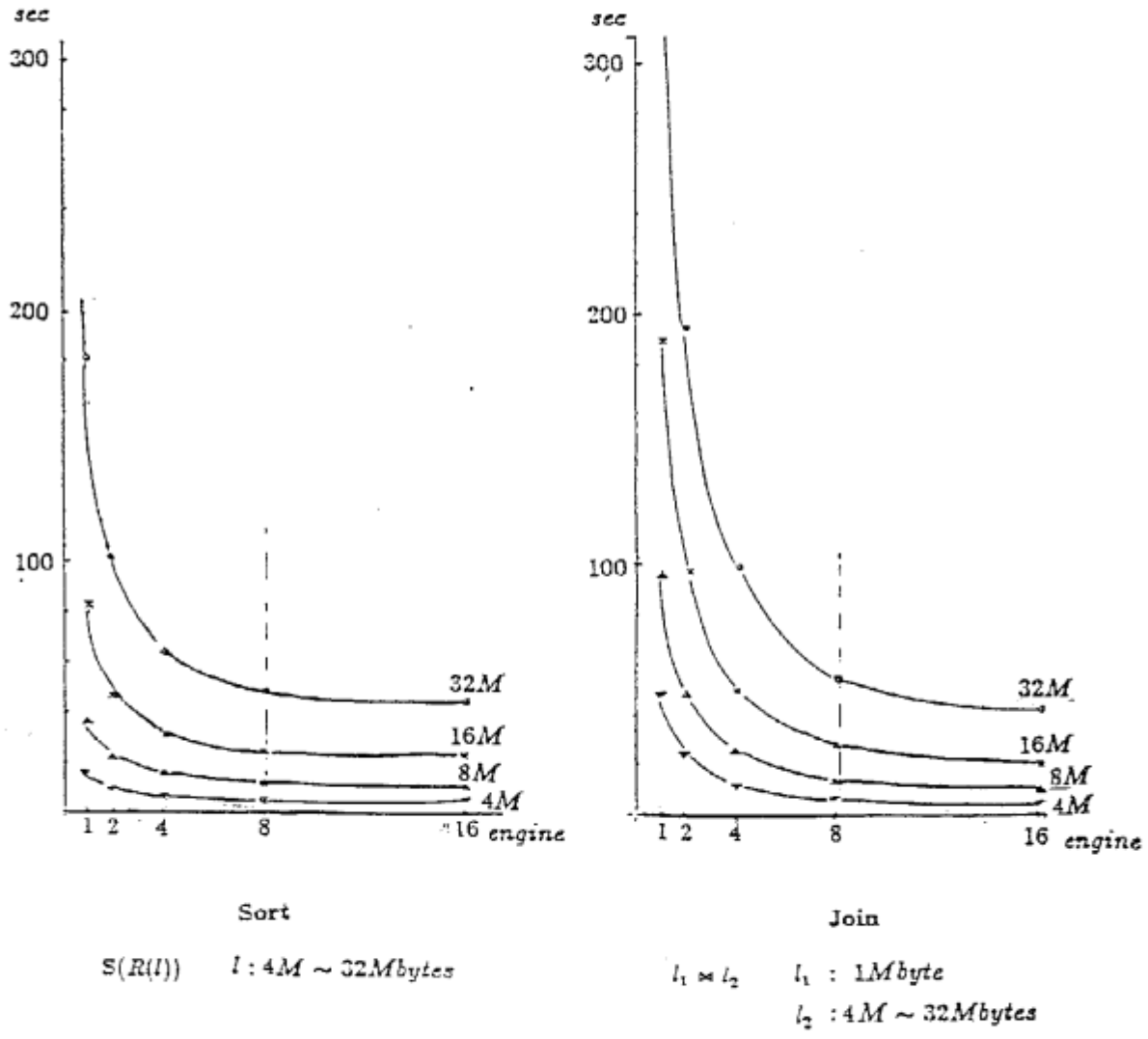


Figure 7 Multiple-engine effect

7.2 Simulation analysis

A simulation of the above-described relational database machine yielded the response characteristics presented below. The simulation parameters are as follows.

1. Resource parameters

   - The IU, CU, SU and HM control unit are located on a single CPU with a $5MIPS$ processing capacity. The engines are attached processors.
   - There are 4 engines.
   - HM buffer memory size is $64Mbytes$.
   - The transfer speed between the HM memory and the engines, and between the HM disk and the HM memory, is $3Mbyte/sec$.

2. Processing time parameters

   The processing time below are the actual time measured on Delta.

   - Execution time for a relational algebra operation on one engine.
   - Parallel execution time for a relational algebra operation on 4 engines.
   - Staging time for an object relation from the HM disk to the HM buffer memory, relation division time, and time to prepare a buffer for the resultant relation.

   The processing time below are estimated from actual measured time, or from dynamic step counts.

   - Processing and control strategy determination time in the CU and IU.
   - HM disk I/O time.

   Other processing time is ignored.

3. Host machine query command parameters

   - Command arrival is assumed to be Poisson-arrival.
   - Object relations are of random lengths from $1 \sim 16Mbytes$.

4. Simulation evaluation

   Under the above conditions, the following simulations were made.

   - Simulation where sort, join and selection operations arrived with equal frequency.
   - Simulation where only sort operations, or only join operations were received.
   - As above, with altered staging timing.

7.3 Observations

The above three simulation results are discussed below in detail.

1. Traffic density

   Figure 8 indicates where sort, join and selection commands arrive with equal frequency. From the response time characteristics, it is clear that there is a traffic width where each of the strategies is advantageous. That is, c is best for low traffic, b for intermediate levels, and a for high traffic densities. The results indicate the need for dynamic alteration of strategy in accordance with operation type, generation characteristics and traffic load.

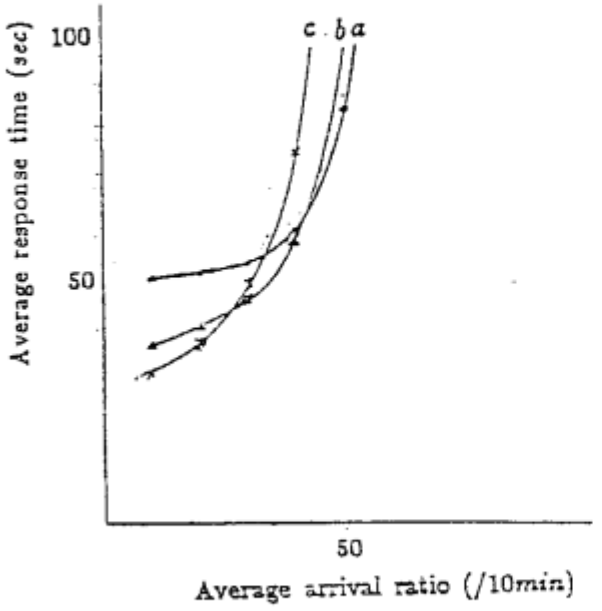
2. Operation type

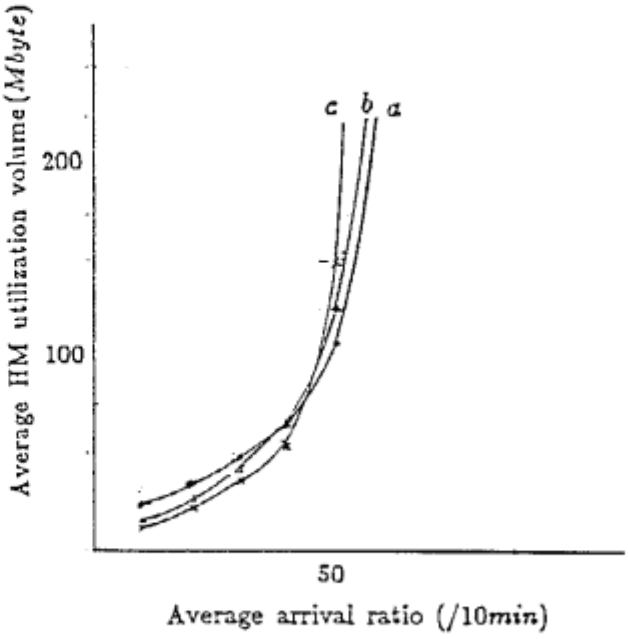
   Figure 9 indicates that for the sort operation, sparse traffic results in the response time ordering c, b, a, from best to worst, reflecting the order in which data division increases. With increasing traffic density, the order reverses, and b approaches a. As described in Section 5.1, the sort operation uses only half the number of engines at a given step as the number in the previous step, so that for high traffic densities, b is always allocated to a single engine. At a certain threshold, c shows a jump in response time, this is because the operations such as deep command tree processing which utilize the results of the above levels, when distribution increases, memory processing load increases accordingly.

   For the join operation, the response time is best in order of c, b, a, for the same reasons as sort's case. As traffic density increases, the order changes to a, c, b. This is because the join operation command tree depth is 1 as described in Section 5.3, so that all engines are freed at once in dummy node. This means static data division c, is more efficient than dynamic data division b, considering the scheduling overhead. The difference between c and a is the division overhead.

3. Memory / Engine priority

   Whether the engines or the memory are given priority as system resources depends on the balance of the two in a system involved. It is also conceivable to schedule by operation type. Figure 8 indicates memory-priority scheduling would seem to offer higher efficiency for deep command tree processing, such as sort operation, and engine-priority scheduling for shallow command tree processing, such as join operation.





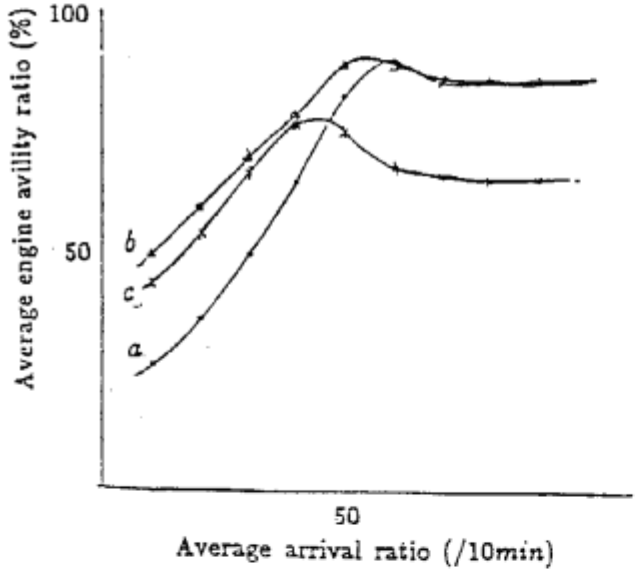


Figure 8 Response characteristics (1)

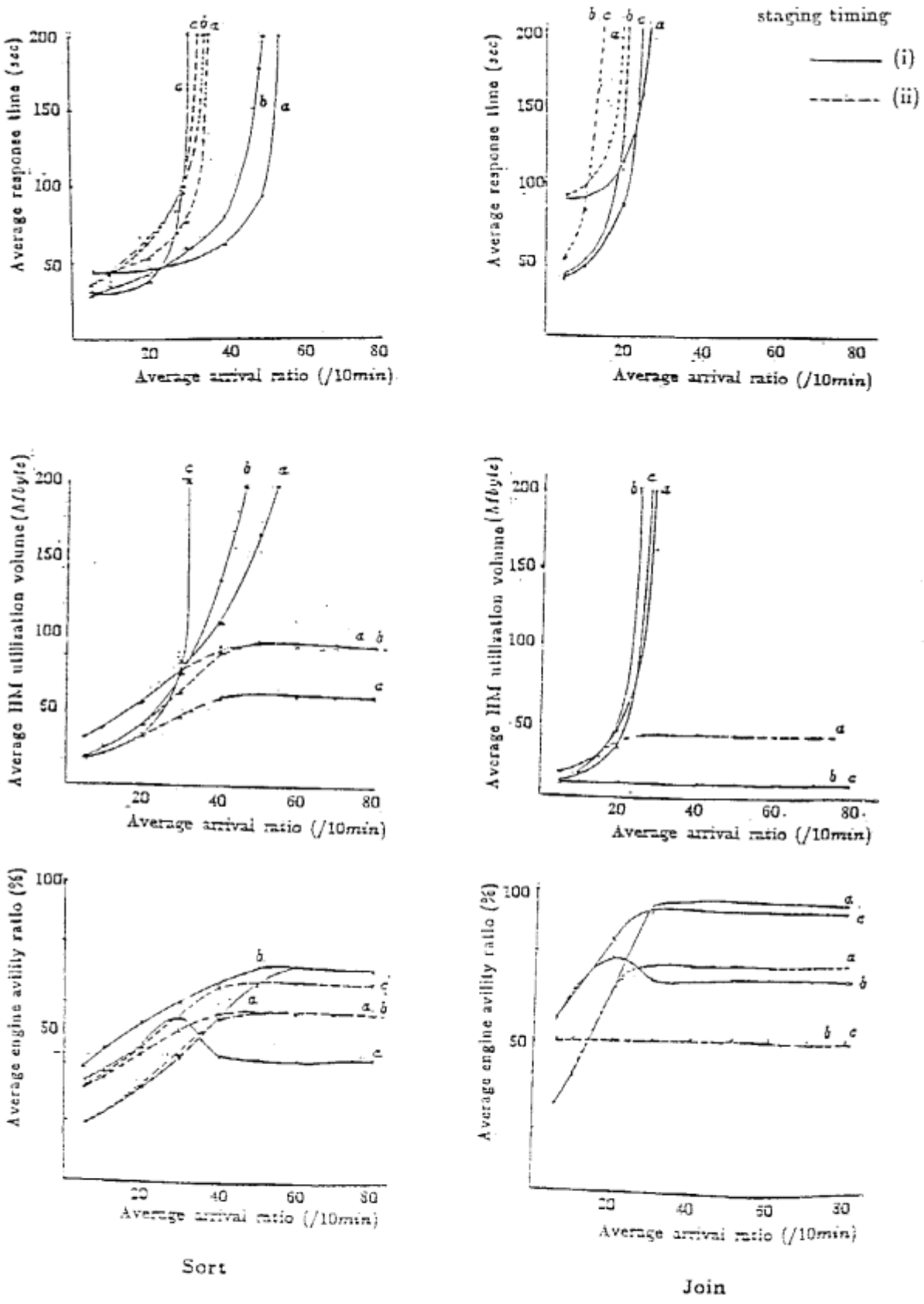


Figure 9 Response characteristics (2)

8 Conclusion

There have been many discussions on improving processing efficiency for relational algebra operations in relational database machines, such as scheduling strategies for operation execution sequences and so on [Smith 75].

Here we showed by simulations that when processing massive data volumes in parallel, other factors such as traffic density, operation type, physical resourse memory / engine priority allocation and relation size must be reflected in various control strategies. In this simulation we assumed a relational database machine based on Delta, but we think the simulation results can be applied to a general back-end type database machine with dedicated engines for relational algebra processing.

Future investigation is planned for the parallel control techniques for dedicated unification engines in the knowledge base machine that is to be developed in the second four-year stage (1984-87) of the project [Yokota 85] .

Acknowledgments

The authors wish to express the appreciation to the cooperation of Toshiba and Hitachi Ltd., and the many personnel involved for their assistance in the development and evaluation of Delta.

References

[Yokota 83] Yokota, M., Yamamoto, A., Taki, K., Nishikawa, H., and Uchida, S., "The Design and Implementation of Personal Sequential Inference Machine: PSI", *New Generation Computing, vol.1, no.2*, pp.125-144, 1983.

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

[Tanaka 84] Tanaka, Y., "MPDC: Massive Parallel Architecture for Very Large Database", *Proceeding of International Conference of Fifth Generation Computer Systems 1984*, pp.113-137, November 1984.

[Tanaka 84] Tanaka, Y., "MPDC: Massive Parallel Architecture for Very Large Database", *Proceeding of International Conference of Fifth Generation Computer Systems 1984*, pp.113-137, November 1984.

[Kitsuregawa 84] Kitsuregawa, M., et. al., "Architecture and Performance of Relational Algebra Machine", *International Conference on Parallel Processing*, 1984.

[Sakai 84] Sakai, H., Iwata, K., Kamiya, S., Abe, M., Shibayama, S., and Murakami, K., "Design and Implementation of the Relational Database Engine", *Proceeding of International conference of Fifth Generation Computer Systems 1984*, pp.419-426, November 1984.

[Todd 78] Todd, S., "Algorithm and Hardware for a Merge Sort Using Multiple Processors", *IBM Journal of Research and Development, 22* ,1978.

[Knuth 73] Knuth, D. E., Fundamental Algorithm, *The Art of Computer Programming*, vol.1, 1973.

[Smith 75] Smith,J.M.,Chang,P.Y., Optimizing the Performance of a Relational Algebra Database Interface, *Communications of the ACM*, pp568-579, October 1975.

[Yokota 85] Yokota, H., and Itoh, H. , "A Model and Architecture for a Relational Knowledge Base", *ICOT Technical Report No. TR-144*,1985.