ICOT Technical Report: TR-157

TR-157

Parallel Control Technigues for Dedicated Relational Database Engines

by
Masa-aki Abe, Takeo Kakuta
and
Hidenori Itoh

February, 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

Masa-aki Abe , Takeo Kakuta , Hidenori Itoh

Institute for New Generation Computer Technology (ICOT)

Mita Kokusai Building, 21F

1-4-28 Mita, Minato-ku, Tokyo 108 Japan

February, 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 a parallel control strategy is considered for improved response time by simulating the database machine in executing database operations using these engines in parallel.

1 Introduction

The Fifth Generation Computer Systems Project in Japan aims to develop a high-level knowledge information processing system including inference and knowledge base functions. In the first three-year stage (1982-84) of the project, the PSI (Personal Sequential Inference Machine) was developed from research on the inference function[1]. For the knowledge base function, a back-end type relational database machine called Delta, compatible with logic programming languages such as Prolog, was developed as the first step towards a knowledge base machine[2]. Delta possesses the following characteristics.

1. Facts from logic programming languages are stored in tables.
2. The logical command interface with the host machines is based on relational algebra level commands.
3. Delta has dedicated relational database engines for rapid execution of join and other operations that can involve high processing loads.

4. Delta has a hierarchical memory (HM) with a semiconductor memory besides its large capacity disk device.

In this paper we assume a back-end type relational database machine equipped with multiple dedicated engines for relational database operations. A simulation of the parallel execution of these engines in performing relational algebra operations and other operations was made , and actual and experimental data are used to evaluate response characteristics and consider a parallel control strategy for improved response time.

2 Execution Method for Relational Database Operations

Selection, projection (including elimination of duplicates), and join operations are relational database operations that can involve high processing loads. Realization of the join operation in particular, though it is a powerful operation combining two relations, has been cited as a major problem in relational database research[3][4].

In order to execute relational algebra operations efficiently, it is often advantageous to sort the object relations by key attributes in advance, thereby reducing processing time. This procedure is especially effective for the join operation. In order to sort at high speed, we utilized a pipelined 2-way merge sort algorithm and developed dedicated hardware called the sorter. In this algorithm, the data to be sorted is converted into a data stream and it can be sorted keeping up with the flow of the input data stream. Dedicated hardware called the relational algebra processing unit, located after the sorter, eliminates delays in executing relational algebra operations on the sorted stream.

In this section we discuss the operation of this dedicated relational database engine (abbreviated to engine below) [5].

2.1 Pipelined 2-way Merge Sort Algorithm

The pipelined 2-way merge sort algorithm was proposed by Todd[6]. When the number of records to be sorted is N, this algorithm can reduce the sorting order $O(N \times log_2 N)$ to $O(N)$[7]. The following equations hold for this algorithm:

$$\rho = log_2 N$$

$$size(M_i) = 2^i \times L$$

where :

N : record count

L : record length

ρ : number of processors

$size(M_i)$: memory volume M_i of the i-th processor

The sorting of relation R with length l by key attribute A is expressed as

$$\mathrm{S}_A(R(l))$$

or simply

$$\mathrm{S}(l)$$

and the resultant relation is expressed as R.

2.2 Join Operation

As mentioned above, the join operation is executed with greater efficiency if the two relations are sorted by key attributes (key fields) in advance. If the two relation tuple (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 followed by sorting can be executed in $O(N_1 + N_2)$.

Let R_1 and R_2 be relations with length l_1 and l_2, and θ be a condition on attributes A_1, A_2 of each relation , then the join operation of R_1 and R_2 is denoted by

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

or simply

$$l_3 \quad = \quad l_1 \bowtie l_2$$

where R_3 is the resultant relation with length l_3. The join operation can be executed by a relational algebra processing unit (abbreviated to RAPU below). It consists of two memories called Memory1 and Memory2, which store $\mathbf{R_1}$ (R_1 sorted by A_1) and $\mathbf{R_2}$ (R_2 sorted by A_2), and a processor, which selects the tuples satisfying the condition.

The join operation is executed in the following steps.

step1: Load relation R_1 into the sorter.

step2: Perform $\mathrm{S}_{A_1}(R_1)$ in the sorter.

step3: Store $\mathbf{R_1}$ in Memory1.

step4: Load relation R_2 into the sorter.

step5: Perform $S_{A_2}(R_1)$ in the sorter.

step6: Store $\mathbf{R_2}$ in Memory2.

step7: Once the first tuple of $\mathbf{R_2}$ has begun to be stored in Memory2, the $val(A_2)$ of that tuple, and the $val(A_1)$ of the first tuple of $\mathbf{R_1}$ stored into Memory1 are compared by the processor. Satisfying the condition $val(A_1)\ \theta\ val(A_2)$, the tuples are combined, or else the unnecessary attributes deleted and the tuples combined, and then output. The above procedure is repeated for each tuple in Memory1 and Memory2 in their order of arrival. Here, the terms $val(A_1)$ and $val(A_2)$ express the values of A_1 and A_2, respectively.

2.3 Selection Operation

In the selection operation, constants in conditions are assumed to be a relation (R_1) composed of an attribute. So it is executed in the same way as the join operation by deleting tuples in R_1.

3 A Dedicated Engine for Relational Database Operations

3.1 Hardware Configuration

The hardware configuration for the engine executing the relational operations described in section 2 is given in Figure 1. The engine is composed of the HM I/O controller, which handles I/O control between the engine and the HM which stores relations; the sorter, which sorts a relation by its key attribute; the RAPU, which executes relational algebra operations; and the engine controller, controlling the entire structure.

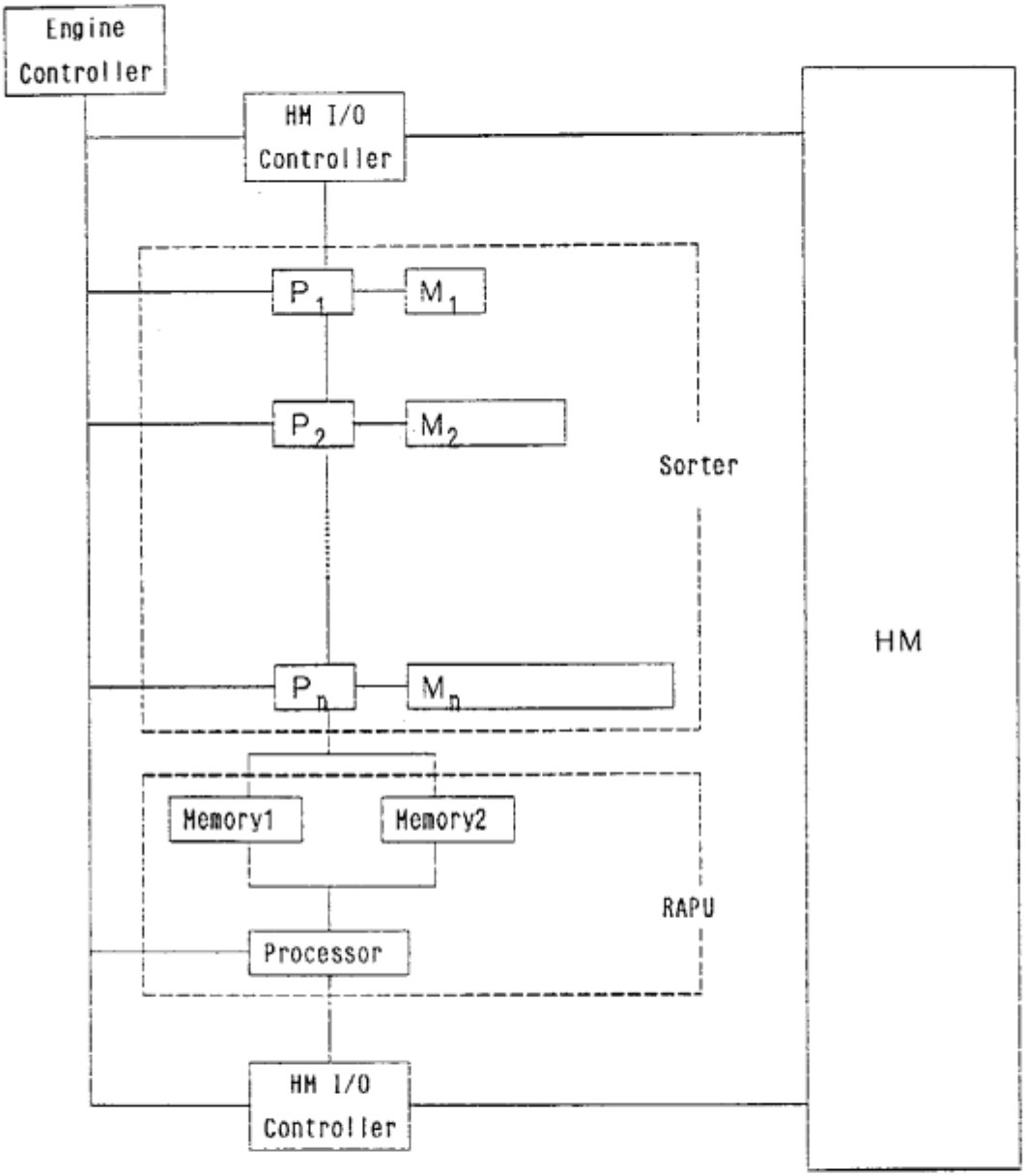
The sorter is composed of n levels of processors with memory units. Memory capacity doubles with each higher processor level. The sorter currently implemented has the following specifications

$$\rho = 12$$

$$size(M_{12}) = 64K\ bytes$$

This means that the sorter is able to sort up to 64K bytes of data at one time. And N_{max}, the maximum number of tuples the sorter is able to process, is given by the following equation.

$$N_{max} = min(2^{12},\ \lfloor size(M_{12}) / L \rfloor)$$



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

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

Figure 1. Hardware Configuration

RAPU is composed of the two memories (Memory1 and Memory2) storing sorted relations , and a processor which selects output tuples satisfying the condition. Currently , each of these memories is 64K bytes, enabling join and selection operation for a pair of up to 64K bytes data bundles (a pair of groups of up to 4,096 tuples).

3.2 Processing Method for a Large Amounts of Data

In section 3.1, it was stated that the sorter is able to sort 64K bytes of data at one time, and that RAPU is able to execute join and selection operations at one time for a pair of 64K bytes data bundles. This section describes the operation processing method used for large amounts of data exceeding these levels.

3.2.1 Sort Operation for Large Amounts of Data

When 64K bytes $< l \leq$ 128K bytes, the first initial 64K bytes of data are sorted by the sorter, and then stored in Memory1. The remaining (l - 64)K bytes of data are sorted by the sorter, and then stored in Memory2. The processor of RAPU merges them to generate the result.

When $l >$ 128K bytes, sorting is accomplished by the following steps.

1st step: The engine receives a pair of 64K-byte data bundles in buffer BUF0 from HM at a time. The sorter and RAPU sort pairs of 64K-byte data bundles, and then output 128K-byte units alternately to buffer BUF1 and BUF2 on HM.

Sth step($S = 2, 3, \ldots, \lceil log_2(l \,/\, 128K\ bytes) \rceil - 1$): At the $(S-1)th$ *step*, the data sorted in $(64 \times 2^{i-1})$K-byte units is stored in buffers BUFa and BUFb. The engine receives a pair of $(64 \times 2^{i-1})$K-byte data bundles in BUFa and BUFb from HM at a time. The RAPU alone sorts pairs of $(64 \times 2^{i-1})$K-byte data bundles by merging, and then outputs (64×2^{i})K-byte units alternately to buffer BUFc and to BUFd. Where S is even, a = 1, b = 2, c = 3, d = 4, and where S is odd, a = 3, b = 4, c = 1, d = 2.

$\lceil log_2(l \,/\, 128K\ bytes) \rceil th$ step: RAPU sorts a pair of data bundles in the two buffers created in the previous step by merging, and then outputs the result into BUF5.

Figure 2 shows sort operation for (64 × 8)K bytes of data.

When the data of length l is sorted, the total buffer capacity required is $3 \times l$, including input buffer BUF0, working buffers BUF1, BUF2 (BUF3, BUF4), and output buffer BUF5.

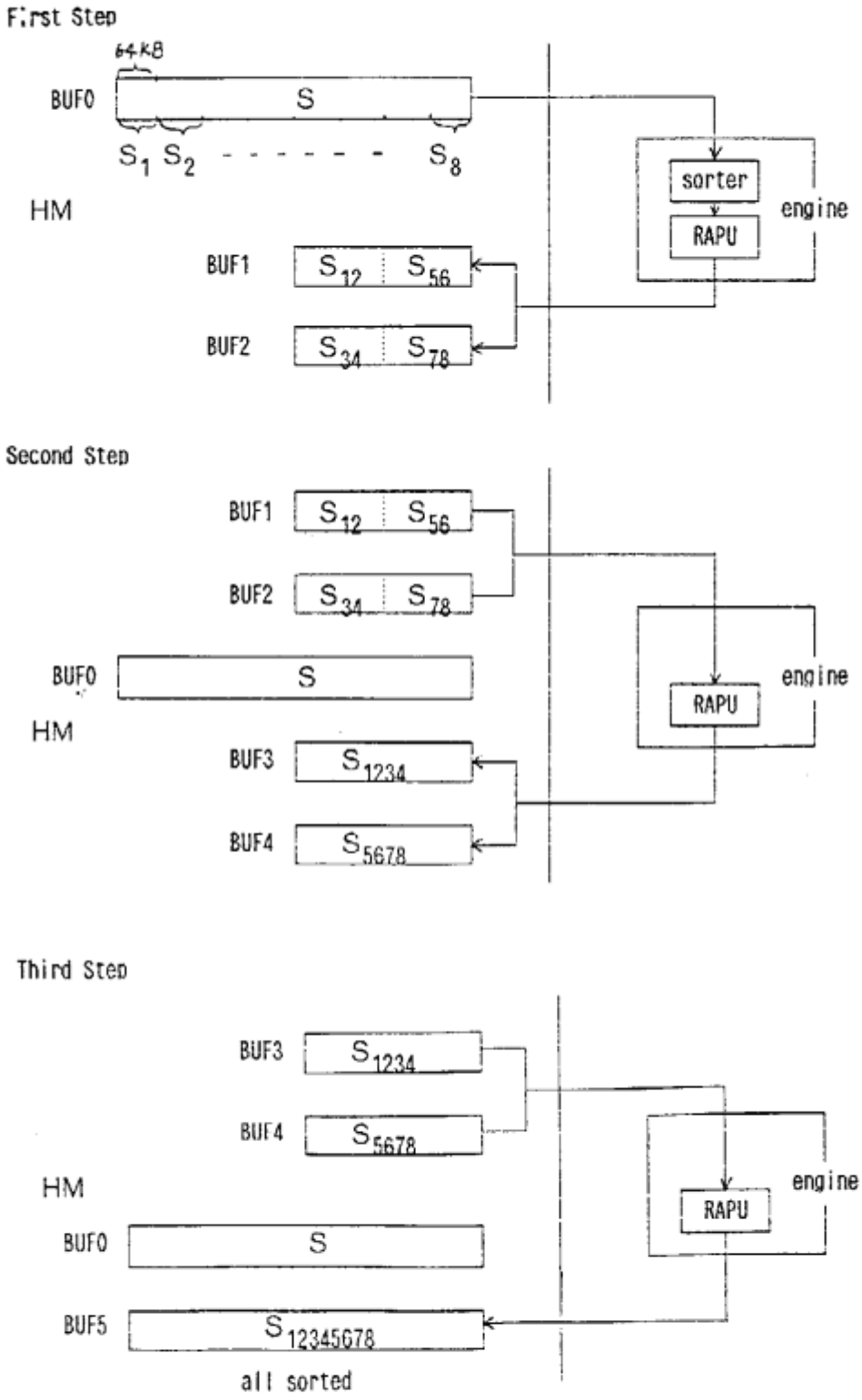


Figure 2. Ececution Example for Sort Operation

3.2.2 Join Operation for A Large Amount of Data

If $l_1 \leq$ 64K bytes and $l_2 \leq$ 64K bytes, the sorter and RAPU can execute the join operation at one time. If the size of either of the two relations exceeds 64K bytes, the following algorithm is used:

for $i = 1$ **to** $\lceil l_1 \ / \ 64K\ bytes \rceil$

The engine receives SR_{1i} from HM, sorts it, and stores it in Memory1

for $j = 1$ **to** $\lceil l_2 \ / \ 64K\ bytes \rceil$

The engine receives SR_{2j} from HM, sorts it, and stores it in Memory2. When the first tuple starts getting stored into Memory2, the tuples in Memory1 and Memory2 are compared in order of arrival, and only those satisfying the condition are output.

In this algorithm, SR_{ij} expresses the j-th sub-relation of relation R_i $(i = 1, 2)$, which is divided into 64K-byte segments.

4 Parallel Execution Method for Relational Database Operations

Section 4 discusses the parallel execution method by which multiple engines are used to execute relational database operations in parallel.

4.1 Data Division Sort Operation

When a large amount of data is sorted, the data is divided into a number of data segments equal to the number of inactive engines, and each data segment is then assigned to an engine to be executed in parallel. That reduces processing time considerably. We call this procedure the data division sort operation. It is executed in the following steps.

*step*1: Let m be the number of inactive engines and l be the length of the data to be sorted. Then in order to make m engines execute in parallel, the data is divided into m data segments and each data segment is stored into one of m buffers. The size of each buffer is l/m.

*step*2: The data segments in the buffers created in *step*1 are assigned to m engines on a 1:1 basis, then sorting is executed in parallel , and the results are output to the m buffers. Let $n \leftarrow m$.

*step*3: The number of sorted data segments is n. Gather up two of them, and make $\lfloor n/2 \rfloor$ pairs, $P1, P2, P3, \ldots, P_{\lfloor n/2 \rfloor}$. The i-th pair P_i $(1 \leq i \leq \lfloor n/2 \rfloor)$ consists of the sorted data segment S_{i1}

(length l_{i1}) and the sorted data segment S_{i2} (length l_{i2}). A inactive engine sorts P_i by 2-way merge and creates a new sorted data segment (length $l_{i1} + l_{i2}$). This is executed in parallel by inactive engines.

step4: Let $n \leftarrow \lceil n/2 \rceil$. If n equals 1 then stop, else goto *step3*.

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.

4.2 Data Division Join Operation

For two relations with sizes 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. Join operation $l_1/m \bowtie l_2$ is executed by a single engine. This is called a data division join operation. It is effected by the following steps.

step1: In order to make m engines execute in parallel, the relation with size l_1 is divided into m data segments, and each data segment is stored into one of m buffers in HM.

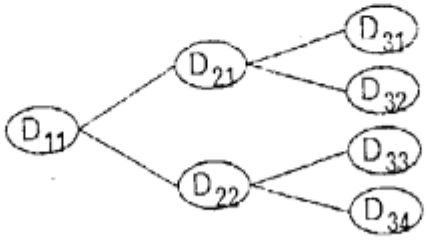
step2: The join operation $l_1/m \bowtie l_2$ is executed in parallel by m engines.

step3: The execution results from each engine are output to the output buffer.

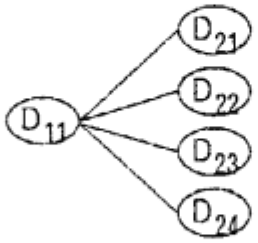
If the selection operation is executed in parallel by m engines in the same way as the join operation, we call this procedure a data division selection operation.

4.3 Data Division Subcommand Tree

The data division sort and data division join (selection) operations are converted into subcommands with tree structures and then executed. We call these trees data division subcommand trees. 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}. A binary tree is formed by the 4-DDST of the sort operation S(8M bytes), as shown below:



Nodes $D_{31} \sim D_{34}$ correspond to S(2M bytes), and nodes D_{21} and D_{22} correspond to the merge processing for the already-sorted pairs of 2M-byte data bundles. D_{11} corresponds to the merge processing for the already-sorted pair of 4M-byte data bundles. In general, the m-DDST of join operation $l_1 \bowtie l_2$ is given by the following tree of depth 2:



D_{11} corresponds to the dummy node for synchronization, and D_{21} and D_{22} correspond to the processing for $l_1/m \bowtie l_2$.

The m-DDST operation is executed from leaf to root. When execution at a node is completed, the symbol CM is affixed above the arc from that node to the next node. If a node has the CM symbols on all arcs extending to it, then execution at that node begins. We call this node an execution-enabled node.

5 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 section discusses internal execution control of the database machine for the host machine inquiries and strategies for the parallel control of the engines.

5.1 Relational Database Machine Block Diagram

The block diagram for the relational database machine is given in Figure 3.

The relational database machine is composed of an interface unit, a control unit, a schedule unit, and an HM controller, along with m^T engines. The interface, control and HM control units consist of a single CPU, and we envisage the engines as attached processors. The HM control unit includes a large-capacity external memory disk device (HM disk).

1. Interface Unit (IU)

   IU receives a query command C from a host machine, and sends it to the control unit queue. If IU receives the processing complete notice for command C from the control unit, IU send directions

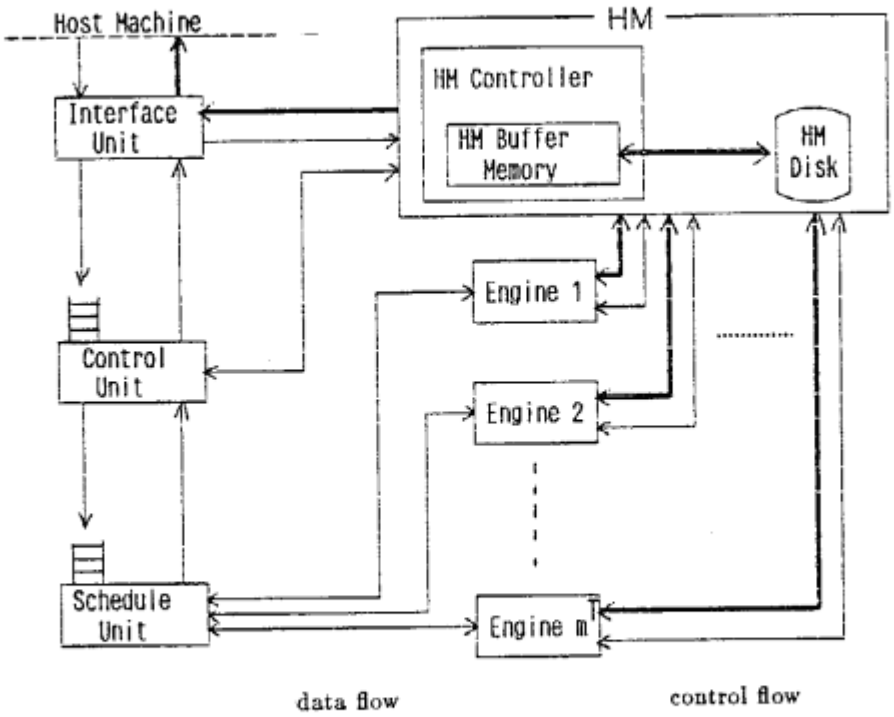


Figure 3. Block Diagram of Relational Database Machine

for output of resultant relations to HM, receives them from HM, and outputs them to the host machine.

2. Control Unit (CU)

CU takes commands $\mathbf{C^i}$ from the queue in sequential order. It analyzes it and translates it into internal commands $C_1^i, C_2^i, \ldots, C_n^i$ such as sort operations and relational algebra operations. It sends execution directions for staging the object relations used by internal commands in the buffer memory and for preparing buffer memory of the resultant relations to HM. It sends these commands to the schedule unit queue. If CU receives all the processing complete notices for $C_1^i, C_2^i, \ldots, C_n^i$ from the schedule unit, it sends the processing complete notice for $\mathbf{C^i}$ to IU.

3. Schedule Unit (SU)

SU takes internal commands from the queue following the engine parallel control strategy (EPCS). By observing the state of active and inactive engines, SU decides whether to divide data or not for the internal commands taken from the queue. If division is required, it determines the number of data segments for each command, sends execution directions for dividing data to HM, and converts it into DDST by EPCS. SU determines which selected commands and execution-enabled nodes will be assigned in what sequence to which engines. It sends execution directions for the selected commands and nodes to assigned engines. If SU receives the (root) processing

completion notice for the internal command (converted into DDST) from an engine, it sends the notice to CU.

4. Engines

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

5. HM Control Unit

   The HM control unit handles the staging of object relations, division processing, and preparing the buffer memory for resultant relations under direction of the CU and the SU. The replacement algorithm between the buffer memory and the disk utilizes the LRU (least recently used algorithm).

5.2 Engine Parallel Control Strategy

We consider three engine control strategies for the schedule unit below. At a certain time t, there is a certain number m of inactive engines. The total number of engines is m^T.

1. No division data FIFO strategy

   The commands are executed by one engine each without dividing the data. m commands are taken from the queue on a FIFO basis. They are performed by m engines in parallel.

2. Data division FIFO strategy

   Command assignment to m engines is processed in priority as follows:

   (a) Commands which already have been converted into DDSTs and not executed to their roots are referred to as unexecuted DDSTs. If there are unexecuted DDSTs at time t, SU assigns execution-enabled nodes of the highest FIFO priority command to m engines.

   (b) If there are still m^* $(0 < m^* \leq m)$ inactive engines , SU takes command C_j^i with the highest FIFO priority from the queue and converts it into m^*-DDST. The leaves of the m^*-DDST are executed by m^* engines in parallel.

3. Maximal data division FIFO strategy

   In the previous strategy, command Ci is converted into m^*-DDST, but in this strategy, it is converted into m^T-DDST. The m^* leaves of m^T-DDST are executed by m^* engines in parallel.

5.3 Objective Functions

We consider three functions as objective functions

1. Average response time

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

2. Average engine availability ratio

   The percentage of time that m engines are operating.

3. Average HM memory utilization volume

   Average memory volume needed in HM for executing commands by engines.

6 Evaluation Result

Section 6 presents the response characteristics for parallel engine execution in the relational database machine discussed in Section 5.

6.1 Simulation Parameters

The simulation parameters are as follows.

1. Resource Parameters

   (a) The interface unit, control unit, schedule unit, and HM control unit are located in a single CPU with a capacity of 5 MIPS. The engines are attached processors.

   (b) We consider that $m^T = 16$ and $m^T = 8$ for the total number of engines.

   (c) HM buffer memory size is 128M bytes.

   (d) The data transfer rate between the HM memory and the engines, and between the HM memory and the HM disk, is 3M bytes/seconds.

   (e) There are two channels between the HM memory and each engine for input/output.

2. Processing time parameters

   The processing time used here is the actual time measured on Delta.

   (a) Execution time for a relational algebra operation on one engine.

   (b) Parallel execution time for a relational algebra operation on up to 4 engines.

   (c) Staging time for an object relation from the HM disk to the HM buffer memory, division time for a relation, and time for preparing a buffer memory for a resultant relation.

The processing times indicated below are estimated from actual measured times, or calculated from estimation of dynamic step counts.

(a) Processing time in the control unit.

(b) Processing time in the interface unit.

(c) Parallel execution time for a relational algebra operation on more than 4 engines.

(d) I/O processing time between the HM buffer memory and HM disk in executing a relational algebra operation.

Processing times other than these are ignored.

3. Parameters of the query commands sent from the host machine.

(a) Command arrival is assumed to be Poisson arrival.

(b) Sort, join, selection commands arrive with equivalent frequencies.

(c) Object relation size ranges randomly from 1M bytes to 16M bytes.

6.2 Response Characteristics

The simulation results under the above parameters are presented in Figures 4 through 9. In these figures the continuous lines show performance under strategy 1, the dashed lines that under strategy 2, and the chain lines performance under strategy 3.

6.3 Discussion

1. Average response time

Figure 4 shows that when traffic rate is low, average response time under strategy 3 is lowest, higher under strategy 2, and highest using strategy 1, but beyond $6 \sim 7$ ($\times 10^2/hour$) average arrival ratio, average response time increases abruptly and the previous relation between the strategies is inverted. This means that when traffic rate is low, execution by dividing data into more data segments increases efficiency, but when traffic rate is high, execution without dividing data is more efficient. The corresponding curves for $m^T = 8$ show the same general tendency, but they are successively displaced back along the horizontal (average arrival ratio) axis.

2. Average availability ratio

Figure 6 shows that when traffic rate is low, the average engine availability ratio under strategy 2 is highest, lower under strategy 3, and lowest using strategy 1, but when traffic rate is high, the

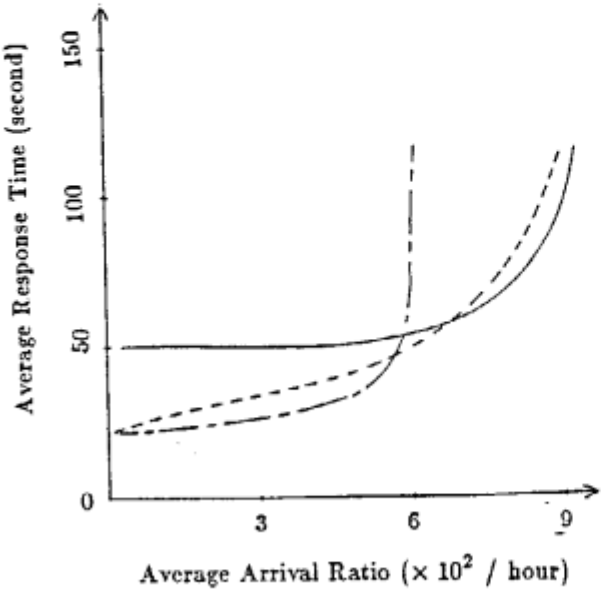


Figure 4. Average Response Time ($m^T = 16$)

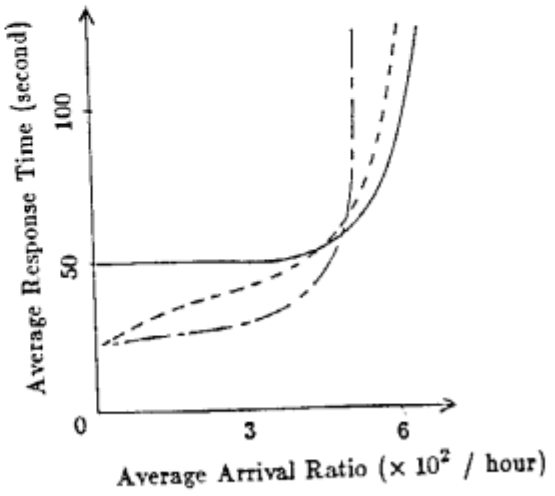


Figure 5. Average Response Time ($m^T = 8$)

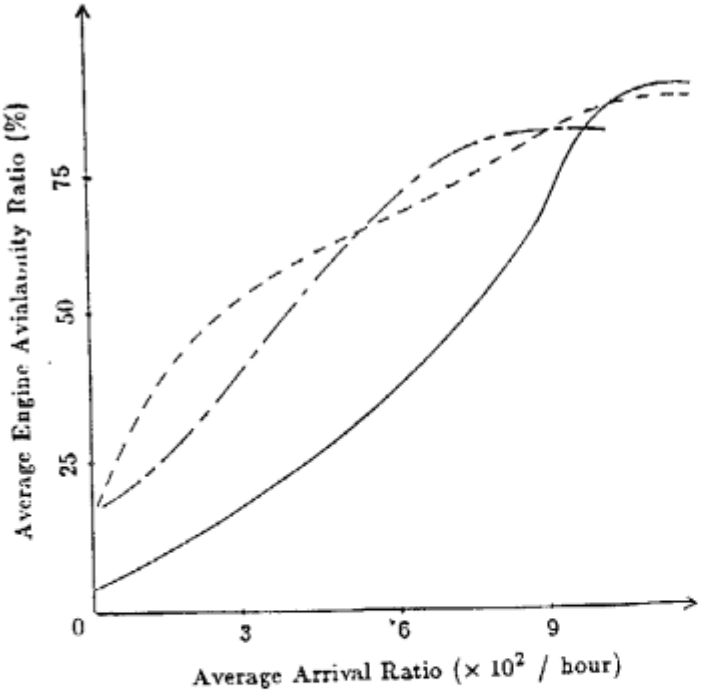


Figure 6. Average Engine availability ratio ($m^T = 16$)

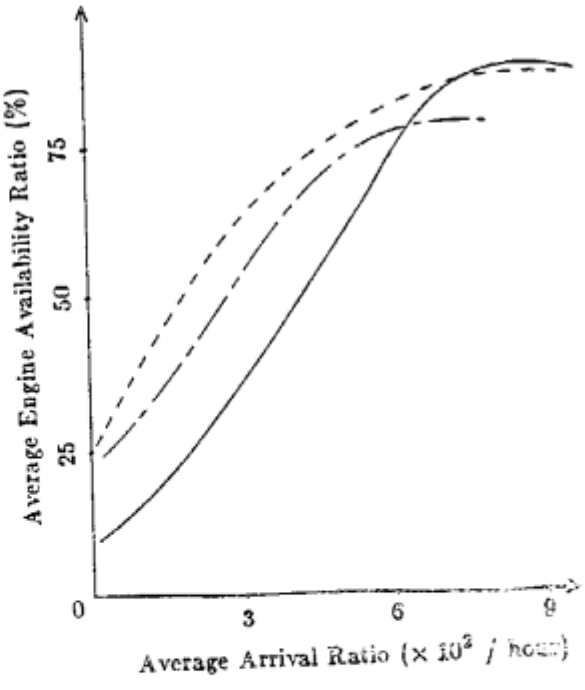


Figure 7. Average Engine Availability Ratio ($m^T = 8$)

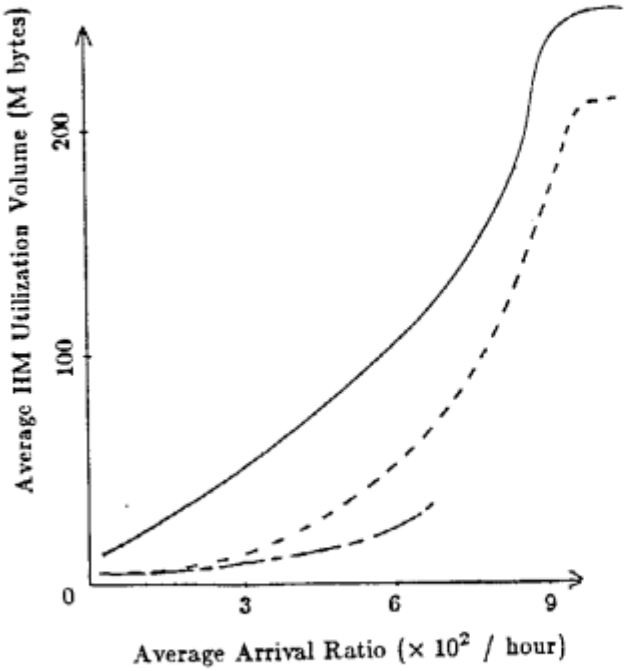


Figure 8. Average HM Utilization Volume ($m^T = 16$)

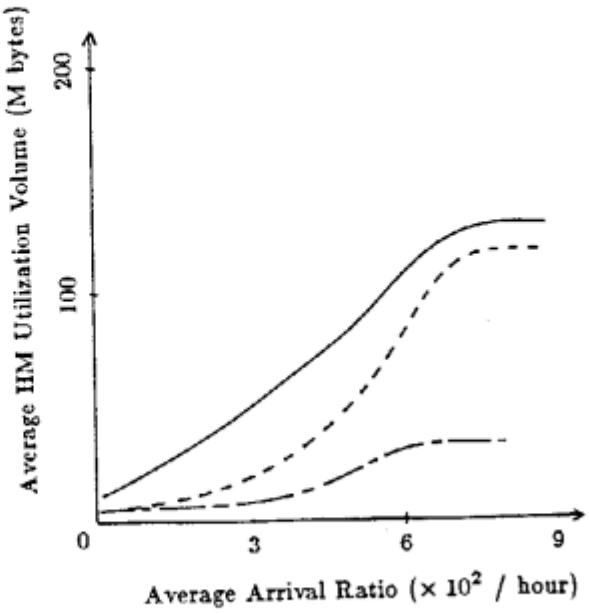


Figure 9. Average HM Utilization Volume ($m^T = 8$)

increasing rate of availability ratio under strategy 1 is high and strategy 1 surpasses strategy 2 and 3. Beyond average arrival ratio $8 \sim 10$ ($\times 10^2 / hour$), the curves under these strategies reach peaks, and then fall a little. This means that frequent I/O processing between the disk and the memory results in intermittent execution of the engines. The corresponding curves for $m^T = 8$ show the same general tendency.

3. Average HM memory utilization volume

Figure 8 shows that the average HM memory utilization volume under strategy 1 is highest, lower strategy 2, and lowest using strategy 3 regardless of traffic rate. This means that execution by dividing data into more data segments requires less memory volume. The reason for this is that buffer memories for the already-executed nodes in the data division subcommand trees can be opened, while buffer memory for execution without dividing data cannot be opened untill execution is completed. The higher the traffic rate, the steeper the slopes of the curves under these strategies. But at a certain traffic rate, the curves reach peaks and are parallel to the horizontal axis. This means that the engines are operating at full capacity at that traffic rate.

7 Conclusion

We assumed a back-end type relational database machine equipped with multiple dedicated relational database engines. We evaluated the response characteristics and considerated the parallel control strategies.

By simulation we found that when traffic rate was low, the strategy of dividing data into more data segments brought less response time with higher engine availability ratio, but when traffic rate was high, the opposite was the case. Clearly, different strategies are required for different traffic rates. For the utilization HM memory volume, we found that the strategy of dividing data into more data segments required less memory volume.

In this simulation we assumed a relational database machine based on Delta, but we think that the simulation results can be applied to a general back-end type database machine with dedicated engines using the parallel method in this paper.

We are plan to extend this research by investigating parallel control techniques for dedicated unification engines in the knowledge base machine that is going to be developed in the second four-year stage (1984-87) of the project[8].

Acknowledgments

Delta was developed with the cooperation of Toshiba and Hitachi Ltd. The authors thank Mr. H. Yasuo of Toshiba and Mr. H. Ishikawa of Hitachi for their useful discussions, and Mr. Y. Mitomo of the Japan systems corporation for his coding the simulation program.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

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

[7] Knuth, D. E., Sorting and Searching, *The Art of Computer Programming*, vol.3, 1973

[8] Yokota, H., and Itoh, H. , :A Model and Architecture for a Relational Knowledge Base", *ICOT Technical Report No. TR-144*, to appear in The 13th International Symposium on Computer Architecture, November 1985.