TR-284

Parallel Control Technique and Performance of an MPPM Knowledge Base Machine

by
H. Sakai, S. Shibayama (Toshiba), H. Monoi
Y. Morita and H. Itoh

June, 1987

Parallel Control Technique and Performance of an MPPM Knowledge Base Machine

Hidetoshi MONOI, Yukihiro MORITA, Hidenori ITOH
ICOT Research Center
Hiroshi SAKAI, Shigeki SHIBAYAMA
Toshiba R & D Center

June 13, 1987

Abstract

This paper describes parallel control techniques and performance evaluations by simulation for the knowledge base machine (KBM) using the multiport page-memory (MPPM) and unification engine (UE).

Including recursive definitions, relational knowledge base retrieval requires repeated application of the unification-join (U-Join) operations on term relations. It also requires each U-Join to be dynamically scheduled, observing the result of the previously executed U-Join.

We investigated control techniques to execute the query process of the relational knowledge base effectively in the KBM using the MPPM and the UE. For parallel execution of coarse grain operations, each operation is decomposed to concurrently executable fine grain operations by partitioning input data sets. We identified a number of features depending on processor allocation strategies. In particular, we found that query processing resulting in repetitions of the retrieval operation, by a certain control strategy, increases the amount of the total data input to highly parallel UEs and lowers processing efficiency.

1 Introduction

Aiming to realize high-speed knowledge processing, various inference machines are being researched actively. In the inference machine, the primary memory is the workplace to store and manipulate knowledge represented by a certain data structure at a high speed. Consequently, the cost of such a memory is high [1-3]. However, as the volume of knowledge to be processed becomes large, storing all of the knowledge in the primary memory becomes increasingly difficult and costly. Moreover, as in the usual database, common knowledge that should be shared by users. Therefore, our aim is to establish a mechanism capable of holding a large volume of knowledge in low-cost, large-capacity secondary storage and loading necessary knowledge into the primary memory only when required.

To satisfy this requirement, Japan's Fifth Generation Computer Project is engaged in researching a knowledge base machine (KBM) incorporating this mechanism. In the initial stage of the project, a relational database machine, Delta [4], was developed as the first step toward such a KBM. In the intermediate stage, we are developing a KBM with advanced retrieval functions.

To serve logic programming-based inference machines, the knowledge base model for storing knowledge should be able to handle both facts and rules that appear in logic programming. That is, the knowledge base model has to handle not only knowledge represented extensionally, like databases, but also intensionally represented knowledge that include a recursive definition. Thus the knowledge base model needs some inference mechanisms to handle the knowledge represented intensionally.

We have proposed a knowledge base model to realize the above requirement. The model represents knowledge in logical terms and stores a set of terms as a table (relation). The collection of these relations is called the relational knowledge base (RKB). We have also proposed a set of operations on the RKB, which is a natural extension of the relational algebra and can be used to perform inference on stored rules. The operations are called, collectively, retrieval by unification (RBU) operations. Basically, they are divided by extending an equality condition of relational algebra to the unifiability condition or unification operation.

We have also proposed a KBM architecture [5] in which RBU operations can be processed efficiently. We considered that the following two points are the keys to the architectural design.

(1) Reinforcement of the secondary storage system in terms of total I/O capacity and multiplicity of readout ports

(2) Incorporation of dedicated high-speed processors used in combination with the secondary storage system

The KBM architecture thus resulted in the combination of the multiport page-memory (MPPM) [6] for (1) and multiple unification engines (UEs) [7] for (2).

Roughly speaking, to answer queries to an RKB including recursive intensional definitions, the retrieval process must expand search trees from intensional definitions. This is done by repeatedly applying unification joins, an RBU operation, to term relations stored in the RKB.

We have already proposed a hardware algorithm and configuration of a UE [7]. This paper concentrates on the parallel control strategy for efficiently retrieving the RKB under the proposed architecture. We identified a number of features depending on processor allocation control strategies.

In particular, we found that query processing resulting in repetitions of a retrieval operation, by a certain control strategy, increases the amount of the total data input to highly parallel UE and lowers processing efficiency.

This paper describes a control method for parallel execution of query processing to remedy this, resulting from the simulation on the KBM architecture. Section 2 briefly describes the architecture of our KBM and each component of the machine. Section 3 describes the control methods used in the KBM. Section 4 describes the simulation results of the control methods described in section 3. Section 5 discusses the results.

2 Knowledge Base Machine Architecture

2.1 Hardware Configuration

Figure 2-1 shows the basic configuration of the KBM. The main components of the machine are the control processor (CP), multiport page memory (MPPM), unification engines (UEs), and disk systems (DKSs).

Knowledge is stored in the DKS as a collection of term relations. As shown in Figure 2-1, term relations are supplied from the DKS through MPPM. Term relations in the DKS should first be staged in MPPM.

The above configuration has the following advantages.

(1) Using the MPPM as disk cache memory for the DKS and connecting the DKS to the UEs through multiple ports, the I/O bottleneck between primary and secondary storage is reduced.

(2) By allowing the UEs to access the MPPM simultaneously, transfer bottlenecks are reduced when the UEs perform operations in parallel.

2.2 Multiport Page-Memory

Like a database machine, a KBM must sometimes repeatedly perform a set of simple operations on a large amount of data. Therefore, to balance the system, the architecture must realize not only high-speed internal retrieval but also high-speed data transfer, matching the internal retrieval, between the secondary storage and the retrieval processors. To exploit parallelism, each RBU operation is divided into concurrently executable fine-grain operations, which are distributed to multiple UEs. An interconnection structure ensuring smooth simultaneous access from multiple UEs to a large-capacity shared memory as a disk cache must therefore be used in the machine to support the parallelism.

For the purpose of a preliminary architectural discussion, we evaluated and compared three interconnection structures between the disk cache memory and multiple processors from the perspective of the conflicts for a transfer path. These interconnections are the time-shared common bus, crossbar switching network, and MPPM. The data transfer rate was assumed to be 50, 100, 150 MB/sec for the common bus and 5, 7, 10 MB/sec for the crossbar and MPPM. These rate were chosen because they are feasible in the same class of manufacturing technology. The memory module conflict was assumed to be 0.1. Figure 2-2 shows the results of the simulation.

An MPPM provides each processor with an independent transfer path. Processing capability improves in proportion to the number of processors. The common bus provides all processors with a single data transfer path, causing processing capability improvement to be limited by the data transfer capability. The crossbar cannot provide multiple transfer paths for access to the same memory module. It thus generates conflict for the transfer path though not to the extent of the common bus. It is, therefore, less effective than the MPPM.

The common bus is the smallest unit of hardware. It does not, however, match the performance requirement given above. The MPPM and crossbar switching network are on a similar scale [6]. The above discussion resulted in the MPPM.

2.3 Unification Engines

A UE is a dedicated processor that retrieves input term relations. Term relations are input to UEs in the form of a stream (a contiguous flow of terms).

A UE is connected to the MPPM through separate ports. It accepts input terms from the MPPM, executes retrieval concurrently within data transfer, and outputs the result concurrently to the MPPM. A UE has three ports, two of which are used to input terms of two relations simultaneously and the other to output the resultant terms. Figure 2-3 shows its configuration. The main components are two pipeline merge sorters, a pair generation unit, and a unification unit [7]. The pair generation unit receives an arranged sequence of input terms from the merge sorters, produces pairs of tuples which may be unifiable, and send them out to the unification unit. The unification unit checks the rigorous unifiability of each pair and outputs the resulting tuples.

The maximum number of terms that a UE can process at one time is the parameter of the merge-sorter implementation. This capacity is called the buffer size of a UE in the following discussion [8].

3 Knowledge Base Retrieval in the Knowledge Base Machine

3.1 Relational Knowledge Base and Query Processing

A relational knowledge base (RKB) assumes that knowledge is represented by tuples consisting of terms and contains sets of tuples in the form of relations (term relations). The operations on the term relations are called retrieval by unification (RBU) operations. The RBU operations, unification join (U-Join), and unification restriction (U-Restriction) operations were introduced.

As described in Section 1, retrieval in the RKB is realized by executing RBU operations on term relations. Figure 3-1 shows an example of a retrieval procedure for the knowledge base [5]. A Prolog program is stored in a two-attribute term relation, T, in which a tuple corresponds to a Horn clause. The first attribute stores the head and the second stores the body of a clause. A goal clause is also represented in another term relation. Retrieval is performed as follows. First, tuples within the term relation, T, whose head can unify with the goal are selected (U-restriction) and a term relation, T_0, is generated from the body and the goal. Then successive U-Join is invoked between relations T and T_0 to generate a new term relation, T_1. In general, a U-Join generates a new term relation, T_{i+1}, from relations T and T_i. The procedure ends when no more relations are generated by a U-Join. Terms whose body attribute is equal to nil are selected from each relation, T_i, and stored in a resultant relation, R.

The procedure for retrieval by repetition of a U-Join shown in Figure 3-1 is the typical query processing in the RKB. In this procedure, each U-Join step produces the next generation of term relations.

3.2 Parallel Execution of the Query Processing

In the KBM, queries are processed as shown in Figure 3-2. First, queries are compiled into a command sequence, which consists of coarse grain RBU operations on term relations stored in the RKB. To permit faster execution of such operations by exploiting parallelism, each must be decomposed by partitioning the input data set. Therefore each RBU operation is decomposed into concurrently executable operations and distributed among multiple UEs. Conceptually, each UE creates input streams from the input MPPM pages and an output stream to the output pages. While data is transferred through the streams, the UE performs the decomposed RBU operation allocated.

As shown in Figure 3-1, query processing repeats U-Joins on term relations stored in the DKS. The amount of data that must be transferred from the DKSs would increase prohibitively if DKSs were accessed for the permanent relation each time the U-Join was repeated. Therefore, the term relation processed for a query is held as long as possible in the MPPM until query processing is completed. When one operation is decomposed and executed concurrently, concurrent requests referring to the same page of the MPPM occur frequently. This access conflict can degrade the performance of the systems, unless the MPPM is incorporated.

3.3 Parallel Execution of RBU Operations

3.3.1 Decomposition of RBU operation

As described in the previous section, the RBU operations are decomposed into multiple concurrently executable operations by partitioning input term relations. Figure 3-3 shows decomposition of the U-Join using the multiprocessor nested-loop algorithm [11]. We chose this method because we wanted to obtain the fundamental performance measure for the proposed architecture, and because the algorithm is simple and easily is held. Each part, on a page basis, of a relation is called a segment. All pairs of segments of an inner and an outer relation are obtained, and each pair is allocated to a UE as input operands of a decomposed operation.

The procedure of decomposing the RBU operation is called the division process, and each decomposed operation is called a split operation.

Split operations generated as shown in Figure 3-3 can be expressed as follows. Assume that $\mid T \mid$ represents the size of a term relation, T, s the size of a segment, and $\mid T \mid //s$ the number of segments of T (provided that T is segmented along the whole scope of every tuple). If, under this convention, a U-Join is to be applied to term relations Q and R of which the segment sizes are s_1 and s_2, the number of split operations generated

$$\mid Q \mid //s_1 \times \mid R \mid //s_2$$

If $q_i (1 \leq i \leq \mid Q \mid //s_1)$ represents the segment of Q, and $r_j (1 \leq j \leq \mid R \mid //s_2)$ represents the segments of R, the following equation holds:

$$Q \overset{\bowtie}{\diamond} R = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} q_i \overset{\bowtie}{\diamond} r_j$$

where $m = \mid Q \mid //s_1$ and $n = \mid R \mid //s_2$

Here, we consider the total amount of data input to UEs to execute the whole split operation. As a UE has two independent ports and overlapping segment input, the time required to input two segments is bound by a larger segment input time. We can take the size of a segment as a measure of the input time. Thus, the input size of a split operation is s_1 if $s_1 \geq s_2$. If term relations Q and R are divided and U-Join is applied, segments are input for execution of split operations. Therefore, the total input amount, I_D, is given

$$\begin{aligned} I_D &= s_1 \times (\mid Q \mid //s_1) \times (\mid R \mid //s_2) \\ &= \mid Q \mid \times(\mid R \mid //s_2) \end{aligned}$$

by the definition of $//$, $s_1 \times \mid Q \mid //s_1 = \mid Q \mid$

If a U-Join on Q and R is done with one UE, $\mid Q \mid$ and $\mid R \mid$ need be input only once. Thus, the input amount, I, is $\mid Q \mid$ if $\mid Q \mid \geq \mid R \mid$. Therefore, I_D is $\mid R \mid //s_2$ times of I. This means that the total amount of input data increases in proportion to the number of segments.

This phenomenon indicates that if the execution time of a UE is proportion to the data transfer time (which requires a sophisticated algorithm), the decomposition of an operation prevents the execution time from improving linearly as the number of UEs. This problem also arises in join operations in a relational database. However, decomposition is necessary to exploit UE parallelism. The problem, then, is to find a method that suppresses the increase in total amount of data input, while keeping the UEs doing useful work as much as possible.

Figure 3-4 shows the basic scheme for allocating split operations to parallel UEs. When a certain number of UEs completes execution of a split operation, a new division process is executed. New split operations are repeatedly generated until no more results are produced by any of the UEs.

We considered two operation division methods for the above query processing procedure. One is the single page at a time (SP) method that generates a fixed-size split operation from fixed-size segments. The other is the multiple pages at a time (MP) method that adjusts the size of the split operations according to the size of the next input relations. The next two sections describe these method.

3.3.2 SP method

The SP method fixes the size of a segment of a term relation. In the proposed KBM architecture, the fixed size in naturally the MPPM page size.

To process a query, U-Joins between permanent and temporary relations are executed repeatedly. Therefore, if a UE has to wait for the completion of the generation of next split U-Join operations to other UEs, the wait time increases and the UE utilization ratio decreases. The SP method aims to avoid this wait time by fixing the input data size of a split operation and invoking a division process every time new results are produced by split operation in a data-driven manner.

Query processing by the SP method proceeds as explained below. Assume that Pr represents a permanent term relation in a knowledge base, $Tr_i (i \geq 0)$ a temporary term relation generated by the U-Join of each generation, and p the size of an MPPM page. Then, Pr and initial temporary term relation Tr_0 are divided into the following sets of segments:

$$\begin{aligned} \{Pr\} &= \{v_i \mid 1 \leq i \leq \mid Pr \mid //p\} \\ \{Tr_0\} &= \{w_{0,i} \mid 1 \leq i \leq \mid Tr_0 \mid //p\} \end{aligned}$$

By this division, U-Join is decomposed to the split operations as follows:

$$v_i \underset{\diamond}{\bowtie} w_{0,j} \text{ where } 1 \leq i \leq \mid Pr \mid //p \text{ and } 1 \leq j \leq \mid Tr_0 \mid //p$$

The next split operations are generated and put into the UE allocation queue by detecting all previous split operations that yielded results. Assume that n UEs are used and that t^m represents the output from the mth UE for one split U-Join operation. Then, the number of generated split operations is

$$\mid t^m \mid //p \times \mid Pr \mid //p.$$

This method keeps the UE utilization high, because many split operations are generated and kept in the UE allocation queue. However, the method also increases the number of generated split operations and the total amount of input data. Conversely, if the MPPM page size is disproportionately large, the number of generated split operations is reduced because of the small number of (large) pages, and sufficient parallelism cannot be obtained for the number of UEs. Determining the optimum MPPM size for the SP method is very difficult.

3.3.3 MP method

The MP method adjusts the segment size according to the number of UEs (the degree of parallelism) and according to the size of the input term relations. Since the MPPM can only be accessed in units of pages, the adjusted segment size must be a multiple of the MPPM page size.

When a split operation is completed, output tuples are input to the next U-Join. The segment size must be determined for the division process. If the division process is executed for each completion of UE, and the degree of parallelism is high, this division method would be the same as the SP method. In the MP method, output tuples from split operations are collected once, then the division process is invoked when the number of idle UEs exceeds a certain number. The tuples belonging to different generations can be collected together. This method inherently decreases the UE utilization ratio compared to the SP method, because a UE is not allocated to the next split operation immediately it becomes free. To remedy this situation as far as possible, UEs that become idle are detected in real time.

The division process is explained below. Assume that Tr, representing the collection of output tuples at one time for the division process, is invoked and that pr represents a permanent term relation. Assume also that s_p and s_t represent the segment sizes of Pr and Tr. By the MP method, the segment size. s_p and s_t must be determined by the division process is determined. Then s_p and s_t are determined so that they minimize the total amount of input data to perform the split operations.

We consider the total amount of input data, I_D, for this method, if the degree of parallelism is n. Assuming that Pr is divided into l segments and Tr divided into m segments and assuming $n = l \times m$, I_D is computed as follows.

$$\begin{aligned} I_D &= n \times max(\mid Pr \mid /l, \mid Tr \mid /m) \\ &\geq n \times (\mid Pr \mid /l + \mid Tr \mid /m)/2 \qquad \text{The equation holds when } s_p = s_t \\ &= (m \mid Pr \mid + l \mid Tr \mid)/2 \\ &\geq \frac{\sqrt{m|Pr| \times l|Tr|}}{2} \qquad \text{The equation holds when } s_p = s_t. \end{aligned}$$

This inequation shows that $s_p = s_t$ to minimize I_D. The MP method divides $\mid Pr \mid \times \mid Tr \mid$ into n equal partitions, and determines s_p and s_t as the square root of this partition (i.e. $s_p = s_t = \sqrt{\frac{|Pr| \times |Tr|}{n}}$).

However, assuming that the UE's buffer size is b, s_p and s_t must be smaller than b. The whole tuples of Pr must be consumed in one division process to simplify division. When $\mid Pr \mid \times \mid Tr \mid \geq n \times b^2$ (especially when $\mid Pr \mid \geq n \times b$), s_p is determined maximally as b and the number of split operations generated is greater than n. In this case, st is determined as b when $\mid Tr \mid \geq b$ or as $\mid Tr \mid$ when $\mid Tr \mid \leq b$.

4 Performance of SP and MP Methods

A simulation was carried out to evaluate the performance. The simulation complied with the query processing procedure shown in Figure 3-4. The UE proposed by [7] was chosen as the basic hardware model. We first broke down the model to register transfer level design, refined several points for performance improvement and implemented it in software so that the simulated execution time exactly corresponds to that of a hardware implementation. According to the design, the UE's hardware clock period is assumed to be 200 nsec. Each for-byte port of the MPPM is assumed to match the clock period, resulting in a 20 MB/sec transfer rate. The capacity of the MPPM was infinite, that is, query processing was executed without I/O to and from secondary storage. Control overhead associated with software such as compilation of queries, management of processes, and division processes were not added to processing time because we wanted to clarify the effect of division methods.

4.1 Performance of SP Method

Figure 4-1 shows the elapsed time and the UE utilization ratio for various MPPM page sizes in the SP method. Elapsed time indicates the total execution time of the parallel execution for the query processing shown in Figure 3-1.

The number of split operations generated by the SP method varies according to the MPPM page size. To a certain extent, the larger the page size, the smaller the amount of input data and the shorter the processing time if the number of UEs is fixed. However, if the page size is enlarged excessively, not enough split operations for installed UEs are generated, the utilization of UEs is reduced, and processing time is prolonged. When the number of UEs is increased, the effects of parallel processing overcome the increase of the total amount of data input by dividing operations and reducing change in the elapsed time.

The variation of the elapsed time described above is confirmed by the utilization ratios. When the page size and number of UEs are both relatively small, utilization ratios remain high. However, the smaller the page size, the lower the utilization ratio.

Figure 4-1 shows that the optimum page size depends on the degree of parallelism. We can easily conjecture that the optimum page size also depends on the size of term relations processed in query processing. The number of split operations generated in the division process must be adjusted according to the degree of parallelism.

4.2 Performance of MP Method

Figure 4-2 shows the elapsed time and UE utilization ratio in the MP method. The page size has little influence on processing times when this method is used, because this method always generates split operations irrespective of the unit page size. Compared with Figure 4-1, the variation of the elapsed time is small in relation to the MPPM page size. This shows that the MP method adapts the number of split operations to the degree of parallelism and effectively avoids increasing the amount of input.

The UE utilization ratio is less than in the SP method because the MP method does not invoke the division process until a certain number of UEs becomes free, increasing the idle time of a UE. However, the decrease in utilization does not affect the elapsed time much, because (1) the MP method generates split operations as evenly as possible to make UEs work nearly synchronously, and (2) it suppresses the total amount of input.

5 Discussion

We found that both division methods caused the total amount of data input to UEs to increase in proportion to the minuteness of segmentation of processes, lowering the efficiency of parallel processing. Parallel processing gives almost no advantage if the degree of parallelism is low and split operations are generated by the SP method independent of the degree of parallelism. The MP method division processes eliminates wasteful split operations when the degree of parallelism is low. It thus improves the processing time in comparison with the SP method. However, the MP method is still unable to eliminate completely the increase of input caused by division processes. Figure 5-1 shows the effect of the degree of parallelism by implementing the MP method. (The speedup shown here is defined as [processing time with parallelism degree n]/[processing time with parallelism degree 1].) As shown in this figure, the performance improvement ratio does not improve in proportion to the degree of parallelism. This is because the MP method increases the number of split operations with in proportion to degree of parallelism, reducing the parallel query processing efficiency.

To improve the processing efficiency of the MP method when the parallelism is high, the increase in input caused by process division must be constrained. The division processes discussed in this paper always generate all pairs of segments as split operations. Some of the generated split operations produce no tuples that can be joined by unification, in other word, they produce no output. If these wasteful segment pairs can be eliminated when processes are divided, a fair amount of input can be avoided.

In the case of join operations on a relational database, hashing can be applied to eliminate a significant number of wasteful split operations [12][13]. Process division method can be used to generate split operations from clusters. In U-Join, as terms have variables that unify any corresponding counter parts, the elimination is not as straightforward as the mere equality checking. To remedy this, we have already found a term hashing method, which is reported elsewhere [15]. Our future research will investigate how to apply the hashing method to the division process in terms of eliminating wasteful split operations.

REFERENCES

[1] Onai, R., et al. "Architecture of a Reduction-based Parallel Inference Machine: PIM-R", New Generation Computing. OHMSHA, 3(2), June 1985

[2] Ito, N., et al., "The Dataflow-based Parallel Inference Machine to Support Two Basic Languages in KL1", in Proc. IFIP TC-10 Working Conference of Fifth Generation Computer Architecture. UMIST (Manchester), July 1985

[3] Moto-oka, T., et al., "The Architecture of a Parallel Inference Engine-PIE-", in FGCS '84, ICOT, November 1984

[4] Kakuta, T., et al., "The Design and Implementation of Relational Database Machine Delta", in Proc. Int. Workshop on Database Machine '85, March 1985

[5] Yokota, H., et. al., "A Model and an Architecture for a Relational Knowledge Base", In Proc. 13th Ann. Int. Symp. Computer Architecture, June 1986, pp.2-9.

[6] Tanaka, Y., "A Mültiport Page-Memory Architecture and A Multiport Disk-Cache System, New Generation Computing, OHMSHA, 2, February 1984, pp.241-260.

[7] Morita, Y., et al., "Retrieval-By-Unification Operation on a Relational Knowledge Base", in Proc. 12th Int. Conf. Very Large Database, August 1986, pp.52-59,.

[8] Itoh, H., et al., "Parallel Control Technique for Dedicated Relational Database Engines", in Proc. 3rd Int. Conf. on Data Engineering, February 1987, pp.208-215

[9] Valduriez, P., "Semi-Join Algorithms for Multiprocessor Systems", ACM Transactions on Database Systems, Vol.9, No.1, 1984, pp.133-161

[10] Boral, H., et al., "Processor Allocation Strategies for Multiprocessor Database Machines", ACM Transactions on Database Systems, Vol.6, No.2, June 1981, pp.227-254

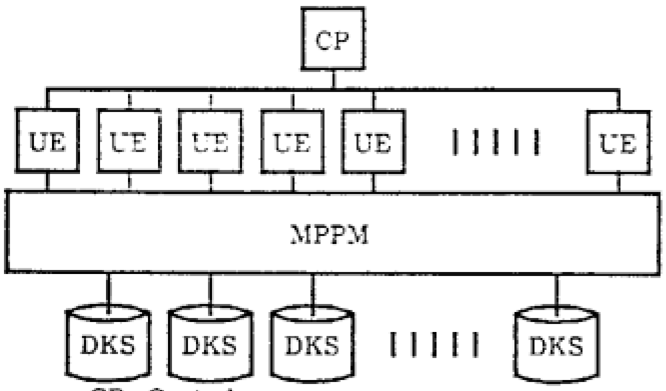
[11] Boral, H., et al., "Design Consideration for Data-flow Database Machines", in Proc. ACM-SIGMOD 1980 int. Conf. Management of Data, May 1980, pp.95-104

[12] Kitsuregawa, M., "Architecture and Performance of Relational Algebra Machine GRACE', University of Tokyo, Technical Report, 1983

[13] DeWitt, D., "Multiprocessor Hash-Based Join Algorithms", in Proc. Int. Conf. Very Large Database, 1985, pp.151-164

[14] Morita, Y., et al.,"Performance Evaluation of a Unification Engine for a Knowledge Base Machine", ICOT TR-204, 1987

[15] Morita, Y., et al., "Structure Retrieval via the Method of Superimposed codes", in Proc. the 33rd IPSJ Conference, 1986 (in Japanese)



CP: Control processer
UE: Unification engine
MPPM: Multiport page memory
DKS: Disk system

Figure 2-1 Knowledge base machine configuration

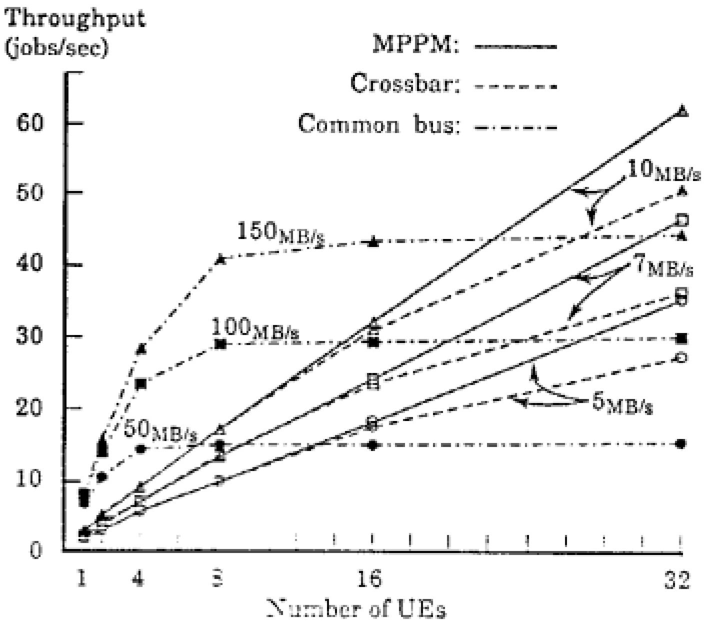


Figure 2-2 Comparison of interconnection structures

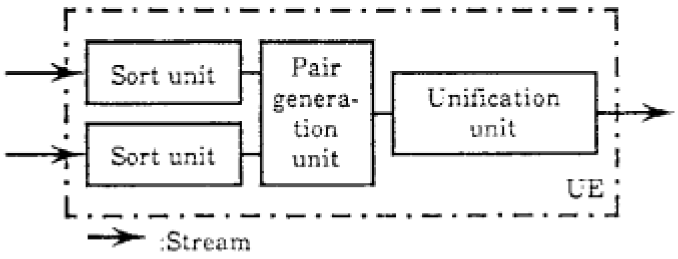


Figure 2-3 Configuration of UE

```
R ← $\phi$;
$T_0 \leftarrow \sigma_{head \diamond body}(T)$;
while $T_i \neq \phi$ do;
    begin;
        $R \leftarrow \sigma_{body = [\,]}(T_i) \cup R$;
        $T_{i+1} \leftarrow \Pi_{T_i \bullet head,\, T \bullet body}(T_i \; {}_{body}\overset{\bowtie}{\diamond}{}_{head} \; T)$;
        i = i + 1;
    end;
end;
```

where

T:	Permanent Relation	$\overset{\bowtie}{\diamond}$:	U-Join
R:	Resultant Relation	$\sigma_{a \diamond b}$:	U-Restriction
T_i:	Temporary Relation	Π:	Projection

▨ : Iteration of U-Join

Figure 3-1 Retrieval procedure in the relational knowledge base

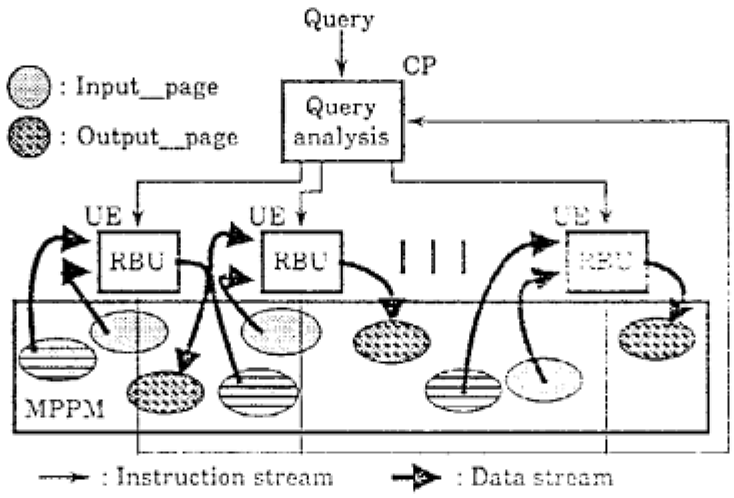


Figure 3-2 Query processing in the KBM

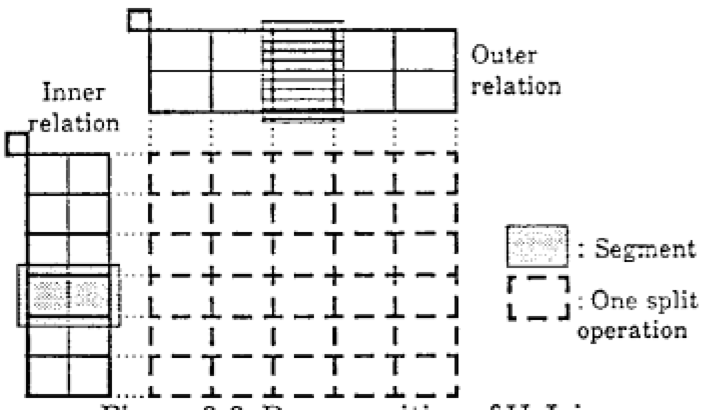


Figure 3-3 Decomposition of U-Join

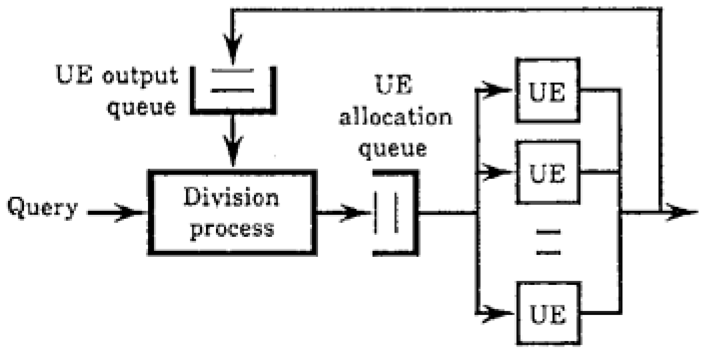


Figure 3-4 Basic control scheme for the parallel query processing

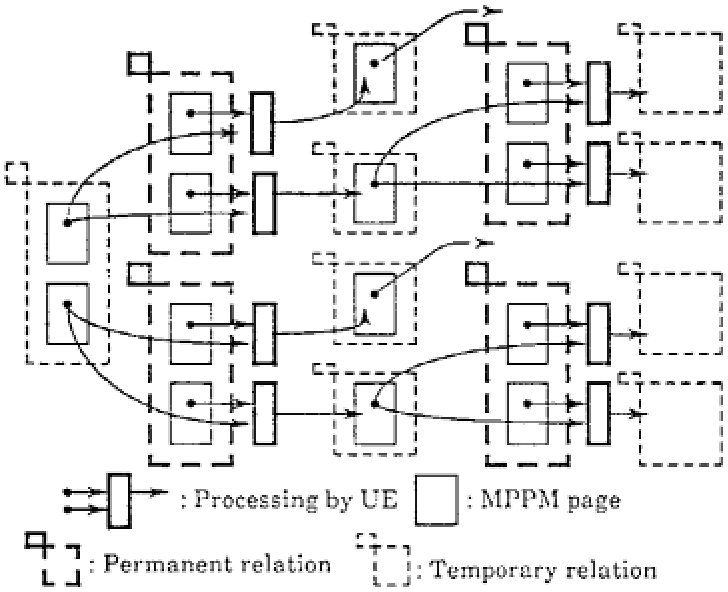


Figure 3-5 Parallel processing by the MP method

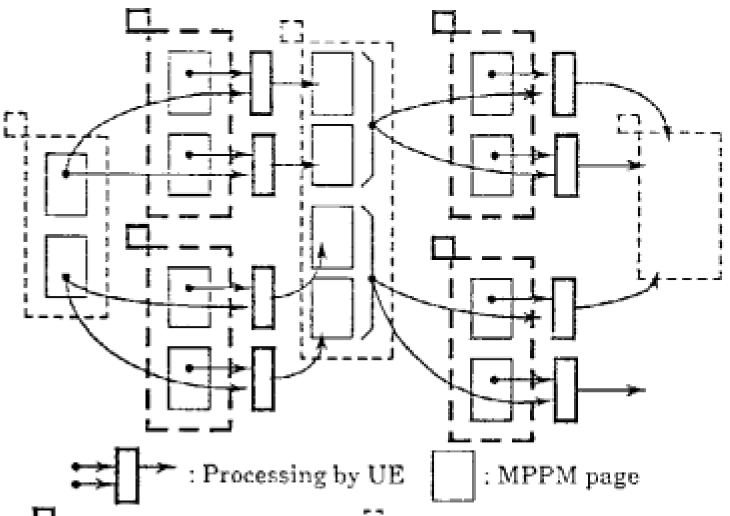


Figure 3-6 Parallel by the MP method

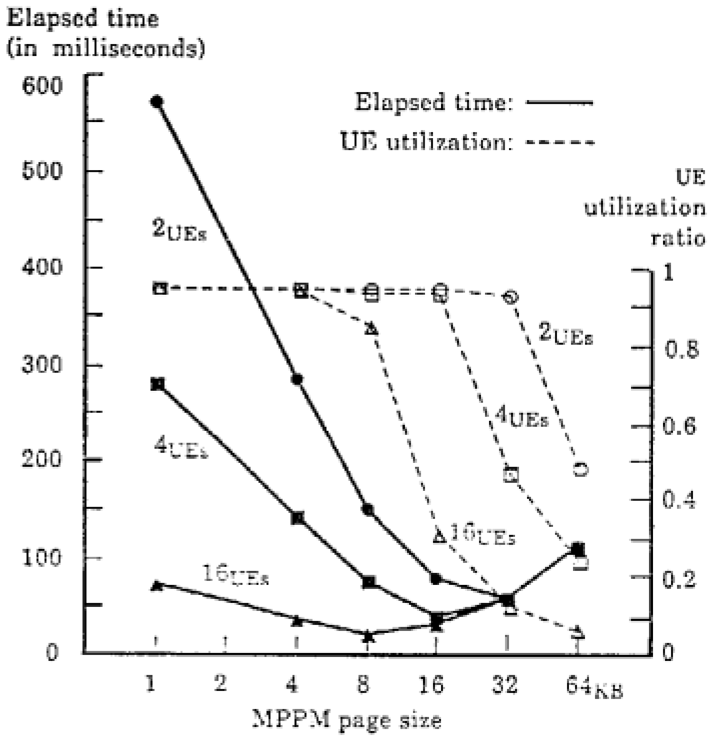


Figure 4-1 Elapsed time and UE utilization ratio by SP method

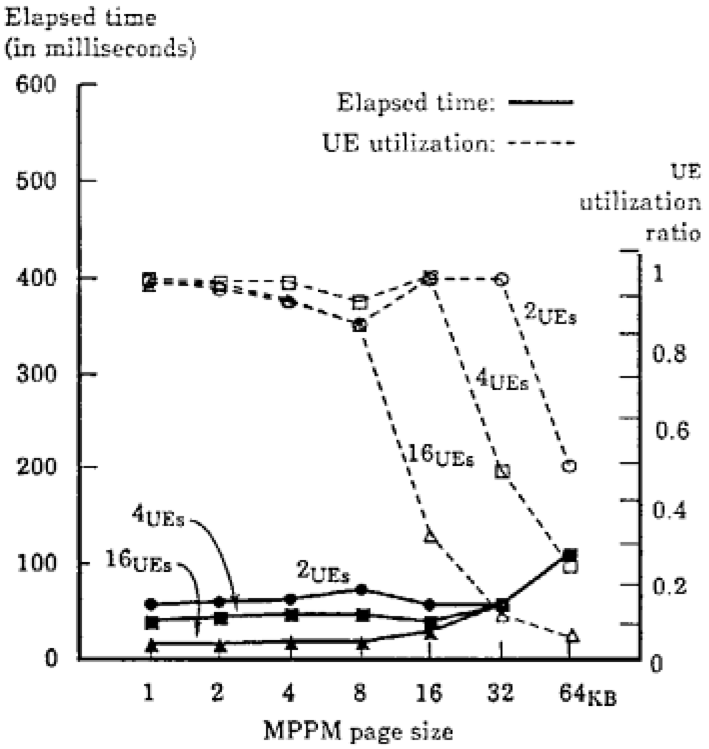


Figure 4-2 Elapsed time and UE utilization by MP method

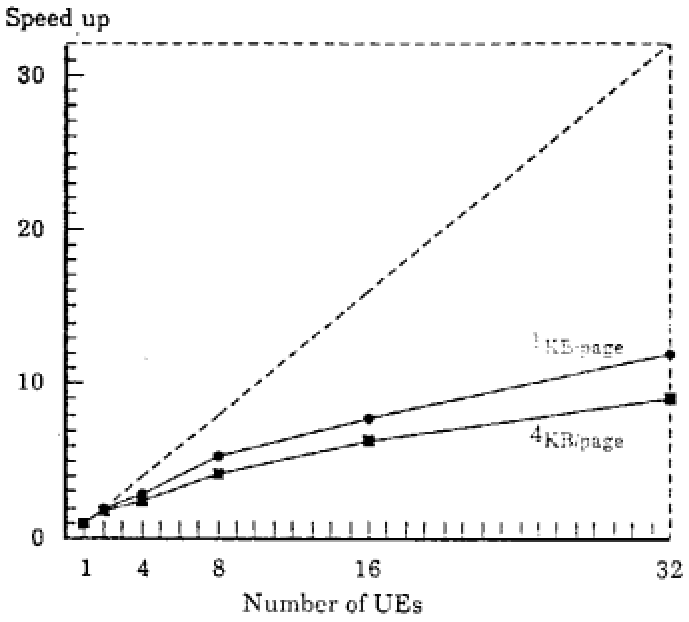


Figure 5-1 Effect of parallel processing.