Acceleration of Join Operations by a relational database processor, RINDA

Tetsuji Satoh, Hideaki Takeda, Ushio Inoue, Hideki Fukuoka
NTT Communications and Information Processing Laboratories,
Yokosuka, Kanagawa 238-03, Japan
e-mail : satoh%nttbss.ntt.jp@relay.cs.net

Abstract : Fast join methods implemented in a relational database processor, RINDA, are described. RINDA performs complex queries including sorts and joins with specialized hardware. Join operations by RINDA are executed in three phases: filtering phase, sorting phase and merge-join phase. In the filtering phase, unjoinable tuples are removed with hashed-bit-arrays. Remaining tuples are sorted in the sorting phase. Sorted tuples are merged and connected together in the merge-join phase. Iterating operations in the filtering and sorting phases are rapidly executed by RINDA's specialized hardware. Especially in the filtering phase, a new multiplication-folding method is used as a hashing function to set and refer hashed-bit-arrays. It strongly reduces collisions for any type and length of keys. Three kinds of join algorithms, nested-loop, single-table filtering and dual-table filtering algorithms, are dynamically selected according to the number of tuples to be joined. Performance evaluation shows RINDA accelerates join operations about ten times compared with conventional software systems.

1. Introduction

In recent relational database systems, high speed query execution are required for retrieving over 1-Giga byte databases. However, non-indexed queries which access all data objects stored in disks take a lot of time when they are executed by general purpose computers because these computers aim to achieve high performance for complex operations with a few data objects. Database machines have been studied and developed to realize high-speed execution of database operations based on many iterative data comparisons and replacements.

Database machines can be classified into two types. The first is designed to solve the I/O bottleneck between CPUs and DISKs and the second type solves the CPU neck caused by sorts and joins. CAFS[1] is in the first category and uses special purpose processors attached to disk controllers. Each processor executes tuple selections and restrictions within tuple read out time from disks. GREO[2] with a special hardware sorter and the IDP[3] expanded vector processor which executes database operations are of the second type. The Server/8000[4] series database processor is a hybrid for solving both the I/O and CPU necks. It consists of many micro-processors for I/O operations and a specialized RISC processor designed for database operations.

RINDA[5] is a new relational database processor composed of CSPs and ROPs. The CSP, Content Search Processor, executes tuple selections and restrictions within tuple transfer time from disks. The ROP, Relational Operation accelerating Processor, mainly executes sorting and accelerates join operations by using specialized hardware. Thus RINDA executes non-indexed queries 100 times faster than conventional software database management systems since both I/O and CPU bottle necks are solved.

This paper presents fast join algorithms based on the three-phase join method[6] applied in RINDA. The three-phase join method consists of filtering, sorting and merge-join phases. In the filtering phase, unjoinable tuples are removed from joining tables. Remaining tuples, that is the candidate tuples of the join, are sorted in the sorting phase. After sorting tuples of joining tables, those tuples are merged and connected together in the merge-join phase. Operations in the filtering and sorting phase are rapidly done by specialized hardware in the ROP. The merge-join phase is done by software in a host computer because the condition of tuple connection depends on the user's requirement. The work load of the merge-join phase can be almost entirely eliminated by filtering unjoinable tuples and sorting the two tables.

Hashing functions for filtering unjoinable tuples are very important. To filter all unjoinable tuples, the hashing function used to set and refer hashed-bit-arrays must not collide in any key data types and length. If a collision occurs, unjoinable tuples remain by mistake. Therefore, a new multiplication-folding method based on the rotation-folding[7] and multiplication[8] methods was implemented.

In section 2, we discuss the problems of conventional relational database management systems and solutions offered by the RINDA system. A fast join algorithm based on the three-phase-join method is discussed in section 3. In section 4, the multiplication-folding method for filtering functions is proposed and evaluated using a large number of actual keys. In section 5, RINDA architecture for accelerating join operations is described. The performance enhancement of join operations by RINDA is also

DATABASE SYSTEMS FOR ADVANCED APPLICATIONS '91
Ed. A. Makinouchi

evaluated.

2. Problems of Conventional Systems and Solutions offered by RINDA

The performance of relational database systems has been improved by the effective use of indexes. However, some types of queries, i.e. selection by non-indexed columns, sorting after selections and aggregation for statistical analyses, cannot take advantage of them. These queries are basically evaluated by iterative comparisons for a large number of tuples read out of disks. Therefore, they consume a great amount of CPU and IO time.[9] The amount of CPU and IO time increases according to the number of tuples in a table. Unfortunately, queries including join and sort operations need much more CPU time. For instance, sorting operations for n records need at least n*log(n) times of comparisons each of which is a very simple operation. However, the conventional general purpose computer is designed to rapidly execute complex numeric operations.

RINDA achieves high-speed executions of database operations by using specialized hardware. It aims to decrease both CPU and IO time. A typical RINDA system organization is shown in Figure 1. RINDA is composed of CSPs and ROPs each of which is connected to a host computer by a channel interface. The CSP directly searches tables stored in disk volumes, selects tuples and columns requested by a query, and then transfers the results to the host computer as a temporary table. The ROP sorts tuples in a temporary table transferred from the CSP via the host, and then returns a sorted temporary table back to the host. Unjoinable tuples are removed within sorting term.

The RINDA system has increased the cost-performance ratio about an order of magnitude compared with conventional software database management systems on a general purpose computer.

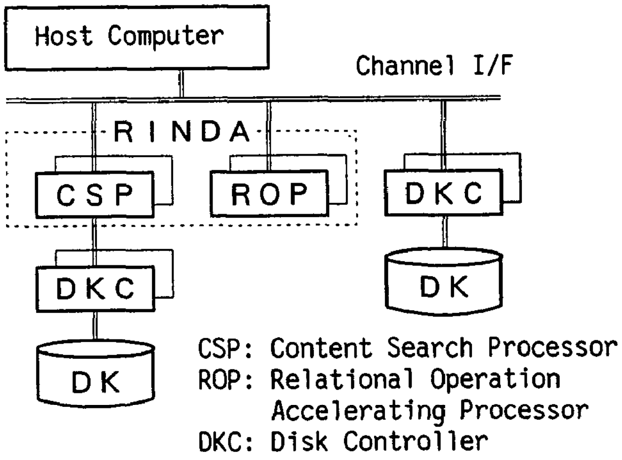

Host Computer
Channel I/F
RINDA
CSP
ROP
DKC
DKC
DK
DK
CSP: Content Search Processor
ROP: Relational Operation Accelerating Processor
DKC: Disk Controller

Fig.1 RINDA system organization

3. Accelerating Methods for Join Operations

3.1. Design Consideration

Conventional join algorithms mainly comprise three groups: nested-loop join algorithms[10] , sort-merge join algorithms[10] and hash-join algorithms.[11] In nested-loop join algorithms, each tuple in the first table is repeatedly compared with all tuples in the second table. Thus, this algorithm can be used only if both tables are small because, although it is easier to control than the others, it requires very large number of comparisons. In hash-join algorithms, both joining tables are split into many packets by a hashing mechanism. Actual join operations are executed between split packets in which every tuple has the same hashing value. Therefore, hash-join algorithm is suitable for parallel execution. In sort-merge algorithms, the amount of merge-join computations decreases to a linear order by separating sort operations for both tables. We selected sort-merge algorithm for join operations in the RINDA system because sort operations are rapidly executed by using specialized hardware.

As mentioned above, the three-phase join method based on the sort-merge algorithm is implemented in RINDA. It consists of filtering, sorting and merge-join phases. Unjoinable tuples are removed in the filtering phase. Remaining tuples each of which is a candidate tuple of join in both tables are sorted in the sorting phase. After that, sorted tuples are merged and matched tuples are connected together in merge-join phase.

The concepts behind how the filtering and sorting phases are realized by hardware are described below.

In the first filtering phase, unjoinable tuples are removed by a filtering method with hashed-bit-arrays.[12] Remaining tuples after the filtering phase are sorted and then merged. Thus, if all unjoinable tuples are removed in the filtering phase, useless sorting and merging operations for unjoinable tuples can be eliminated. However, a few unjoinable tuples inevitably remain because of collisions of hashed results. Therefore, a refined hashing function is required because the number, data type and distribution of the key are unknown before the hashing operation. Moreover, operations in this filtering phase are done on the largest amount of tuples in the three phases. Therefore, we decided the filtering operations should be executed by specialized hardware.

Sorting operations are well known to consume much computing power. We also decided sorting operations should be executed by specialized hardware. The number of keys and its data type are generally changed dynamically. Thus, we implemented a multiway merge sorter[13] that easily handles any number and types of keys. The multiway merge sorter realizes a compact large-capacity hardware sorter.

In the merge-join phase, the number of input tuples are decreased by filtering operations and they are already sorted. Therefore, merge-join operations are executed with little computation. Tuple connection based on merging is assigned many variations by users. That is the reason why merge-join operations are execute by software on the host computer.

3.2. Three-Phase Join Methods

Three-phase join can be attained through single table filtering and dual-table filtering methods. These methods, shown in Figure 2, are described below.

(a) Single-table Filtering Method:

Temporary tables for join operations, R' and S', are made by the CSP from the base tables, R and S, respectively. Tuples in the first table R' are input into the ROP to set a hashed-bit-array and to be sorted. Sorted tuples are output from the ROP as a sorted table R". After that, tuples in the second table S' are input into the ROP to refer the hashed-bit-array and to be sorted. Unjoinable tuples in S' are removed by this set and refer operations to the hashed-bit-array. Sorted tuples are also output as a sorted table S". Finally, each tuple of both sorted tables, R" and S", is merged and connected in host computer. Thus, unjoinable tuples in the second table S' are removed in the single-table filtering method.

(b) Dual-table Filtering Method:

Tuples in the first table R' are input to the ROP only for setting a hashed-bit-array. Tuples in the second table S' are then input for filtering unjoinable tuples. Remaining tuples in the S' set another hashed-bit-array and are sorted as a sorted table S". Tuples in the R' are then input again for filtering using the second hashed-bit-array set by remaining tuples in the S'. Remaining tuples in the R' are sorted as a sorted table R". Finally, each tuple of both sorted tables, R" and S", is merged and connected in the host computer. Thus, unjoinable tuples in both tables are removed in the dual-table filtering method.

Join operations based on either method can be done when the number of remaining tuples after filtering is below the capacity of the sorter. Therefore, tables whose tuples are over the capacity of the sorter can be handled if the number of tuples in the filtered table is below the capacity. Moreover, both the time of tuple transferring and join operations in the host is decreased by filtering unjoinable tuples.

4. Hashing Function

Collisions of hashed results generally occur when unknown keys are hashed. Especially, in the join operations, keys may be composed of several columns, each of which has a different data type and non-uniform distribution by the former selection operations. In this case, the probability of collisions will be increased. Unjoinable tuples cannot removed when collisions of hashed results occur. Therefore, a sophisticated hashing function is necessary for decreasing the number of hashing collisions.

An ideal hashing function can distribute all keys randomly in the addressing space of a hashed-bit-array. In papers (7) and (8), conventional hashing functions were compared from the view point of collisions using fixed-length short keys. The hashing functions compared included division, multiplication, folding and other methods. The division method exhibited a good result with fewer collisions for the unknown set of keys. However, in the filtering phase of join operations, the division method cannot be used directly because keys may be long and of variable-length. The requirements of hashing functions for filtering operations are as follows:

a) The hashing table can be composed of a bit array. Loading factor, i.e. the number of hashing keys over the size of a bit array, is slightly small.

b) It is important to decrease the collisions, but the operation for overflow is not necessary.

c) Any keys having various data types can be hashed by the same hashing function.

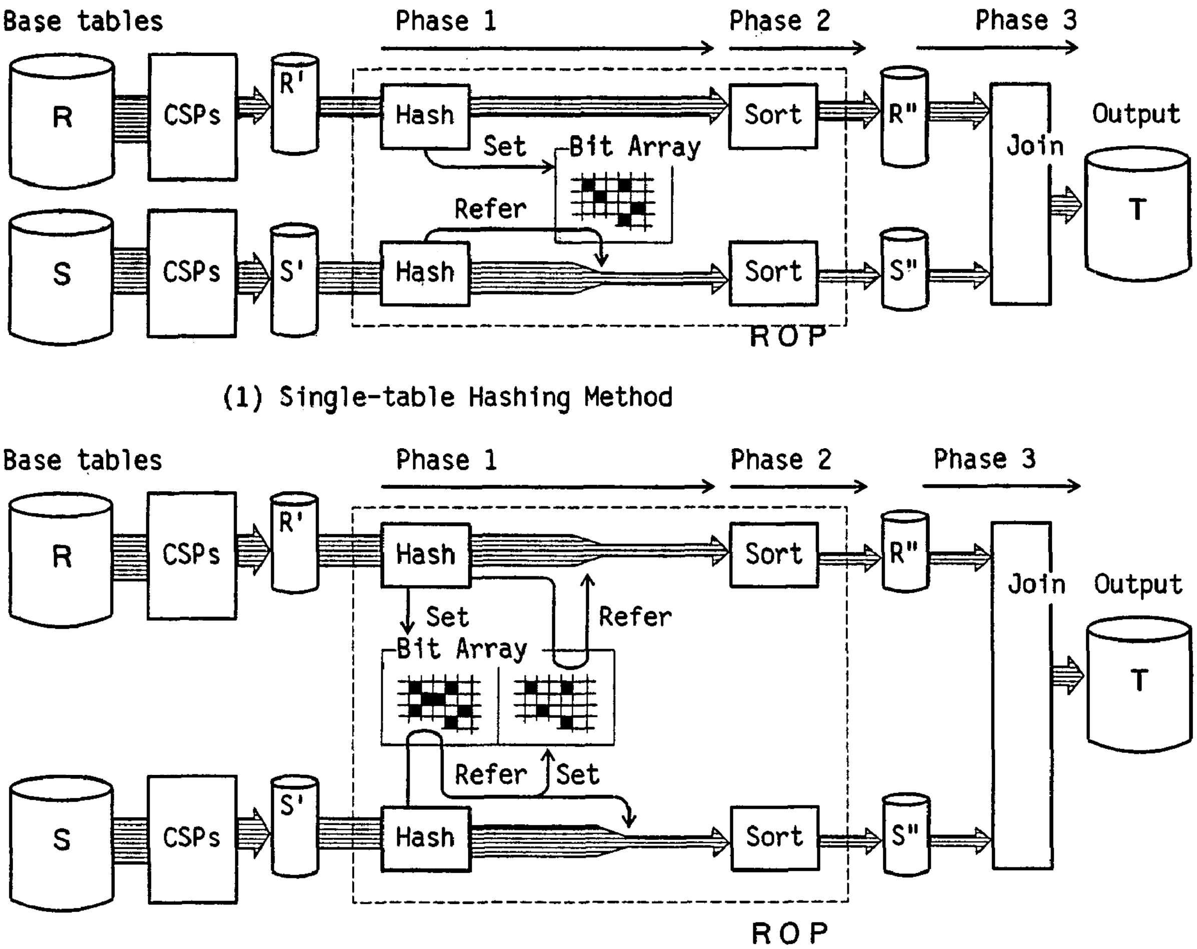
d) Any keys with long and variable-length can be hashed.

For hashing long and variable-length keys, the exclusive-or method is available. Keys are divided into some fixed-length fragments. These fragments are folded by exclusive-or logic. However, in character strings, especially in Japanese kanji strings, the probability of '1' occurrence in each bit is not even. Thus hashed results have biases by the simple folding method with one or two-byte fragments. To solve this problem, fragments should be folded after shift or bit-order reversed operations.

A new multiplication-folding method for key hashing was developed for RINDA. This method, which we call the RINDA method, is composed of a folding method with bit shift operation and a multiplication method for fixed-length fragments. The multiplication method is applied to randomize the character code, and the folding method is applied to handle variable-length keys. The RINDA method is achieved by compact specialized hardware such as exclusive-or and simple multiplication circuits.

The RINDA method was evaluated by the large number of actual keys. The hashing effect of the RINDA method is compared with an ordinal bit-shift folding method in Figure 3. The horizontal axis is the number of collision keys, which means the number of different keys having the same hashing result. The vertical axis is the number of keys in which collisions occur. The ordinal folding method is only shown in the six-bit shift case which is the best one within zero to seven-bit shift cases. Figure 3(a) shows the result for random numeric keys with 16-byte fixed length. Figure 3(b) is for variable length keys, each of which is a headword of an English dictionary. Both are evaluated using 235k keys.

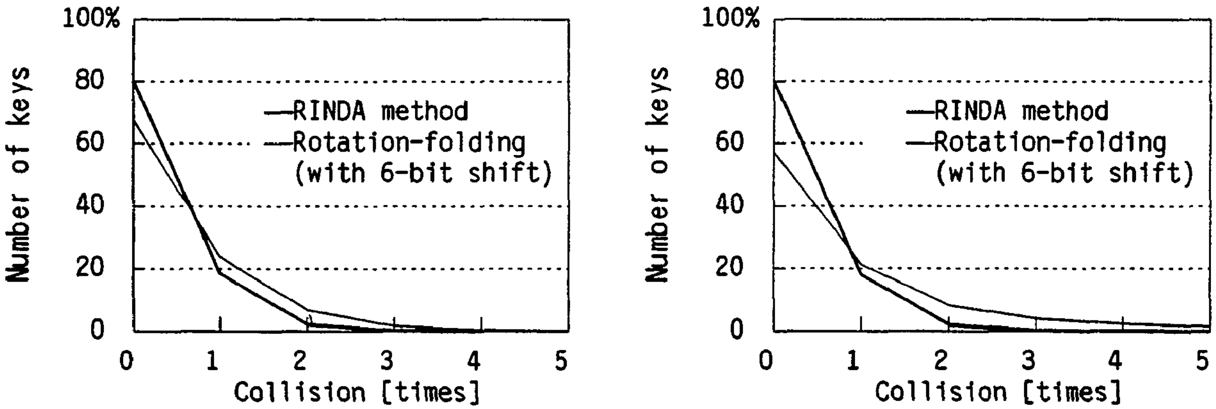
The results of evaluations indicate the RINDA method achieves better effects than the ordinal method in all cases. Especially, for headword keys which have key distribution bias, over 80% of the keys were hashed without collisions by the RINDA method. On the other hand, hashing with the ordinal method was below 60%. Moreover, compared with the number of collisions, the RINDA method gets similar hashing results in both cases,



(1) Single-table Hashing Method

(2) Dual-table Hashing Method

Fig.2 Sort-merge Join Methods using RINDA



(a) Numeric keys <16-Byte fix>

(b) Headword keys <Valiable length>

Fig.3 Hashing Effect on RINDA method (235k keys/1M-bit cells)

while the ordinal method is influenced by the distribution of keys. In the worst collision case, five or more headwords, which is over 10% of all keys, have the same hashed result by using the ordinal method. In the RINDA method, however, collisions occurring between three or more keys were rare.

Now, we discuss the relationship between loading factor and collisions. The probability of collision occurrence was evaluated using several sizes of bit-arrays with the same number of keys. In Figure 4, the loading factor, that is key numbers over the bit-array size, is covered from 1 to 1/8. When the loading factor is 1, that means the number of keys equals the number of cells of the bit-array, about 40% of the keys were hashed without collisions. The remaining 60% of the keys caused collisions. When the loading factor reaches to 1/4, the probability of collisions decreases to 20% and the number of collision keys, if collision occurs, is only two in most cases. The probability of collisions is decreased according to loading factor decrease. Therefore, in the filtering operation with hashed-bit-arrays, key collisions can be reduced by increasing the size of the bit-array. This means increasing the size of the bit-array is necessary when the number of keys increases.

5. Implementation and Performance Evaluation

5.1. Join Implementation in RINDA

Dynamic optimization of join methods are implemented in the RINDA system. The optimal join method, in single-table filtering, dual-table filtering or nested-loop join methods, is selected by the number of tuples counted by the CSP. The nested-loop join method is performed when both temporary tables read out by CSP are small and can be joined without extra-IO operations. The single-table filtering method is performed when the temporary table is small enough compared with the other. In other cases, the dual-table filering method is performed.

A block diagram of the ROP, which performs filtering and sorting operations, is shown in figure 5. ROP characteristics for join execution are as follows:

(a) Internal Key

Tuples are generally composed of several columns each of which has a different data type such as an integer or a decimal number with or without sign or character string. Null value, which means the value of that column is undefined, may appear.

The ROP has key-extract hardware which composes internal keys from tuples. This hardware translates from the tuple having several kinds of data types to a comparable key and makes it fixed-length from variable-length or the null value. Therefore, filtering and sorting operations for fixed-length keys are rapidly executed in simple hardware.

(b) Working Memory Storage:

The internal key is used in the ROP for realizing small sized and easy controlled hardware. Original tuples must be stored to make an output temporary table. Thus, each key has a pointer assigned to the tuple stored position. After filtering and sorting operations, appropriate tuples are read out by the pointer attached to the key, and then transferred to the host as a temporary table. Incidentally, hashed-bit-arrays need some amount of capacity as discussed in section 4. Therefore, three types of memory storage for storing keys, tuples and bit-arrays must be prepared in the ROP.

Single working memory storage is implemented in the ROP for storing keys, tuples and bit-arrays. This storage unit is compact because it is composed of large capacity RAM chips. The boundaries between key area, tuple area and bit-array area are dynamically moved for storing any length keys and tuples. The size of the bit-array is assigned by the constant ratio of the working storage capacity to keep loading factor low. If the capacity of the storage is increased, so is the size of bit-arrays. Keys and tuples are stored in the remaining area of the working storage. The boundary between keys and tuples moves dynamically because the length of each tuple is variable.

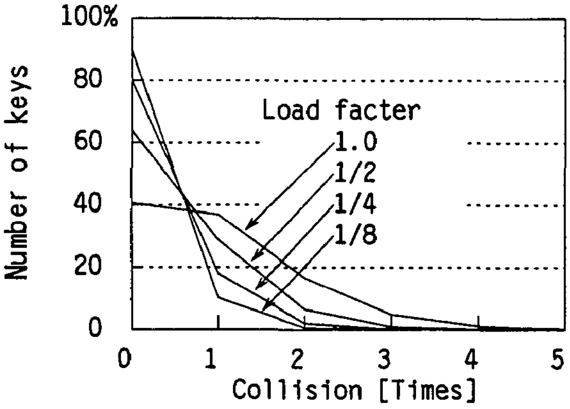


Fig.4 Hashing Effect for Load Facter

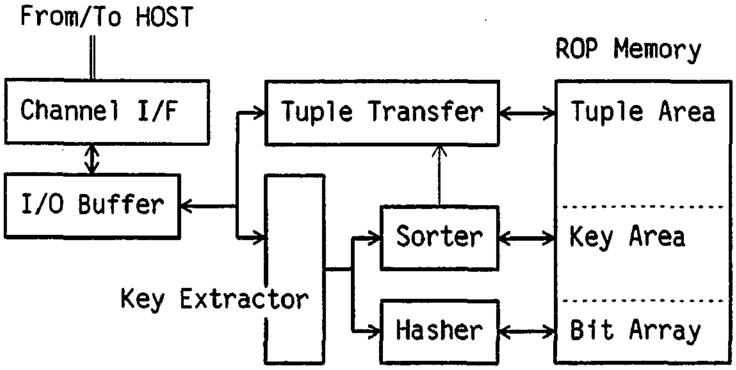


Fig.5 ROP Block Diagram

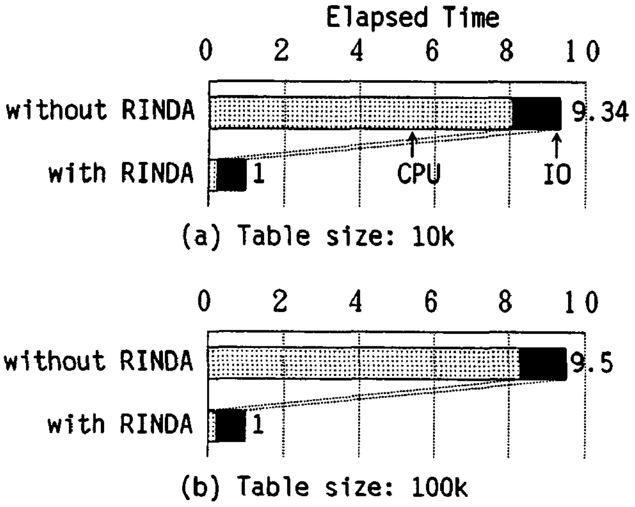


Fig.6 Performance Improvement of Join operation (Normalized by time with RINDA)

5.2. Performance Evaluations

The performance of a two table join operation by the RINDA system was evaluated using the Wisconsin Benchmark.[14] Elapsed time of the join for 10,000 and 100,000 tuples are shown in Figure 6. The time is measured in the host computer from the start of execution of the query to the complete storing of the join results into a temporal table. The RINDA system was composed of a small-sized DIPS-V30E host computer, two CSPs and an ROP.

RINDA accelerates join operations about 10 times independent of the size of tables. The CPU time of the host was dramatically decreased by using RINDA. The reasons of this acceleration with RINDA are as follows.

Join operations of these queries are performed by the single-table filtering method after 10% selection by CSPs. In the case of 100,000 tuples, the selected 10,000 tuples are input in the ROP. They set the hashed-bit-array and are sorted. Then, the other 100,000 tuples are input to the ROP and filtered by referring to the hashed-bit-array. In the Wisconsin Benchmark, the number of tuples after filtering is 10,000 by a collision-free filtering operation because the join column is a unique attribute. Thus, the time of tuple transferring from the ROP to the host and storing as a sorted temporary table is decreased substantially. Moreover, tuple comparing time in the host is dramatically decreased because most unjoinable tuples have already been removed by the ROP.

6. Conclusions

Join accelerating methods have been described. These methods are implemented in the relational database processor, RINDA.

(1) The three-phase join method was implemented for accelerating join operations. It consists of the filtering phase which removes unjoinable tuples, the sorting phase for remaining tuples and the merge-join phase for connecting appropriate tuples. In RINDA, the filtering and sorting phases are directly performed by specialized hardware.

(2) A multiplication-folding method for the hashing function of filtering operations is used for hashing any type and length of keys.

(3) According to the performance measurements, RINDA accelerates join operations 10 times compared with a conventional software system.

Acknowledgments

The authors would like to thank Masato Haihara and Kenji Suzuki for their useful suggestions and discussion. The authors are also grateful to Toshio Nakamura and Jyunichi Kuroiwa for their helpful assistance with the measurements.

References

1. Babb E.:, "Implementing a Relational Database by Means of Specialized Hardware," *ACM Trans. Database Systems*, vol. 4, no. 1, pp. 1-29, 1979.
2. Kitsuregawa M. and Yang W. et al., "Implementation of LSI Sort Chip for Bimodal Sort Memory," *Int'l Conf. on VLSI'89*, pp. 285-294, Aug. 1989.
3. Kojima K., Torii S., and Yoshizumi S., "IDP- A Main Storage Based Vector Database Processor," in *Database Machines and Knowledge Base Machines* , ed. Kitsuregawa M. and Tanaka H., pp. 47-60, Kluwer Academic, 1988.
4. Britton Lee Inc., *Server/8000 for use with ShareBase II Software*, Dec. 1988.
5. Inoue U., Hayami H., Fukuoka H., and Suzuki K., "RINDA - A Relational Data-base Processor for Non-indexed Queries," *Int'l. Symp. on Database Systems for Advanced Applications*, pp. 382-386, Apr. 1989.
6. Inoue U. and Kawazu S., "A Relational Database Machine for Very Large Information Retrieval Systems," in *Database Machines, NATO ASI Series*, vol. F24, pp. 183-201, Springer-Verlag, 1986.
7. V.Y. Lum, P.S.T. Yuen, and M. Dodd, "Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files," *Comm. of the ACM*, vol. 14, no. 4, pp. 228-239, 1971.
8. G.D. Knott, "Hashing functions," *Computer J.*, vol. 18, no. 3, pp. 265-287, 1975.
9. Boral H. and Redfild S., "Database Machine Morphology," *Proc. of 11th Int'l. Conf. on Very Large Databases*, pp. 59-71, 1985.
10. Blasgen M.W. and Eswaran K.P., "Storage and Access in Relational Data Bases," *IBM System Journal*, no. 4, pp. 363-377, 1977.
11. Kitsuregawa M., Tanaka M., and Moto-oka T., "Application of Hash to Data Base Machine and Its Architecture," *New Generation Computing*, vol. 1, no. 1, pp. 63-74, 1983.
12. McGregor D.R., Thomson R.G., and Dawson W. N., "High Performance Hardware for Database Systems," in *Systems for Large Data Bases*, ed. P. C. Lockemann and E. J. Neuhold, pp. 103-116, North-Holland Publishing Company, 1976.
13. Satoh T., Takeda H., and Tsuda N., "A Compact Multiway Merge Sorter using VLSI Linear-array Comparators," *Int'l. Conf. on Foundation of Data Organization and Algorithms*, pp. 223-227, June 1989.
14. Bitton D., DeWitt D. J., and Turbyfill C., *Benchmarking Database Systems - A Systematic Approach,* CTSR #526, Univ. of Wisconsin-Madison, 1983.