Translation from Transactions of IPSJ

A Multiway Merge Sorter for Sorting of Large Databases

TETSUJI SATO*, HIDEAKI TAKEDA* and NOBUO TSUDA*

A multiway merge algorithm by using specialized hardware is proposed for sorting a large number of records. By adopting a sorting array that compares k records in parallel and a data-driven control technique that selects k strings for merging, the algorithm performs merge processing for a large number of k-way whose speed is independent of the merge ways. A large number of records can be sorted by iteration of the k-way merge operations. The number of iterations is substantially reduced by increasing the number of merge ways. A compact hardware sorter can be achieved to satisfy both speed and capacity respectively since the sorting array and the working storage can be implemented independently. The database processor *RINDA* applies the multiway merge sorter for accelerating sorts and joins. The configuration and performance of the sorter are also discussed.

1. Introduction

Sorting is a fundamental operation in non-numeric computational areas. A wide variety of software algorithms have been proposed [1] to perform sorting in addition various application domains have been clearly mapped out. In current relational database operations, the number of sorting records varies tremendously according to the application and there is an increasing demand for the capability to sort more than a million records. Moreover, sortings must also efficiently accommodate records of varying length from very short integers to long character strings. Meanwhile, application-specific hardware can easily be fabricated with the dramatic advances in VLSI technologies in recent years.

This paper proposes a multiway merge algorithm that can be applied to implement a compact, high-speed sorter. The sorter is specifically conceived for application to relational database processing where a high degree of flexibility is required in terms of number and length of records. Many kinds of sorters have been proposed to increase sorting speed by using hardware parallelism and pipeline processing. However, such schemes involve increasing the hardware volume as the number of records increase such that $O(\log_2 N)$ [2-4] or $O(N)$ [5, 6], where N is the number of records to be sorted. Even in the case of a pipeline merge sorter [4], which is typical of the $O(\log_2 N)$ approach and generally involves less hardware, as many as 20 merging circuits are required to sort a million records. The circuits are complicated because each one has a different amount of working storage and extra hardware is required to expand the limitation of record numbers and to handle various length records [7].

The newly devised merge sorter consists of a working storage where the records are stored as the strings and a sorting array which is a one-dimensional array for parallel comparison. Applying a data-driven string-selection technique for selecting the strings to be merged in the working storage, perfect-successive multiway merging is achieved independent of the number of merging ways. Through the cascade iteration of multiway merge processing, a large number of records can be sorted. The number of iterations can be decreased with increases in the number of merge ways.

The maximum number of sorting records is determined by the capacity of the working storage, and is not affected by the configuration of the comparator and the controller. Because the number of sorting records depends on the capacity of the working storage and the length of the records, a larger number of shorter records can be sorted. The sorting array which takes up most of the area of the sorter board is particularly well suited to implementation in VLSIs because many sorting elements can be integrated together without increasing the pin-count of the VLSI chip. The working storage is also suitable for compact implementation because it can be integrated in a single area with large capacity dynamic-RAM chips.

The multiway merge algorithm and its hardware configuration are described in Section 2. Section 3 covers technologies involved in devising the compact sorter and describes the multistage merge algorithm, the string

This is a translation of the paper that appeared originally in Japanese in Transactions of IPSJ, Vol. 31, No. 11 (1990), pp. 1653-1660.

*NTT Communications and Information Processing Laboratories, Base Systems Architecture Laboratory.

storing method in the working storage, and the structure of the sorting array that performs the parallel comparisons. The feasibility and performance evaluation of a prototype sorter are discussed in Section 4. An example application to the relational database machine *RINDA* is also described.

2. Multiway Merge Algorithm

Sort operations in a relational database are performed on the results of selections and restrictions with respect to a database that is stored in a disk. Thus, it's impossible to know in advance how many records are to be sorted. It may also be necessary to sort a huge number of records, depending on the application. It is thus difficult to optimize the number of merge circuit stages in the case of conventional sorters [2] where the maximum number of sorting records is dependent on the number of merge circuit stages.

By separating the record comparator from the record storage circuit, however, the authors have developed a multiway merge algorithm that satisfies two independent requirements at the same time [8] — namely, the maximum number of sorting records and sort speed. The rest of this section provides a detailed description of the hardware configuration and algorithm that perform the multiway merge operations.

2.1 Hardware Configuration

The basic structure of the sorter is shown in Fig. 1. As can be seen in the figure, it consists of a linear sorting array that performs the parallel comparisons on the records, a working storage to hold the records, and a merge controller. The sorting array is configured as a one-dimensional array structure consisting of sorting elements in which two records are compared to determine which is larger (smaller). A major departure from conventional pipeline merge sorters [4, 7] is that the centralized working storage is implemented in a small area circuit board using large-capacity dynamic-RAM chips. The k-way merge operations are controlled by the merge controller. Many k strings are merged at one time using the sorting array. Merged strings output from the sorting array are re-stored in the working storage as a new string. Many records can be sorted in a few stages of k-way merging because the merge ways k is increased.

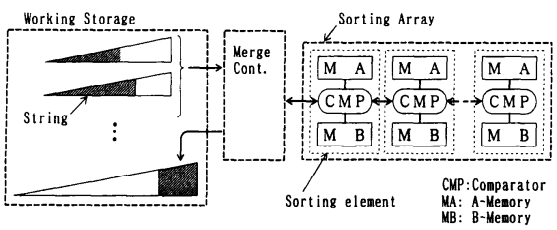

Fig. 1 Block Diagram of Multiway Merge Sorter.

The parallel comparison of the sorting array is illustrated in Fig. 2. Records represented by integers are sorted in descending order. A comparison operation and a transfer operation based on the comparison result are synchronously performed in each sorting element. During an input operation, records are input from the left edge of the sorting array. The smaller record (or more accurately, the less large of the two) in either MA or MB is selected and transferred to the neighboring rightward element. Conversely, during an output operation, the larger record is selected and transferred leftward. The largest record in the array is thus always kept in the left-most sorting element at each input/output step. In other words, regardless of the sequential order of the records, the records are read out without any delay time. For a sorting array consisting of $\lceil k/2 \rceil$ comparator elements, k records are sorted and k strings merged within the time it takes to input and output the records, where $\lceil k/2 \rceil$ is a minimum integer greater than or equal to $k/2$.

2.2 Multiway Merge Algorithm

A large volume of records can be sorted with few merge stages by implementing a large number of merge ways. Here we describe a data-driven string-selection technique using bank-tags that permits a high throughput to be maintained without complicating the control mechanism, even expanding the number of merge ways.

A k-way merge control using bank-tags is depicted in Fig. 3. The figure shows an example of four-way merging in descending order. Four strings indicated by the bank numbers #0 to #3 have been stored in the working storage.

M1-phase: The sorting array is filled with the largest records in all the strings to be merged; that is, the records located at the tops of the strings. Bank-tags are attached that indicate to which strings the records belong.

M2-phase: Merge operations—record output and subsequent input to the sorting array—are performed suc-

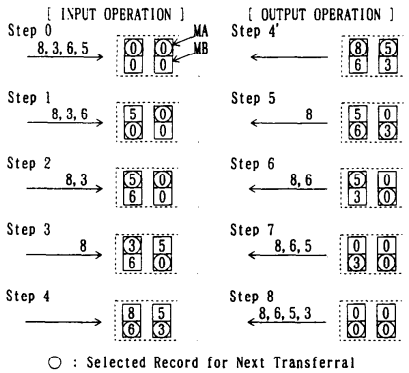

Fig. 2 Schematic diagram of input-output operation.

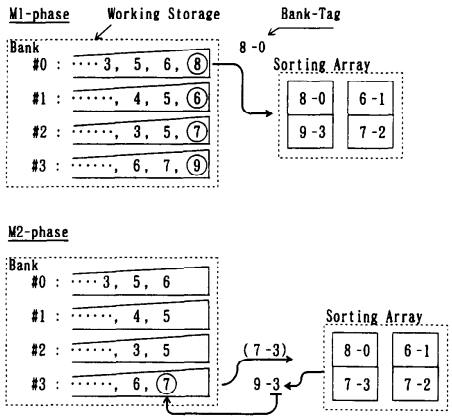

Fig. 3 Continuous Multiway Merging Diagram using Bank-tag.

cessively until all merging strings are empty. The largest record in the sorting array is immediately output with the bank-tag. This is the largest included in all merging strings. The second-largest candidate records are limited to any of the remaining ones in the sorting array or the top one in the string indicated by the largest record's bank-tag. This top one is input to the array and is compared with the remaining $k-1$ records. Successive multiway merging is achieved by this alternate record output-input operation.

The bank-tag controlling the k-way merge processing is attached to the least significant part of the record so as not to affect the record comparison. The minimum number of required bits for the bank-tags is $\lceil \log_2 k \rceil$; thus, a large number of k-way merges can be easily achieved employing a sorting array with $\lceil k/2 \rceil$ sorting elements with very little control overhead.

The total number of merge stages for sorting N records is $\lceil \log_k N \rceil$. The relation between the merge way number and the record number is shown in Fig. 4. It is apparent that more records can be sorted with fewer stages as the number of merge ways increases. Also, the sort processing time for N records—*viz.*, the interval from when the first record is input until the last sorting result is output—is realized in $O(N)$ time because the processing in each stage is done as a parallel comparison by the sorting array. The time is thus proportional to the number of stages $O(N*\log_k N)$.

2.3 Advanced Features of the Sort Algorithm

The basic algorithm for sorting a large amount of records is achieved by the repetition of k-way merge processing. Here we consider some of the unique attributes of the algorithm.

(1) A multiway (several tens of ways) merge circuit is easily realized by combining the parallel comparison on the sorting array and data-driven string-selection technique using bank-tags. Merge processing can be executed in $O(N)$ for a large number of merging ways,

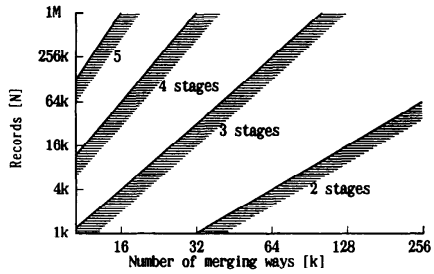

Fig. 4 Relationship of merging ways and record numbers.

and thus a large number of records can be sorted very quickly with few stages. In actual applications employing the multiway merge algorithm, processing times of $O(3N)$ and $O(4N)$ have been achieved.

(2) The maximum number of sorting records depends on the capacity of the working storage, and is not dependent on how the sorting array or controller are configured. Moreover, by increasing the scale of the sorting array and expanding the number of merge ways, the duration of sorts can be shortened with fewer merge stages. This means the sorter can be flexibly configured to optimize either sorting speed or the maximum number of sorting records.

(3) Since the sorting array where the records are compared and the working storage are implemented separately, great flexibility is available in terms of scale and extent of hardware implementation. For example, by adding a dedicated hardware sorting array to a general purpose computer, merge operations with conventional software can be greatly accelerated. *Or*, by constructing an all-hardware sorter with a merge controller, a very fast and compact sorter is realized.

3. Large-Capacity Sorter Configuration

A sorter applied to database processing handles retrieval results in a temporary table, and must therefore be capable of flexibly handling different numbers and different lengths of records. In this section, we describe a multistage merge sorter that meets these requirements. In this sorter, k strings in the working storage are merged and then re-stored there. Through repetition of this process in each k-way merge stage, a massive number of records can be sorted. What follows is a more detailed consideration of an efficient method of storing the strings in the working storage and a sorting array that flexibly deals with records of different lengths.

3.1 Multistage Merge Sorting

(1) Multistage Merge Control

Three- or four-stage multiway merge processing is fully capable of sorting a sufficiently large number of records to be practical. In contrast to conventional

multistage hardware schemes such as whose used in pipeline merge sorters, here a high-speed sorter is compactly realized by exploiting a successive merge processing approach. The multistage merge sort is executed in three types of stages summarized as follows.

Pre-merge Stage: In this stage, all sorting records are divided into k-record strings and sorted in the sorting array. Sorted records are stored in the working storage as strings of size k where the size of the strings is equal to the number of records they contain. When the record input is finished, there are $\lceil N/k \rceil$ strings in the working storage.

Intermediate Merge Stages: In these stages, the strings stored in the working storage are successively merged in iterative k-way merge processes. At each stage of the intermediate merge processing, the k strings stored in the working storage are merged one at a time and then restored as a single string. The size of the generated strings is increased k times and their number is decreased in $1/k$. The intermediate merge stage processing continues until the number of strings is below k.

Output Merge Stage: In this stage, the strings in the working storage that have been reduced to less than k in the intermediate merge stage processing are merged and output.

The processing of the pre-merge and output merge stages is carried out concurrently while records are being transferred between the host computer and the sorter. Thus, the delay time in the sorter (from the end of record input to the beginning of output) is $N*(\lceil \log_k N \rceil - 2)$, where $N \geq k^2$. Typically this is equal to the time required for the one stage—at worst the two stages—of the intermediate merge processing. Furthermore, since the sort processing is carried out through successive merges, the maximum number of records that can be sorted is disassociated from the hardware configuration.

(2) Storage Management

In the intermediate merge stages of the successive k-way merge processes, k strings are read out from the working storage, merged, then re-stored there. Thus, an efficient memory management method is essential to effectively exploit the memory space that is available. We will now evaluate a number of memory management schemes in terms of the effective memory utilization factor, assuming N records of L length are to be sorted—*i.e.*, $LN/$(storage capacity).

Conventional memory management algorithms include the dual memory, the pointer, and the block division methods [9]. In the dual memory approach the working storage is divided into two areas for alternate use. Control is quite simple but the utilization factor is 1/2. The pointer method manages available space by adding a pointer to each record that indicates the next record to be accessed. The method offers very simple control, but its memory utilization factor is $L/(L+$pointer length), and efficiency declines further for shorter records. In the block division method, fixed-sized blocks are allocated to store some or part of strings and reused by implementing pointer chains for control within the blocks. This represents a finer level of subdivision than just partitioning the memory in two as in the dual-memory approach, and thus provides a better utilization factor. With the block division method, auxiliary k blocks equaling the k-way merge number are requried. This method is effective in areas where the way number is small.

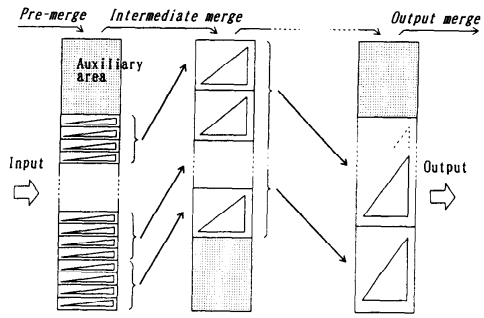

Fig. 5 Merge Process using the Area Store Method.

Considering the drawbacks of the conventional methods, we investigated an area store method for memory management in the multistage merge sorter that yields a memory utilization factor of $k/(k+1)$. With this method, the utilization factor is enhanced as the number of merge ways is increased. The merge process using the area store method is shown in Fig. 5. In the premerge stage, input records are sequentially stored as strings of size k in the working storage starting from the bottom address. A single string is stored in a continuous area which permits uncomplicated memory management. At the end of this stage, the auxiliary area is located above the area containing the strings. The intermediate merge stages are performed so as to maintain a continuous auxiliary area, and merged strings are continuously stored from the top and bottom of the working storage in alternating odd-even stages.

An auxiliary area distinct from the area where the k input strings are kept is prepared to store the output merged strings. Its capacity is equal to the maximum size of the strings stored in the working storage. In other words, it is equal to the size of the largest string output in the last intermediate merge stage.

The condition for minimizing the size of the auxiliary area is that the size of the output string generated in the last intermediate stage consist of uniform-sized k strings. This can be accomplished by generating a string equivalent to k^2 strings in the preceding last intermediate merge stage. In effect this means that a string equivalent to an exponential power of k strings is produced in working storage at the first stage of the intermediate merge processing. At the first stage of in-

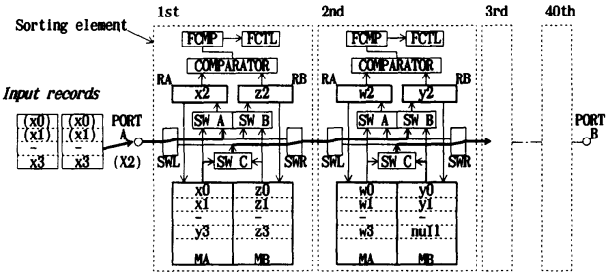

Fig. 6 Architecture of the Prototype Sorting Array.

termediate merge stages, power-of-k strings can be generated with a size differential of no more than k even in the worse case because the number of input records is definitely known at the end of the pre-merge stage. Generally, $N \gg k$ and multistage merge operations can be controlled so the difference in the k size of the strings is kept through the intermediate merge stages. Greater uniformity than this is not required.

3.2 Implementation of the Sorting Array

(1) Basic Configuration

The sorting array consists of one-dimensional cascade-connected comparators, and thus is well adapted to implementation in large-scale integration. A prototype sorting array chip [10] integrating 40 sorting elements is shown in Fig. 6. The figure shows an example of record sorting using 5 records, each of which consists of 4 bytes. The records W, Y, and Z have already been input into the sorting array and have been quasi-sorted in descending order. The sorting sequence consists of transferral and comparison of single bytes. Record X is to be input and compared with record Z in four sequences at the first element. In the sequence stage shown in the figure, $x2$ (the 3rd byte of record X) is input and compared with $z2$ which was read from memory MB in the first element. Simultaneously, $y2$ is input to the second element, (Y was transferred as a result of the previous record comparison in the first element), and compared with $w2$ read from memory MA in the second element. After four comparisons and transfers, the record comparison result is determined. Based on the result, switches (SWA, SWB, SWC) are set to make the data transfer path. The data transfer direction determined by record input and output operations is switched to either SWL or SWR.

(2) Record Length Expansion

Records are kept and compared in each sorting element. Therefore, the record length is limited by the memory capacity of each element. Obviously longer records could be accommodated by increasing the memory capacity of the elements, but this would result in poor memory utilization when short records are processed. Thus, instead of increasing the memory capacity of the elements, longer-length records are handled by cooperative interconnection between the elements. Records that exceed the memory capacity are partitioned and stored in interconnected elements. Partitioned record segments are independently compared in the elements to which they have been allocated. Then the comparison results are integrated by an inter-element control circuit. When elements are interconnected in this way, the effective element number of the sorting array becomes 1/(number of interconnected elements).

This element interconnection method requires circuits to accumulate the comparison results and bypass circuits to transfer data between elements. If these circuits to support interconnection between elements were applied across multiple chips, this would increase the pin-count of the chips and also decrease the comparison speed. Element interconnection has thus been confined within chips and is not applied between chips. By integrating this interconnection circuit together with the sorting elements, a greater number of elements can be realized in the same hardware volume.

(3) Bi-Directional Sorting Method

The sorting operation for k records is performed within the successive k-record input and output time in the pre-merge stage. The input and output of records thus becomes intermittent. To remedy this problem, a bi-directional sorting method has been implemented that reduces the processing time in the pre-merge stage by half.

With bi-directional sorting, alternating input/output operations are performed through PORT-A and PORT-B (located on the left and right edges of the sorting array, as shown in Fig. 6), which can also proceed concurrently. To implement the capability, records entering the sorting array through PORT-A must somehow be distinguished from those entering via PORT-B. The conventional approach was to tag all records entering the sorter regardless of whether through PORT-A or through PORT-B, then determine the entering port at each comparator circuit aligned in a one-dimensional array. Unfortunately, this scheme involves very complicated compare circuit control6, which results in a marked decline in throughput. To

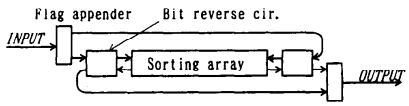

Fig. 7 Block Diagram for Bi-directional Sorting.

Table 1 Record modification rules for bi-directional sorting.

Sorting Order	Flag	Bit-reverse circuits	
		PORT-A	PORT-B
Ascending	'0'	Bit-reverse	Through
Descending	'1'	Through	Bit-reverse

avoid this problem, we have added hardware to the input/output terminals of the present sorting array that implements the bi-directional capability.

A schematic of the hardware supporting bi-directional sorting is shown in Fig. 7. After attaching control flags to the fronts of records, *i.e.*, the most significant bit, they are input to the sorting array via a bit-reverse circuit that reverses the input/output records with control flags. By controlling both the control flag value and bit-reverse circuits, all records input through PORT-A are always larger than those input through PORT-B. Records input through the two ports are thus clearly distinguishable to realize bi-directional sorting. The record modification rules for implementing bi-directional sorting are summarized in Table 1. This method permits sorting in either ascending or descending order while at the same time supporting the bi-directional capability. The sorting array is thus able to perform consistent comparison and transfer operations regardless of the sorting order. The control of the comparator elements is also simplified. Moreover, since the flags attached to the fronts of records and the bank-tags described earlier are the same length, the record length handled by the sorting array can be made uniform through all the process stages—pre-merge, intermediate merge, and output merge.

4. Implementation and Performance Evaluation of the Sorter

Each chip employed in the prototype sorting array shown in Fig. 6 is capable of comparing 80 16-byte records in parallel at a throughput of 3M-bytes/second. A single-board prototype sorter was configured using this VLSI sorting array chip, an 8MB working storage, and a microprocessor. The prototype sorter sorted 500,000 records in three merge stages in 13.4 seconds. Each record consisted of a 15-byte integer and 1-byte control data which act as either bank-tag or control flag. The record length expansion technique described in Section 3.2 and the bi-directional sorting method were thus found to perform up to the designed expectations. By further implementing the area store method, a large-capacity sorter was realized.

A hardware sorter based on the present algorithms was implemented in the relational database processor *RINDA* [11, 12]. The sorting array and merge controller in the sorter is integrated by using dedicated VLSIs. The *RINDA* hardware executes high-speed selections and restrictions for large databases stored in disks and sorting, and also accelerates join operations. The *RINDA* is connected between a host computer and a disk device by means of channel interfaces. The compact sorter implemented in *RINDA* is described below.

(1) Pre-merge and output merge stage processes are performed within the tuple transfer terms to/from the *RINDA*. The transferred tuples are placed in a temporary table including a null column and variable-length tuples. Dedicated hardware then performs column extraction and code conversion to convert the tuples to fixed-length records. As a result, the length of the sorted records generally becomes shorter than the tuple length. The sorter architecture does not depend on the number of columns or attributes. Measured results reveal that the processing time in the intermediate merge stages takes less than 10% of the tuple transfer time, and the sorting could be accomplished in about the same time as the tuple transfer.

(2) The comparator element interconnect capability permits the sorting of records up to 250 bytes in length. Moreover, by employing large capacity dynamic-RAM chips, 64 MB of working storage could be implemented compactly.

5. Conclusions

The basic algorithm and implementation of a compact sorter to apply a large database has been described.

(1) A high-speed multiway merge algorithm has been devised that is capable of several tens of merge ways through the use of a data-driven string-selection technique and a pipelined parallel comparison technique. Through repetition of the merge processing, a large amount of records can be processed.

(2) A sorting array for comparing the records and a working storage for storing the records can be implemented separately in high-density VLSIs. This makes it possible to devise a compact sorter that can be optimized for a range of sorting applications.

(3) By using the element connection method in the sorting array, the length of sorting records can be extended. A hardware sorter implementing this technique has been designed that offers enhanced flexibility to accommodate longer record lengths.

(4) An area store method has been devised that efficiently exploits the working storage to enable large-capacity sorting. For a multiway (several tens of ways) merge sorter, a very satisfactory memory utilization factor is achieved.

Acknowledgement

The authors express gratitude to Tadamichi Kawada, Executive Research Engineer at NTT's Applied Electronics Laboratories, for his guidance in pursuing the present work. They also gratefully acknowledge many useful suggestions from their colleges in the *RINDA* Group.

References

1. KNUTH, D. E. The Art of Computer Programming, **3**, Sorting and Searching, Addison-Wesley, 1973.
2. TODD, S. Algorithm and Hardware for a Merge Sort Using Multiple Processors, *IBM J. Res. & Dev.*, **22**, 5 (Sept. 1978), 509–517.
3. TANAKA, Y., NOZAKA, Y. and MASUYAMA, A. Pipelined Searching and Sorting Modules as Components of a Data Flow Database Computer, *Proc. of IFIP Congress '80* (Oct. 1980), 427–432.
4. KITSUREGAWA, M. and YANG, W. Evaluation of 18-stage Pipeline Hardware Sorter, *Int'l Workshop on Database Machine* (June 1989), 142–155.
5. KUMAR, M. and MIRSCHBERG, D. S. An Efficient Implementation of Batcher's Odd-Even Merge Algorithm And Its Application in Parallel Sorting Schemes, *IEEE Trans. Comput.* (Mar. 1983), 254–264.
6. MIRANKER, G., TANG, L. and WONG, C. K. A "Zero-Time" VLSI Sorter, *IBM J. Res. & Dev.*, **27**, 2 (Mar. 1983), 140–148.
7. KITSUREGAWA, M., YANG, W., FUSHIMI, S., KIMURA, H., SHINANO, J. and KASAHARA, Y. Implementation of LSI Sort Chip for Bimodal Sort Memory, *Int'l Conf. on VLSI '89* (Aug. 1989), 285–294.
8. 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* (June 1989), 223–227.
9. KITSUREGAWA, M., FUSHIMI, S., TANAKA, H. and MOTO-OKA, T. Memory Management Algorithms in Pipeline Merge Sorter, *Int'l Workshop on Database Machine* (1985), 208–232.
10. TSUDA, N., SATOH, T. and KAWADA, T. A Pipeline Sorting Chip, *IEEE ISSCC Digest of Technical Papers* (Feb. 1987), 270.
11. INOUE, U., HAYAMI, H., FUKUOKA, H. and SUZUKI, K. RINDA—A Relational Database Processor for Non-indexed Queries, *Int'l Symp. on Database Systems for Advanced Applications* (Apr. 1989), 382–386.
12. SATOH, T., TAKEDA, H., INOUE, U. and FUKUOKA, H. Acceleration of Join Operations by a Relational Database Processor, RINDA, *Int'l Symp. on Database Sysems for Advanced Applications* (Apr. 1991), 243–248.