SPIRIT-III: an advanced relational database machine introducing a novel data-staging architecture with Tuple Stream Filters to preprocess relational algebra

by NORIYUKI KAMIBAYASHI
Hiroshima University
Hiroshima, Japan

and
KAZUO SEO
Mitsubishi Electric Corporation
Amagasaki, Japan

ABSTRACT

This paper proposes an advanced architecture of the relational database machine (RDBM), named SPIRIT-III, which is basically organized into a three-level memory hierarchy with a sophisticated data-staging and preprocessing architecture for executing relational algebra. SPIRIT-III aims at totally improving both I/O and CPU processing boundary problems and has two major architectural features. One is the introduction of the relational-database-oriented data-staging mechanism, called the look-ahead data-staging mechanism, which can optimally schedule data movement in the memory hierarchy. The other is to attach refined preprocessing mechanisms for relational algebra operations to data transfer lines connected between each memory stage. When a relation stages up or down in the memory hierarchy, these preprocessing mechanisms can function to select tuples and attributes needed by a query and to arrange the relation for parallel processing. SPIRIT-III provides three basic preprocessing filters, called as a whole the Tuple Stream Filter: the tuple selector, the attribute selector, and the grouping filter, implemented with a hash function, which rearranges an original relation and groups the relation into subrelations. The operation of this grouping filter is the primitive preprocessing operation for executing Join and Projection. Then, without the overhead of interprocessor communications, each microprocessor can execute relational algebra operations to a few subsegments assigned to it in parallel. Therefore, SPIRIT-III can perform Join and Projection operations by \( O(N/L) \) \((L = \text{number of microprocessors})\), whereas the early RDBMs required \( O(N \times N/L) \). The proposed SPIRIT-III, which includes features from data-staging architecture to relational algebra execution architecture under the total concept, is the most powerful RDBM based on the state of the art.
1. INTRODUCTION

The relational model\(^1\) proposed by E. F. Codd is based on the set theory and has the advantages of simplicity, data independence, and symmetry of access. These features make it possible to provide nonprocedural query languages and high-level user interface. This tends to enhance the usability and the intelligence of database management systems. When one tries to implement a relational database system on a von Neumann type of general-purpose computer, there are crucial problems, such as poor capacity of data transfer and inefficient relational operation. Thus, the need to develop a relational database machine (RDBM) has emerged. The primary aims in designing RDBMs are to improve the execution time of sophisticated relational algebra operations such as Join and Projection and to reduce the cost of data transfer between the database stores (DBS) and the primary processing subsystem.

However, it has been very difficult to solve both the two major problems in one try. Because traditional RDBMs aimed at improving the efficiency of relational operations, these machines had restricted ability from the viewpoint of total relational database processing and could not provide the harmonized mechanism that would reduce the burdens of both data staging and relational operations.

We discuss the dilemma that system designers face in developing practical RDBMs, taking an example of RAP\(^3,4\) projects. The RAP project, well known as a pioneer of RDBM, based on the concept of logic-per-track, employs architecture on which the associative processing mechanism is added on a fixed-head disk, used as DBS to transfer the final results of a query to the host. It has been noted, however, that RAP\(^3\) has difficulty in supporting efficiently such complicated set manipulations as Projection and Join, and also in handling large databases because of its restriction on the capacity of DBS. In RAP\(^2,4\), therefore, the processing subsystem is separated from DBS and the moving head disk is employed as DBS. This approach is realistic in regard to background, such as technical maturity and possibility, and is advanced from the viewpoints of (1) implementing RDBMs based on the state of the art and (2) the need to support large databases. Bottlenecks of data access and transfer, however, still exist in such RDBMs. Thus it is important to solve the problem of the cost of large data staging at the system architectural level.

In this paper, in order to develop realistic RDBM that copes with this problem, we propose an advanced architecture of the relational database machine, named SPIRIT-III. It is basically organized into a three-level memory hierarchy (MH); primary work memory (PWM), staging buffer (SB), and database store (DBS). SPIRIT-III aims at totally solving both I/O and CPU boundaries, which are key problems in implementing RDBMs. It consists of three cooperative sub-systems and has two major architectural features. One is the introduction of the relational-database-oriented data-staging mechanism which can contribute to the optimal scheduling of data access and transfer between each two levels of the memory hierarchy. The other is to attach refined preprocessing mechanisms of relational algebra operations to data transfer lines connected between each two levels of the memory hierarchy. When a relation stages up or down from DBS or PWM to SB, these preprocessing mechanisms can function to select the tuples and attributes to be staged up and to arrange the relation for parallel processing. SPIRIT-III provides three basic preprocessing functions. Two of these are the tuple selector, filtering only tuples to match retrieval conditions; and the attribute selector, repacking only attributes needed by executing a given query. Because these functions, which select only attributes and tuples required by the query, contribute to reducing the data to be staged up and to saving the space of upper-level memory, SPIRIT-III can enhance both the throughput of data staging in the data staging subsystem (DSS) and the execution efficiency of relational operations in the relational processing subsystem. In addition to these filtering functions, SPIRIT-III incorporates a grouping filter, proposed by the authors, into each stage of memory hierarchy. In streaming a relation on data lines connected between each two memory stages, this grouping filter groups the relation into subrelations. This grouping operation is the primitive of executing heavy relational operations, such as Join and Projection. In practice, this grouping filter can be implemented with a hash function. Microprocessors of RPS can execute postoperations associated with relational algebra operations to a few subsegments assigned to each of these in parallel without the overhead of interprocessor (subsegment) communications. The relational algebra execution architecture of SPIRIT-III is composed of two stages. One stage, prepared in the DSS, contributes to selecting only necessary tuples and attributes and to arranging an original relation for parallel processing. The next stage in RPS parallel environment is responsible for performing the postoperations on the grouped subsegment. The postoperations include duplicate elimination within the subsegment and concatenation operation between tuples within the subsegment of relation R and tuples within the subsegment of relation S. Both subsegments are filtered by the same hash function.

Therefore, SPIRIT-III can obtain the time performance \(O\left(N/L\right)\) in performing Join and Projection operations \((L = \text{number of microprocessors} \quad N = \text{cardinality of a relation})\); whereas the first-generation RDBMs\(^7\) required \(O\left(N \times N\right)\) and the second-generation machine\(^5\) realized \(O\left(N \times N/L\right)\).

We now view several architectural approaches to RDBM and discuss them. The first-generation RDBM attempted to
2. Basic System Architecture of SPIRIT-III

2.1 System Architecture

Figure 1 shows the conceptual system architecture of SPIRIT-III, which is composed of the three major subsystems: the integrated system scheduling subsystem (ISS), the data-staging subsystem (DSS), and the relational database processing subsystem (RPS).

In SPIRIT-III, a three-level memory hierarchy is introduced to enhance staging throughput of data. The memory hierarchy consists of three types of memories, such as the primary work memory (PWM) for database microprocessors, the staging buffer (SB) as a disk cache, and the database store (DBS), composed of moving arm disks. SPIRIT-III incorporates refined preprocessing architecture, for efficient execution of relational operations, and advanced data-staging architecture into the memory hierarchy. These architectures cooperate to improve the throughput of data staging and the efficiency of set level operations.

ISS is responsible for scheduling the process execution in RPS and data staging in DSS according to the proposed policy, called ISR, for Integrated process and data Scheduling in Relational database environments. The ISR policy integrates the data-driven process-scheduling policy in RPS environment and the look-ahead data-staging policy employed by DSS (described in Section 4). ISS also coordinates the actions of the other two subsystems. To be concrete, when a transaction arrives at the system, ISS registers the processes used by the transaction to the process pool and registers the data access request for the processes of the transaction to the data staging pool. DSS can schedule the sequence of data staging out of accordance with the predefined execution sequence of a transaction in normal mode. When all data required in executing a process are staged up to the memory of RPS, ISS changes the state of this process in the process pool to the executable state. Then, according to the policy of data-driven process scheduling, RPS executes the process under a multi-transactions environment. This scheduling strategy is based on the property of the high flexibility of the execution sequence of the relational algebra tree. The introduction of the ISR policy, minimizing the total data-staging cost, leads to the realization of a high-performance RDBM.

DSS optimally schedules data staging between each two memory hierarchies in order to minimize the total cost of data staging, which is defined as the cost of data access and transfer between storage devices in the memory hierarchy. DSS comprises three types of functional processors: the coordination processor (CP), the staging processors (SP), and the replacement processors (RP). These functional processors cooperate to schedule data staging in the memory hierarchy.

RPS is composed of multiple high-performance microprocessors. This subsystem can perform postoperations for relational algebra to each of the subsegments that has been already arranged for parallel processing in DSS. A typical postoperation is the link operation, which actually concatenates tuples within the subsegment in relation R and tuples within the subsegment in relation S. In this case, both the subsegments roughly are grouped through the grouping filter with the same hash function associated with joining attributes. The other postoperation of RPS is duplicate elimination within each subsegment which roughly is grouped through the grouping filter with the hash function for the attribute required by a Projection. Therefore, if different subsegments are assigned to different microprocessors in RPS, these microprocessors can simultaneously process the subsegment assigned to them without the overhead of interprocessor communication. This forms the ideal environment for parallel and asynchronous processing. This two-stage architecture for performing relational algebra can coordinate the data-staging strategy and can obtain the execution time complexity \( O(N/L) \) in the case of Projection and Join, though the other RDBMs can realize \( O(N \times L) \) execution time complexity.
2.2 The Two-stage Execution Architecture of Relational Algebra

This section describes an advanced two-stage relational algebra execution architecture employed by SPIRIT-III. First of all, we discuss the following two architectures introduced by the conventional RDBMs.

1. First-generation architecture. The relational algebra execution architectures are employed by first-generation RDBMs such as DIRECT, RAP, RAP-2, and SPIRIT-I. The architectural feature is the direct execution of original and unordered relations. These architectures, implemented by a multichip microprocessor configuration, attempt to handle directly unordered relations in executing relational operations in parallel. This makes it possible to improve the execution time of relational algebra operations. The major difference between these architectures and Restriction can be executed in \( O(N/L) \) time. However, this type of architecture faces the difficulty of improving the execution time complexity \( O(N \times N) \) of Join and Projection operations. Figure 2(a) shows the basic approach to processing relational operations supported by the early days' RDBMs.

2. Second-generation architecture. Second-generation architecture uses the two-step process in executing relational algebra. In the first step, original relations that are read from the database store are searched and sorted by the dedicated VLSI processor. This processor, composed of many elementary cells, takes \( O(N) \) time in performing the sort operation and selects the tuples that satisfy a given predicate. Next, in the second step, the primary database processor can handle Projection and Join operations to the presorted relations by the sequential algorithm with \( O(N) \). Therefore, the best time performance that we can achieve for Projection and Join operations by means of this type of architecture is totally \( O(N) \).

Figure 2(b) illustrates the basic concept and performance of the presorting method introduced by the second-generation RDBM. The performance of the second-generation RDBM is superior to that of the first-generation RDBM.

3. Third-generation architecture. SPIRIT-III is designed as a third-generation RDBM, which should enhance the capabilities of both relational algebra execution and data staging under an integrated concept. The real performance of the RDBM depends on the efficiency of relational operations and the throughput of data staging in the memory hierarchy.

SPIRIT-III employs two-stage architecture for executing relational algebra. Two-stage architecture is realized by using three major filtering functions. The filtering functions are implemented by the simple processor, called the Tuple Stream Filter (TSF). SPIRIT-III has three types of TSF. These TSFs are attached to all data lines connected between each two levels of the memory hierarchy.

Figure 2(c) shows the basic concept of manipulating relational algebra operations. The major difference between SPIRIT-III and the second-generation RDBM is the function adopted in the first stage. The second-generation RDBM adopted the sort function. However, this concept leads to being faced with the practical implementation issues of the dedicated VLSI sort processor with execution time performance \( O(N) \). One of the implementation issues is the difficulty of developing a sort processor that can support the processing of any relation size, any relation cardinality, and any tuple length in the same fashion. The other is the inefficiency of postoperation to the presorted relation in Projection and Join, because the algorithm of the postoperation is essentially sequential and is not fit for parallel architecture. The third-generation RDBM, SPIRIT-III, is more advanced, because the architecture can support the processing of any size and cardinality of relation and any tuple length in the same manner and also manipulate the preprocessed relation in parallel without any overhead. Moreover, this architecture is very suitable for coordinating a data-staging subsystem introducing a memory hierarchy.

In this paper, we propose the powerful and flexible two-stage architecture, which uses the filtering functions embedded in each stage of memory hierarchy.
The proposed three filtering functions, as shown in Figure 3, are the following:

1. The tuple selector function: This function is realized by the Selecting Tuple Filter (STF), as shown in Figure 3(a), which is one type of TSF. This function is the most primitive that selects the tuples that satisfy a given predicate. By means of this function, the most important relational operations, such as Selection and Restriction, can be completely achieved.

2. The attribute selector function: This function is realized by the Selecting Attribute Filter (SAF), as shown in Figure 3(b), which is one type of TSF. This function can filter the attributes needed by the next relational operations to be fired and repack each tuple, including items of only necessary attributes. This function performs a relational algebra operation, such as Projection, excluding duplicate elimination. This function contributes to saving work space and results in enhancing the performance of executing the relational algebra operations.

3. Segment-grouping function: This function is the key idea of SPIRIT-III architecture and is realized through the grouping filter, as shown in Figure 3(c). This grouping function is the most primitive suboperation of relational algebra operations, such as Projection and Join. This function groups input relations into subrelations. Next, in the case of Projection, each microprocessor of RPS performs a postoperation that is a complete duplicate elimination. RPS can process the assigned subrelation in parallel without the overhead of interprocessor communication. (The overhead of interprocessor communication degrades performance.) In the case of Join, each microprocessor of RPS handles the postoperation that is a concatenation of tuples in relation \( R \) and tuples in relation \( S \). RPS can perform this link operation between tuples in subrelation \( R_i \) and tuples in subrelation \( S_j \), both of which are grouped through the same filtering function with each of the joining attributes, because this grouping function guarantees that both subrelations contain same value items of joining attributes. Tuple grouping filters, realizing this function with a refined method of hash function, are appended to all data transfer lines linked between each stage of the memory hierarchy system in SPIRIT-III. Therefore, if all data need this preprocess, the data are grouped through the filter in passing on data lines in the memory hierarchy before the data are staged up to primary database microprocessors.

Without any overheads resulting in poor performance, SPIRIT-III, which can efficiently process set level relation operations, is the most powerful and practical machine designed on the basis of the state of the art.

3. RELATIONAL ALGEBRA EXECUTION ARCHITECTURE

3.1 Tuple Stream Filter

We have designed the following functional Tuple Stream Filters, as shown in Figure 3, which should be incorporated in the DSS.

1. The Selecting Tuple Filter (STF): This filter functions to yield a "horizontal" subset of a given relation, that is, a subset of tuples within input relation that satisfy a specified predicate. Figure 3(a) illustrates the symbol and function of the selecting tuple filter. This filter can perform the same relational algebraic operations such as Selection and Restriction, when tuples stream through STF.

2. The Selecting Attribute Filter (SAF): This filter functions to yield a "vertical" subset of input relation, that is, that subset obtained by selecting specified attributes, provided that the filter does not eliminate duplicate tuples within attributes selected. This filter can perform to repack items of specified attributes in a tuple into a new tuple without duplicate elimination. This filter is shown in Figure 3(b).

3. Tuple Grouping Filter (TGF): This filter functions to yield subrelations (subsegments), each of which is a subset of an input relation, which consists of tuples including the same specified attribute item. In practice, TGF is implemented by using the hash mechanism in SPIRIT-III. Figure 3(c) displays the symbol and function of TGF. Figure 4 shows a two-stage relational architecture using tuple stream filters embedded in the DSS.

3.2 Relational Algebra Processing

In this section, we explain processing methods of typical relational algebra operations.

1. Selection processing: Selection operation is the simplest but most important operation of relational algebra. In
SPIRIT-III, most Selection operations are performed while tuples are passed through TSFs attached on data channels in the memory hierarchy. This means that the execution time of Selection is actually overlapped with the data transfer time. Therefore, in SPIRIT-III, it is not necessary to consider Selection processing time in order to measure the total performance, because most Selections are performed while relations stream on data channels. Restriction is performed on the basis of the same method.

2. Projection processing: The Projection operator contains two major operations, which are to select specified attributes or a given relation and then to eliminate duplicate tuples within attributes selected. The first operation is completely performed through ASFs. Therefore, the processing time for the first operation of most Projections is overlapped with the data-staging time. TGF is responsible for performing the function to partition a relation into subrelations. TGFs distributed in DSS can roughly group an input relation into subrelations, using the hash mechanism, while original relations are passed through TGFs. Then each microprocessor for postoperation in RPS can eliminate completely duplicate tuples within the grouped subrelation assigned to it. In this case, the processing time associated with the hash function can be made very short by using specialized hardware.

Moreover, if TGF groups the relation composed of \( N \) tuples and \( K \) specific values into \( M \) subsegments, and the number of microprocessors in RPS is \( L \), each grouped subsegment \( j \) contains \( (N/M) \) tuples and a few \( (K = K/M) \) specific values. This makes each microprocessor execute the grouped subsegment assigned by the algorithm with time complexity \( O(N)/(K + 1)/2) \), provided that \( N \) is the number of tuples within the grouped subsegment and \( K \) is the number of unique items in subsegment \( j \). The best time performance of Projection that SPIRIT-III can achieve is near \( O(N/M) \). This two-stage mechanism succeeds in gaining the time performance \( O(N/M) \) for most Projections in principle. This performance is extremely superior to the performance \( O(N \times (K + 1)/(2 \times L)) \) achieved by early RDBMs and the performance \( O(N) \) by the second-generation RDBM.

3. Join processing: Join is the key operation guaranteeing the flexibility and powerful capability of the relational model, but it is the most important operation that must be solved by the system designer. SPIRIT-III can perform the Join operation using the same mechanism for Projection. We can explain the equi-Join processing mechanism by giving an example: Relation \( R \) and relation \( S \) stage up from database store to staging buffer; Relations \( R \) and \( S \) are grouped into Subrelations \( \{R_i\}, \{S_j\} \) through TGF over joining attributes \( R.A \) and \( S.A \). Therefore, subrelations \( R_i \) and \( S_j \) consist of tuples containing the same items within the joining attributes \( R.A \) and \( S.A \). Because the processing time associated with this grouping function is overlapped with the data-staging time, the processing time to group a relation into subrelations through TGF can be ignored. This advantage results in reducing the processing time of Join. Moreover, in the next stage, SPIRIT-III employs the subsegment allocation strategy that both the grouped subrelations \( R_i \) and \( S_j \) are assigned to the same microprocessor of RPS. In case of equi-Join, each microprocessor can perform the actual concatenation of tuples between the grouped subsegments \( R_i \) and \( S_j \) without the access to other subrelations \( R \) \((i \neq j)\). This capability can make possible an ideal parallel, asynchronous processing environment, which contributes to a remarkable enhancement of the time performance of postoperations associated with Join in the second stage. In order to evaluate the time complexity of Join processing in the SPIRIT-III environment, we assume the following:

1. Relations \( R \) and \( S \) contain \( N \), and \( N \), tuples, respectively.
2. TGF groups them into \( M \) subrelations by each of the joining attributes \( R.A \) and \( S.A \).
3. Each subrelation is exhibited as \( R_i \) and \( S_j \), respectively.
4. \( R_i \) and \( S_j \) include a few unique items within each joining attribute that are exhibited as \( K_{r.i} \) and \( K_{s.j} \), respectively.

In case of equi-Join processing, the processing time required in the first stage in DSS is actually hidden within data transfer time in the memory hierarchy. In the second stage, if both \( R_i \) and \( S_j \) subrelations are assigned to the work space of the same microprocessor in RPS, each microprocessor can concatenate between tuples of \( R_i \) and tuples of \( S_j \) in parallel without the overhead of interprocessor communication. In this case, the processing time complexity of the second stage in RPS is \( O((N/(K + 1))K_i)/2) \) or \( O((N/(K + 1))K_j)/2) \). In
practice, this time complexity is nearly \( O(N_r \times u) \) or \( O(N_s \times v) \) because \( K_r \) and \( K_s \) are much less than \( N_r \) and \( N_s \), respectively.

This means that SPIRIT-III succeeds in executing equi-Join with time complexity \( O(\text{Min}(N_r / K_r, N_s / K_s)) \). The relational algebra architecture employed by SPIRIT-III is very superior to the first-generation architecture of time complexity \( O(N \times N/L) \) and the second-generation architecture of time complexity \( O(N) \).

### 3.4 Implementation Architecture

Figure 6 illustrates an implementation architecture of the proposed SPIRIT-III. SPIRIT-III employs large-capacity moving head disks with a track-in-parallel read/write mechanism as database stores, and a number of VLSI random-access memory chips as a staging buffer, which is used as a disk cache and tuple clustering memory.

This track-in-parallel read/write mechanism makes it possible to remarkably reduce the data transfer time between SB and DBS and to efficiently filter the relations that stream on parallel transfer lines.

Double-loop configuration is introduced as the interconnection of banks of SB and microprocessors of RPS. Each of the grouped subrelations in each bank of SB is allocated to a corresponding microprocessor through the double loops.

Figure 7 illustrates the detailed interconnection of the staging buffer and the Tuple Stream Filter. The Tuple Stream Filter Controller (TFC) issues the filtering conditions to each

In this case, Query 1 may be expressed in SEQUEL as follows.

```
SELECT SNAME
FROM S
WHERE CITY = "London" and S# IS IN
(SELECT S#
FROM SP
WHERE P# = "P2")
```

The SEQUEL compiler directly translates Query 1 into the relational algebra tree representation, as shown in Figure 5a. However, the relational algebra tree representation shown in Figure 5b is not always the most efficient execution sequence. We can translate this relational algebra tree into the more efficient execution sequence tree, according to the optimization policy proposed by Smith. Figure 5b shows the optimized relational algebra tree from the query of Figure 5a. It is beneficial to move Selection operations as far down the tree as possible using such transformations. This is because the Selection operations reduce the number of tuples to be processed by subsequent operations. Any reduction is particularly advantageous when Join and Projection operations occur later. There are also benefits to be gained by moving Projection operations down a query tree. Projection operations decrease the width of tuples and, due to the elimination of duplicate tuples, may also decrease the number of tuples in a relation. Each of the new Projection operations perform selecting attributes needed by subsequent operations and does not eliminate the duplicate tuples input relation. Therefore, this transformation remarkably increases the efficiency of executing Query 1.

Next, the optimized relational algebra tree of the Query 1, as shown in Figure 5b, is smoothly translated into the execution procedure with the tuple stream filters of SPIRIT-III, as shown in Figure 5c.

The relational algebra execution architecture of SPIRIT-III harmonizes with the execution sequence expressed by the optimized relational algebra tree and can powerfully support this optimization concept.
Tuple Stream Filter. TGF receives the grouping function and attribute IDs to be grouped.

The retrieval conditions of Selection and Restriction are set to the STF by the TFC, and Selecting attribute IDs are sent to the ASF.

As shown in Figure 6, TGF appends the subsegment number to the tuple when the tuple is passed through TGF, and then the staging buffer management processor actually clusters the tuples, using the attached subsegment number.

4. DATA STAGING ARCHITECTURE

4.1 Basic Concept and Implementation of ISR Policy

The look-ahead data staging policy depends on the following three properties of relational database environments:

1. Possibility of complete recognition of all data used by a transaction: The expression of query is nonprocedural, and the relations that are used by a transaction are declared explicitly.
2. Homogeneity of pages in a relation: Based on set theory, each page in a relation is operated equivalently.
3. High flexibility of execution sequence: Execution sequence of each leaf of a query that is expressed in the relational algebra tree has little restriction.

From properties 1 and 3, DSS can schedule data staging with the original strategy, taking physical environment such as storage device characteristics into consideration. Property 2 guarantees that there is no useless data transfer.

The example transaction shown in Figure 8 helps to explain the detail of the look-ahead data staging and the data-driven process scheduling.

The cost of a seek operation is the most expensive cost in RDBM that employs a moving head disk with the tracks-in-parallel read/write mechanism as DBS. In a conventional system, data staging is performed according to the execution sequence of a transaction: thus in this example, the data staging is performed in the order of A, B, C, and D, out of accordance with the data arrangement on the disk. As a result, the cost of data staging is high.

On the other hand, RDBM employing the look-ahead data staging mechanism performs the data staging with the strategy of minimizing data access cost: thus in this example, data staging is performed in the order D, A, B, and C, according to the data arrangement; and as a result, the cost of data staging is minimized.

Because of data-driven process scheduling, \( P_1 \) becomes executable when both \( A \) and \( B \) are staged up to the processor memory, \( P_2 \) becomes executable when \( C \) is created by \( P_1 \), \( P_3 \) becomes executable when \( F \) is created by \( P_2 \), and \( P_4 \) becomes executable when \( G \) is created by \( P_3 \) because \( D \) has already been staged up. Finally, \( P_5 \) becomes executable when \( H \) is created by \( P_4 \). Staged up by the look-ahead data staging.
mechanism. D stays until $P_a$ is completed by the processing subsystem. At peak activity, data like D increase in each level of the memory hierarchy. If staging based on the same strategy is continued, the processor memory will be filled with only this data. In this situation, the data replacement in the pro-

cessor memory is needed in order to continue the execution of the processor even if the data replacement causes overhead. To avoid the overhead, it is necessary to change the strategy. To realize this, DSS must distinguish the data required by an executable process (active data) from the data required by an unexecutable process (inactive data).

The DSS takes the following two actions:

1. Sort the staging requests in staging classes.
   - SC1: Class of stage up (SU)-request issued by active data
   - SC2: Class of SU-request issued by inactive data
   - SC3: Class of stage down (SD)-request
2. Classify the memory blocks in $MH_a$ into replacement states based on replacement cost.
   - RS1: State of block occupied by data issuing SC1 request
   - RS2: State of block occupied by data issuing SC2 request and having no copy in $MH_a$
   - RS3: State of block occupied by data issuing SC3 request
   - RS4: State of block occupied by data issuing SC2 request and having its copy in $MH_a$
   - RS5: Otherwise
Utilizing this information and mechanism, data stagings between each level can be executed synchronously. In addition, the optimal scheduling, taking physical environment into consideration, is performed at each level. Thus, the look-ahead data-staging policy attains high performance of data staging in the relational database environment.

To achieve the maximum advantage of data staging, the processing subsystem should use data-driven process scheduling in addition to DSS using the look-ahead staging mechanism.

4.2 Architecture of a Look-ahead Data Staging Subsystem

The data staging subsystem based on the ISR policy is different from the conventional memory management system in regard to management policy of data transfer. In the conventional memory management system, every reference by process is directed to the processor memory and causes a transfer among levels of memory hierarchy if the data do not exist in the processor memory; therefore the degree of freedom in scheduling the sequence of data transfer is strongly restricted. On the other hand, in DSS, all data transfers are scheduled by the independent strategy in each level of memory hierarchy; therefore, to enhance the capability of data staging, it is necessary to construct DSS that can asynchronously perform data staging between each level. As a result, DSS is composed of multiple processors, as shown in Figure 1.

(1) The Coordination Processor (CP)

The coordination processor (CP) functions as the interface to ISS and coordinates the processors that schedule the sequence of data staging between levels of the memory hierarchy. For example, the CP registers stage up-requests issued by data used by the transaction in the data-staging pool into the staging queue of higher (or lower) levels when SP completes the data staging.

(2) The Staging Processor (SP)

The staging processor SP(i,i+1) manages data staging between MH and MH(i+1), using the staging queue SQ(i,i+1). All staging requests between MH(i) and MH(i+1) are put into SQ(i,i+1) by CP. The SP(i,i+1) selects the staging request to be served according to the algorithm. After this operation, SP(i,i+1) requires RP(i) to select the blocks to be replaced. The required scheduling policy for each SP is different from other SPs. Generally, the functions of SP fully depend on the storage devices of memory hierarchy.

The operations of a moving head disk consist of Seek, Search, and Transfer operations. The systems employing a moving head disk with a tracks-in-parallel read/write mechanism tend to increase the effect of seek time on total I/O processing time. Therefore, MINIMUM ACCESS COST DATA of the SP(i,i) in the algorithm is selected by using the disk-scheduling algorithm that could minimize seek time.

In most conventional systems, the FCFS (first come, first served) policy is employed as the disk scheduling policy however, to enhance the efficiency of the Seek operation, SSTF (shortest seek time first), SCAN, and CSCAN (circular SCAN) policies have been developed and proposed. The important nature of disk scheduling policy, except the FCFS policy, is that as the number of I/O requests increases, so does the efficiency of the operation.

This suggests that DSS employing the look-ahead data-staging mechanism pulls out the maximum effect of disk-scheduling policy, except the FCFS policy, because all SU requests of transaction data are registered en bloc to the data-staging pool when a transaction enters RDBM.

(3) The Replacement Processor (RP)

The replacement processor for the staging buffer manages by using the memory map table (MMT). The memory map table contains the information of every block in every blank of the staging buffer: the identifier and the replacement state of the data stored in the block. When SP(i,i+1) (or SP(i-1,i)) requests RP(i) to select the block to be replaced, RP searches the block to be replaced in the order of blocks in RS1, RS2, and RS3. If there are only RS1 blocks, RP keeps SP(i,i+1) (or SP(i-1,i)) waiting until any block becomes a replaceable state.

Our performance evaluation by computer simulations shows that the proposed architecture improves both system throughput and response time of transaction.

5. CONCLUDING REMARKS

This paper has proposed an advanced architecture of the relational database machine, named SPIRIT-III, which is basically organized into a three-level memory hierarchy with sophisticated data-staging architecture and preprocessing architecture for relational algebra. The memory hierarchy system is composed of workspace of primary database processors, a staging buffer as the intelligent disk cache, and moving arm disks as the database store.

SPIRIT-III aims at totally improving both I/O and CPU processing boundaries, which are key problems in implementing RDBMs; it has two major architectural features. One is the introduction of the relational-database-oriented data-staging mechanism, called the look-ahead data-staging mechanism, which can optimally schedule data access and transfer in the memory hierarchy. The other is that it attaches refined preprocessing mechanisms for relational algebra operations to data transfer lines connected between each two levels of memory hierarchy. When a relation stages up or down in the memory hierarchy, this preprocessing mechanism can function to select tuples and attributes and to rearrange original relations for parallel processing. SPIRIT-III provides three basic preprocessing functions. Two of these are the tuple selector, filtering only tuples to match retrieval conditions; and the attribute selector, repacking only attributes needed by executing a query. In addition to these filtering functions, SPIRIT-III incorporates the grouping filter proposed by the
authors into each stage of the memory hierarchy. The operation of the grouping filter is the primitive preprocessing operation for executing Join and Projection. In practice, the grouping filter implemented with hash function rearranges original relations and decomposes these into subsegments. Without the overhead of interprocessor communication, each database microprocessor can execute postoperations of relational algebra to a few subsegments assigned to it in parallel. The postoperations include duplicate elimination within the subsegment and concatenation operation between tuples within the subsegment in relation R and the tuples within the subsegment in relation S. Each subsegment is clustered by the same hash function while the original relation stages up.

Therefore, SPIRIT-III can smoothly perform Join and Projection operations by $O(N/L)$, whereas the early RDBMs required $O(N \times N/L)$. The proposed SPIRIT-III covering from data-staging architecture to relational algebra execution architecture under the integrated concept is one of the most practical and powerful RDBMs based on the state of the art.

ACKNOWLEDGMENTS

The authors gratefully acknowledge Professor H. Aiso of Keio University for his encouragement and valuable advice. They also thank members of the Mind group of Keio University for their valuable discussion and members of Ichikawa Laboratory of Hiroshima University for their encouragement.

REFERENCES