データベース・システム　83－3
（１９９１．５．２４）

関数型並列問い合わせ処理における資源割り当てアルゴリズムの拡張

清木　康　大西　元
筑波大学　電子・情報工学系

Abstract

関数型並列データベース・システムＳＭＡＳＨは、データベースの多様な応用分野に適応するための機能群を提供する並列処理システムである。このシステムの特徴は、応用分野に応じたデータベース演算およびデータ構造を定義し、また、それらの演算を組み合わせることにより生成される問い合わせを並列に処理する環境を提供するために、関数型プログラミングの概念を適用している点にある。また、演算の対象データ群をストリームとして扱う点も特徴である。本稿では、このシステムにおいて、任意のデータベース演算から構成される問い合わせを、共有メモリ資源を用いて並列に処理する場合における最適なメモリ資源割り当て方式について述べる。この方式は、関数として表現された任意のデータベース演算により構成される問い合わせについて、最適なメモリ資源割り当ての獲得を保証する。

Extension on the resource allocation algorithm in parallel and functional query processing

Yasushi KIYOKI and Hajime Ohnishi
Institute of Information Sciences and Electronics
University of Tsukuba
Tsukuba, Ibaraki 305, Japan

Abstract

We have proposed a stream-oriented parallel processing scheme and a parallel processing system SMASH for a wide variety of database applications. The main feature of the scheme and the system is functional programming concepts applied to define new database operations and data types and exploit parallelisms inherent in an arbitrary set of database operations. In this system, it is important to perform query optimizations for arbitrary combination of various database operations. This paper deals with optimal memory allocation for a query consisting of an arbitrary set of database operations in a shared memory resource environment. We propose an optimization method for memory resource allocation in the stream-oriented parallel processing scheme. This method guarantees to obtain the optimal memory allocation for any query consisting of an arbitrary set of database operations in stream processing.

1 Introduction

In relational database systems, as a set of database operations and data structures are fixed, memory resource allocation can be considered for the algorithms of those database operations [6]. In those systems, system architectures have been designed for supporting only a fixed set of database operations and data types, and as a result, several algorithms, parallel machine architectures, access methods, query optimization technics have been designed.

On the other hand, in advanced database systems providing facilities for defining new data types and database operations, database operations are not fixed in advance in the design of systems. In those systems, as specific algorithms or access methods cannot be applied to those database operations, system design strategies for supporting arbtrary database operations are required. We apply functional programming concepts [5] to define new database operations and data types and exploit parallisms inherent in an arbitrary set of database operations. We have proposed a stream-oriented parallel processing scheme and a parallel processing system SMASH for a wide variety of database applications [1] [2]. In this system, arbitrary database operations are defined as functions which manipulate databases and intermediate data as streams. Those functions are executed in parallel by demand-driven evaluation incorporating stream processing.

As arbitrary database operations are used in a query, specific memory resource allocation for specific database operations can not be applied. That is, it is necesarry to provide a general method for memeory resource allocation.

In this system, it is important to perform query optimizations for arbitrary combination of various database operations [3][4]. In this paper, we present an optimal memory allocation method for a query consisting of an arbitrary set of database operations in a shared memory resource environment. We propose an optimization method for memory resource allocation in the stream-oriented parallel processing scheme. This method guarantees to obtain the optimal memory allocation for any query consisting of an arbitrary set of database operations in stream processing.

2 The computation method for memory resource allocation

In the stream-oriented processing scheme, grain size(granularity) settings are the most important factor for the total amount of computation times and the degree of parallelism. Granularity settings correspond to shared-memory resource allocation for buffers which are used in computations for database operations. In the stream-oriented processing scheme, the shared-memory resource allocation problem is to fix the optimal granularity settings(buffer size settings) to minimize execution time of a query. In this section, we present a computation method for obtaining the optimal allocation of shared-memory resources in query processing.

2.1 Memory resource allocation

The shared-memory resource allocation problem is formalized by using the following parameters and formulas.

m: the number of buffers using the shared-memory resources

x_i $(i = 1, 2, \cdots, m)$: variables representing each buffer size

$f(x_1, x_2, \cdots, x_m)$: function representing query execution time

BS: the total amount of the shared-memory resources

The shared-memory resource allocation problem is defined as follows:

Restriction

$$\sum_{i=1}^{m} x_i \leq BS$$

$x_1, x_2, \cdots, x_m$: positive number

Object function

$$y = f(x_1, x_2, \cdots, x_m)$$

The objective is to find the combination in $[x_1, x_2, \cdots, x_m]$ which gives the minimal value of the object function.

In our method, the object function is represented in a combination of sub-formulas. The following symbols are used in setting formulas.

n: the number of formulas for representing the object function.

y_i $(i = 1, 2, \cdots, n)$: value of the formula-i.

$\mathbf{X}_i$ $(i = 1, 2, \cdots, n)$: a set of buffer size variables included in the formula-i.

p_i $(i = 1, 2, \cdots, n)$: the number of buffer size variables included in the formula-i.

q_i $(i = 1, 2, \cdots, n)$: the number of sub-formulas which correspond to the arguments of the formula-i.

$f_i(\mathbf{X}_i, y_{i1}, y_{i2}, \cdots, y_{iq_i})$ $(i = 1, 2, \cdots, n)$: the sub-formula-i.

r: the identifier of the formula which represents (corresponds to) the value of the object function.

C_{ij} $(i = 1, 2, \cdots, n)$ $(j = 1, 2, \cdots, n)$: the variable which represents relationship between sub-formulas.

The object function y is described as follows:

$$y = y_r$$

$$\begin{aligned} y_1 &= f_1(\mathbf{X}_1, y_{11}, y_{12}, \cdots, y_{1q_1}) \\ y_2 &= f_2(\mathbf{X}_2, y_{21}, y_{22}, \cdots, y_{2q_2}) \\ &\vdots \\ y_i &= f_i(\mathbf{X}_i, y_{i1}, y_{i2}, \cdots, y_{iq_i}) \\ &\vdots \\ y_n &= f_n(\mathbf{X}_n, y_{n1}, y_{n2}, \cdots, y_{nq_n}) \end{aligned} \quad (1)$$

The formula y_{ij} represents one of sub-formulas which are constructing the object function.

$$y_{ij} \in \{y_1, y_2, \cdots, y_n\} \quad (j = 1, 2, \cdots, q_i) \ (i = 1, 2, \cdots, n) \quad (2)$$

In the following, the total amount of every element of $\mathbf{X}_i$ is represented as $\sum \mathbf{X}_i$. That is,

$$\mathbf{X}_i = \{x_{i1}, x_{i2}, \cdots, x_{ip_i}\} \quad (i = 1, 2, \cdots, n) \quad (3)$$

$$\sum \mathbf{X}_i = x_{i1} + x_{i2} + \cdots + x_{ip_i} \quad (4)$$

Each formula for constructing the object function has the following restrictions(characteristics).

The following conditions mean that the graph of formulas is the form of a tree.

$$\sum_{i=1}^{n} q_i = n - 1 \quad (5)$$

The value of the variable C_{ij} $(i = 1, 2, \cdots, n)$ $(j = 1, 2, \cdots, n)$ is fixed as follows:

If the value of y_v is used as a parameter in the formula y_u, then $C_{uv} = 1$, and otherwise, $C_{uv} = 0$.

$$\forall u, v \ u \neq v, \exists \alpha_1, \alpha_2, \cdots, \alpha_k$$
$$C_{u\alpha_1} \times C_{\alpha_1\alpha_2} \times \cdots \times C_{\alpha_k v} = 1 \quad (6)$$

Suppose the graph whose node corresponds to a formula and whose arc corresponds to a relationship

on the reference of a parameter between two formulas. The condition (1) represents that the number of arcs is one less than the number of nodes. The condition (2) represents that every node is connected to a graph. Therefore, these conditions represent that the structure of the graph is a tree.

Every buffer size variable must be included in formulas, and if referential relationships between formulas are not existing, any buffer size variables $\mathbf{X}_i$ do not appear both of those formulas. These conditions represent that the formulas correspond to pure functional operations, that is, referential transparency is guaranteed among functions which correspond to formulas.

$$\bigcup_{i=1}^{n} \mathbf{X}_i = \{x_1, x_2, \cdots, x_m\} \quad (7)$$

$$\forall u, v \; C_{uv} = 0 \;\; \mathbf{X}_u \cap \mathbf{X}_v = \emptyset \quad (8)$$

The following summerizes the parameters.

s_i: The total amount of memory resources which are used to compute y_i.

$\mathbf{X}'_i$: A vector of buffer size variables which are used in both the formula y_i and the other formula which references the variables.

X'_i: The set of buffer size variables which are included in $\mathbf{X}'_i$.

If the buffersize variables $\mathbf{x}_{ij}$ are used in the formulas y_{ij}, the buffer size variables $\mathbf{X}'_{ij}$ represent the variables which are used in both formulas y_{ij} and y_i. That is,

$$\mathbf{X}'_{ij} = \mathbf{X}_i \cap \mathbf{X}_{ij} \quad (j = 1, 2, \cdots, q_i) \quad (9)$$

The minimal value $Y_i(s_i, \mathbf{X}'_i)$ of the formula y_i is represented as follows.

$$Y_i(s_i, \mathbf{X}'_i) = \min_{\substack{0 \le \sum \mathbf{X}_i + \sum_{j=1}^{q_i} (s_{ij} - \sum \mathbf{X}'_{ij}) \le s_i \\ 0 \le \sum \mathbf{X}'_{ij} \le s_{ij} \;\; (j = 1, 2, \cdots, q_i)}} f_i(\mathbf{X}_i, Y_{i1}(s_{i1}, \mathbf{X}'_{i1}), \cdots, Y_{iq_i}(s_{iq_i}, \mathbf{X}'_{iq_i})) \quad (10)$$

2.2 Algorithm

In this section, we present an algorithm to solve the problem of memory resource allocation. (This algorithm is based on dynamic programming.) In this algorithm, computations proceed from leaf nodes to a root node of a graph. An allocation candidate table is created as a result of the computation of each node. The table is driven from leaf nodes to the root node with adding allocation information.

The allocation candidate table represents the minimal values on each allocation candidate. the minimal value of each formula can be represented as show in the formula (1). This table consists of the candidate sizes for allocation of memory resources(size), minimal values of formulas(cost), and values of parameters(list).

As computations for the values of formulas proceed in bottom-up, the values of upper nodes are not used in lower formulas.

[Algorithm]

[1] Allocation candidate tables are created for each leaf-node of a graph. Values($y_i = f_i(\mathbf{X}_i)$) of a formula are computed for each candidate buffer size, and the pairs of the candidate buffer size and its corresponding value of a formula are registered in the allocation candidate table.

[2] Allocation candidate tables are created for the non-leaf formulas by referring to the created allocation candidate tables.

[3] The step [2] is repeatedly performed until the value of root node(the object function) y_r is computed and the final allocation candidate table for the y_r is created.

[4] By referring to the final allocation candidate table, the candidate buffer allocation which corresponds to the minimal value of the object function is obtained. The obtained buffer allocation is the result of the optimal memory resource allocation.

The algorithm for creating an allocation candidate table is shown in Fig.1. This algorithm is used to create an allocation candidate table $table_i$ of y_i by referring to the allocation candidate tables $table_{ij}$ $(j = 1, 2, \cdots, q_i)$ which correspond to arguments y_{ij} $(j = 1, 2, \cdots, q_i)$ of y_i, as shown in Fig.2.

BS: the total amount of shared memory resources.

$\mathbf{X}_i$ $(i = 1, 2, \cdots, n)$: the set of buffer size variables included in each formulas.

p_i'' $(i = 1, 2, \cdots, n)$: the number of buffer size variables which firstly appear in the formula y_i.

q_i $(i = 1, 2, \cdots, n)$: the number of formulas which are referenced in the formula f_i.

x_{ij}'' $(j = 1, 2, \cdots, p_i'')$: buffer size variables which firstly appear in the formula y_i.

f_i $(i = 1, 2, \cdots, n)$: the formulas constructing the object function.

$table_{ij}$ $(j = 1, 2, \cdots, q_i)$: the table corresponding to the formula y_{ij}.

len_{ij} $(j = 1, 2, \cdots, q_i)$: the number of entries included in the table $table_{ij}$.

idx_{ij}: the variable for indicating an entry in the table $table_{ij}$.

$table_i$: the table for storing the intermediate result.

K_i: the set of buffer size variables which appear in both the current and upper formulas in x_{ij}'' $(j = 1, 2, \cdots, p_i'')$.

L_i: the set of buffer size variables which do not appear in teh upper formulas in x_{ij}'' $(j = 1, 2, \cdots, p_i'')$.

```
type  Entry = {
            size :  integer
                /* the amount of memory resources appearing in the computation of a formula*/
            cost :  real   /* the minimal value of the formula in using the memory resources size.*/
            X'   :  set of (buffer size variable, its value)
                /* the combinations of buffer size variable and its candidata values */
                /* which also appear in the upper formula in x''ij  (j = 1,2,...,p''i).*/
            list :  set of (buffer size variable, its value)
                /* the combinations of buffer size variables and their candidate values */
                /* which appeared in the lower formulas.*/
}
```

input : table of Entry $table_{ij}$ $(j = 1, 2, \cdots, q_i)$

set of int K_i, L_i

output : table of Entry $table_i$

/* At this point, no entry is included in the $table_i$.*/

FOR (all combinations of a buffer size candidate value in the range of $0 \leq x''_{ij} \leq BS$ $(j = 1, 2, \cdots, p''_i)$ and a table entry in the range of $1 \leq idx_{ij} \leq len_{ij}$ $(j = 1, 2, \cdots, q_i)$,

which are satisfied with the condition $\sum_{j=1}^{p''_i} x''_{ij} + \sum_{j=1}^{q_i} table_{ij}[idx_{ij}].size \leq BS$

/* Creation of *candidate_entry* in $table_i$.*/

$candidate\_entry.size = \sum_{j=1}^{p''_i} x''_{ij} + \sum_{j=1}^{q_i} table_{ij}[idx_{ij}].size$

$candidate\_entry.cost = f_i(\ x''_{i1}, \cdots, x''_{ip''_i},$

$table_{i1}[idx_{i1}].X', \cdots, table_{iq_i}[idx_{iq_i}].X',$

$table_{i1}[idx_{i1}].cost, \cdots, table_{iq_i}[idx_{iq_i}].cost)$

$candidate\_entry.X' = \{(x''_{ik}, valueof x''_{ik}) \mid k \in K_i\}$

/* Creation of a entry list *list* of entries in $table_i$ */

/* which do not appear in the upper formulas */

/* by referencing entries in lists corresponding to each lower formula.*/

$candidate\_entry.list = \{(x''_{il}, valueof x''_{il}) \mid l \in L_i\} \cup \bigcup_{j=1}^{q_i} table_{ij}[idx_{ij}].list$

IF(no entry (= *old_entry*) of $table_i$ exists which satisfies the condition

$old\_entry.size < candidate\_entry.size$ and

$old\_entry.X' == candidate\_entry.X'$ and

$old\_entry.cost \leq candidate\_entry.cost$

*/

FOR(all entries (= *old_entry*) in the entries included in $table_i$, which satisfies the condition

$old\_entry.size \geq candidate\_entry.size$ and

$old\_entry.X' == candidate\_entry.X'$ and

$old\_entry.cost > candidate\_entry.cost$

)

Delete *old_entry* from $table_i$.

END FOR

Add *candidate_entry* into $table_i$.

END IF

v

Fig 1: The algorithm for creating an allocation candidate table

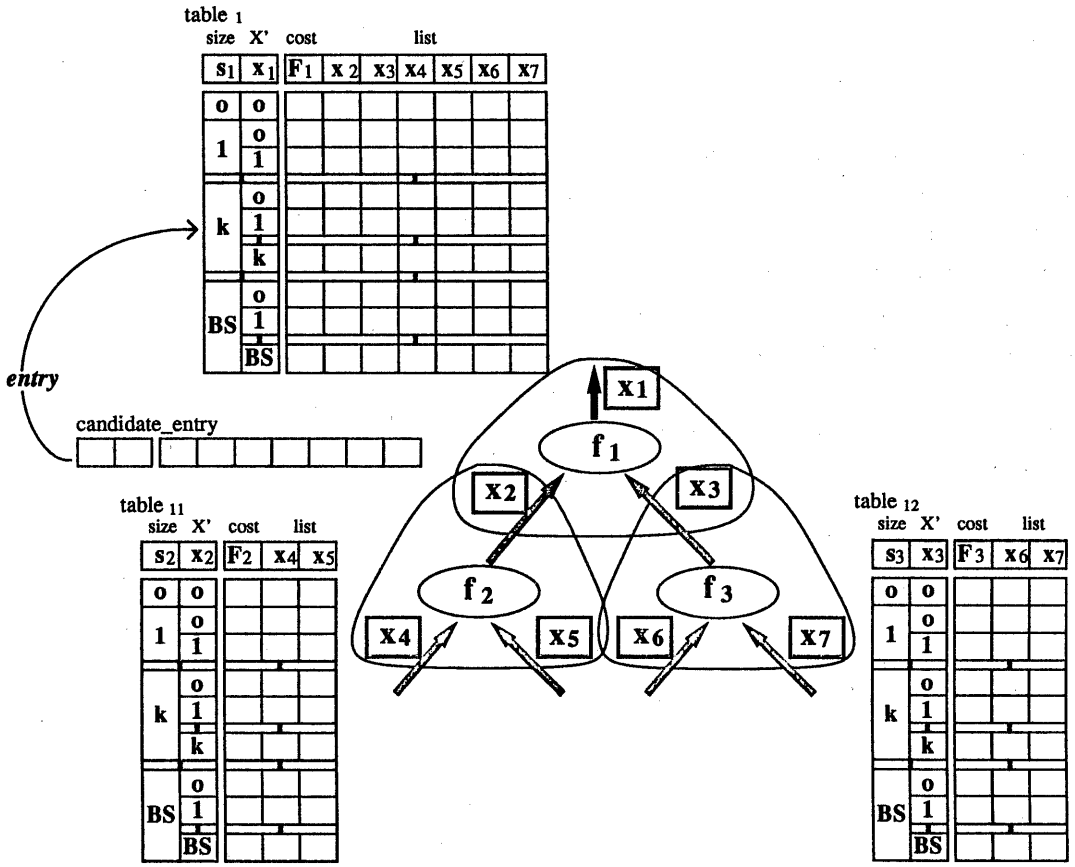

Fig 2: An overview of the algorithm for creating tables.

2.3 Complexity of the algorithm

The complexity of the proposed algorithm is fixed by the number of buffers (m), the total size of memory resources (BS), the number of formulas included in the graph(n), and the number of variables ($p_i, q_i \ \ (i = 1, 2, \cdots, n)$) included in those formulas. Relationships among those elements are shown as follows:

$$\sum_{i=1}^{n}(p_i - \sum_{j=1}^{q_i} m'_{ij}) = m \qquad (11)$$

$$\sum_{i=1}^{n} q_i = n - 1 \qquad (12)$$

In the step [2], the complexity of computations for creating an allocation candidate table is fixed by the number of candidate sizes for a buffer and the number of the arguments of the formula. The size of a buffer does not exceed the total size of memory resources. The number of the arguments corresponds to the number of allocation candidate tables which are referred to in the formula.

Each formula includes $(p_i + q_i)$ arguments. Computations are performed for every combination of those arguments. Therefore, the complexity of the computations for obtaining minimal values of the formula in each combination is $O(BS^{p_i+q_i})$. The total amount of computations for a graph for obtaining the optimal memory allocation is the sum

of computation times of each formula, that is:

$$O(\sum_{i=1}^{n} BS^{p_i+q_i}) \quad (13)$$

This complexity means that the computation times decrease with the decrease of the maximal number of arguments(p_i+q_i). If the number of formulas included a graph is increased, the maximal number of arguments becomes smaller, and as a result, the computation times can be decreased. That is, if the object function is constructed with formulas as many as possible, the computation times can be decreased.

3 Formula settings of query execution time

In this section, we present the method for defining object functions. In Section 3.1, a guide for constructing an object function is discussed. In Section 3.2, the object function which represents the computation times of a query is shown. This object function is applied to memory resource allocation which is used for sequential execution of a query. In Section 3.3, we introduce the method for constructing an object function which is used for parallel execution of a query.

3.1 Definition of the object function

A query is represented in a tree structure. In the tree, a node corresponds to a database operation and the arc corresponds to source or intermediate data which is represented in a form of a stream.

The object function is formalized by using the following two steps:

[1] Formulas for estimating execution time of each database operation are constructed in a unit of a database operation. A formula has two kinds of arguments. One of them is a kind of buffer size arguments. Execution time of an database operation is influenced by the size of buffers which are used to store source or intermediate data. The other is identifiers of other formulas which correspond to lower-sub-tree of the formula itself.

[2] The object function is constructed by combining formulas which are constructed in Step [1].

The object function of a query is formalized by combining formulas each of which corresponds to a database operation. That is, the structure of the object function is the same as that of the query.

3.2 Construction of an object function for sequential execution of a query

In the case of sequential query processing, the execution time of a query T_{QE} is equal to the total computation time of every database operation included in the query. Therefore, the objective is to obtain the resource allocation which minimizes the total computation time.

In the functional computation model, there are two evaluation methods for arguments. Those methods are called call-by-name and call-by-need. In call-by-name, the formal argument is reevaluated whenever the argument is encountered in the function body. In this evaluation, the actual argument can be removed after a reference to it is completed. In evaluating the argument which corresponds to a stream, functional computations can be performed within limited memory resources. However, the number of computations may increase.

In call-by-need, a formal argument is evaluated only once when teh first reference to it is encountered. The evaluated actual argument is then used in the other references to the argument. The actual argument must be retained in teh caching directory until every reference to it is completed. If the actual argument is a stream, the memory could be

swamped. However, reexecution of the same function is unnecessary.

For each of these evaluation methods, we set up formulas which represent the total amount of computations in a sub-tree.

m_i the number of parameters

$x_{i1}, x_{i2}, \cdots, x_{im_i}$ buffer size

$y_{i1}, y_{i2}, \cdots, y_{im_i}$ computation time(evaluation time) of an argument

$g_i(x_{i1}, x_{i2}, \cdots, x_{im_i})$ execution time of the database operation itself

$n_i(x_{i1}, x_{i2}, \cdots, x_{im_i})$ the number of references of an argument

$c_{i1}, c_{i2}, \cdots, c_{im_i}$ the number of re-reference times of an argument in call-by-need

[1] In the case of call-by-name

$$\begin{aligned} & f(x_{i1}, x_{i2}, \cdots, x_{im_i}, y_{i1}, y_{i2}, \cdots, y_{im_i}) \\ &= g(x_{i1}, x_{i2}, \cdots, x_{im_i}) \\ &+ h_1(x_{i1}, x_{i2}, \cdots, x_{im_i}) \times y_{i1} \\ &+ h_2(x_{i1}, x_{i2}, \cdots, x_{im_i}) \times y_{i2} \\ &\vdots \\ &+ h_m(x_{i1}, x_{i2}, \cdots, x_{im_i}) \times y_{im} \end{aligned} \quad (14)$$

[2] In the case of call-by-need

$$\begin{aligned} & f(x_{i1}, x_{i2}, \cdots, x_{im_i}, y_{i1}, y_{i2}, \cdots, y_{im_i}) \\ &= g(x_{i1}, x_{i2}, \cdots, x_{im_i}) \\ &+ y_{i1} \\ &+ (x_{i1}, x_{i2}, \cdots, x_{im_i}) - 1) \times c_{i1} \\ &+ y_{i2} \\ &+ (x_{i1}, x_{i2}, \cdots, x_{im_i}) - 1) \times c_{i2} \\ &\vdots \\ &+ y_{im} \\ &+ (x_{i1}, x_{i2}, \cdots, x_{im_i}) - 1) \times c_{im} \end{aligned} \quad (15)$$

3.3 Construction of an object function for parallel execution of a query

The number of database operations used in a query and the number of processors for query processing are constant. In this case, the total execution time T_{QE} of a query is represented as the sum of execution time T_{PE_j} and idle time T_{PI_j} of a processor.

$$T_{QE} = T_{PE_j} + T_{PI_j} \quad (j = 1, 2, \cdots, N_P) \quad (16)$$

The total utilization time T_Σ of all processors is defined as the product of the execution time T_{QE} of a query and the number of processors N_P. The total utilization time is equal to the sum of the total execution time $\sum_{j=1}^{N_P} T_{PE_j}$ and the total idle time $\sum_{j=1}^{N_P} T_{PI_j}$ of all processors.

$$\begin{aligned} T_\Sigma &\equiv N_P \times T_{QE} \\ &= \sum_{j=1}^{N_P} T_{QE} \\ &= \sum_{j=1}^{N_P} T_{PE_j} + \sum_{j=1}^{N_P} T_{PI_j} \end{aligned} \quad (17)$$

Furthermore, the total computation time $\sum_{i=1}^{N_F} T_{FE_i}$ of all database operations in a query is equal to the total execution time $\sum_{j=1}^{N_P} T_{PE_j}$ of all processors.

$$\sum_{i=1}^{N_F} T_{FE_i} = \sum_{j=1}^{N_P} T_{PE_j} \quad (18)$$

Therefore, the total utilization time T_Σ of all processors is equal to the sum of the total computation time $\sum_{i=1}^{N_F} T_{FE_i}$ of database operations and the total idle time $\sum_{j=1}^{N_P} T_{PI_j}$ of all processors.

$$T_\Sigma = \sum_{i=1}^{N_F} T_{FE_i} + \sum_{j=1}^{N_P} T_{PI_j} \quad (19)$$

As mentioned before, the number of processors does not change during the execution of a query.

Therefore, the execution time T_{QE} of a query is minimal when the total utilization time T_{Σ} of all processors is minimal. That is, the optimal memory resource allocation for a query corresponds to the memory resource allocation in which the total utilization time T_{Σ} of all processors is minimal.

4 Conclusion

This paper has presented an algorithm for memory resource allocation and a method for defining formulas which represent execution time of database operations. By using this algorithm, the optimal memory resource allocation is obtained for a query consisting of arbitrary database operations.

Reference

[1] Y.Kiyoki, K.Kato and T.Masuda, "A Relational Database Machine Based on Functional Programming Concepts", *Proc. ACM -IEEE Computer Society Fall Joint Computer Conf.,* pp.969-978, Nov. 1986

[2] Y. Kiyoki, T. Kurosawa, K. Kato and T. Masuda, " The Software Architecture of a Parallel Processing System for Advanced database Applications," *Proceedings of 7th IEEE International Conference on Data Engineering,* pp. 220-229, Apr. 1991.

[3] P. Liu, Y. Kiyoki and T. Masuda, "A Computation Method for Buffer Resource Allocation in the Stream-Oriented Processing Scheme for Relational Database Operations," *Transactions of Information Processing Society of Japan,* Vol.29, No.8, pp.770-781, 1988.

[4] P. Liu, Y. Kiyoki and T. Masuda, "Efficient algoithms for resource allocation in distributed and parallel query processing environment," *Proceedings of the IEEE International Conference on Distributed Computing Systems,* pp. 316-323, 1989

[5] S. R. Vegdahl, "A Survey of Proposed Architectures for the Execution of Functional Languages," *IEEE Trans. Comput.,* Vol.C-33, No. 12, pp. 1050-1071, Dec. b1984

[6] P. S. Yu and D. W. Cornell, "Optimal Buffer Allocation in a Multi-Query Environment," *Proceedings of 7th IEEE International Conference on Data Engineering,* pp. 622-631, Apr. 1991.