Array Database Internals

V.A. Pavlov <vlad.pavlov24@gmail.com>
B.A. Novikov < b.novikov@spbu.ru >
Saint Petersburg State University,
13B Universitetskaya Emb., St Petersburg, Russia, 199034

Abstract**.** After huge amount of big scientific data, which needed to be stored and processed, has emerged, the problem of large multidimensional arrays support gained close attention in the database world. Devising special database engines with support of array data model became an issue. Development of a well-organized database management system which stands on completely uncommon data model required performing the following tasks: formally defining a data model, building a formal algebra operating on objects from the data model, devising optimization rules on logical level and then on the physical one. Those tasks has already been completed by creators of different array databases. In this paper array formalization, core algebra and optimization techniques are revised using examples of AML, RasDaMan, SciDB – developed array database management systems with different algebras and optimization approaches.

Ключевые слова: array databases; overview; formal array algebra; array query processing; array query optimization; AML; RasDaMan; SciDB

DOI: 10.15514/ISPRAS-2018-30(1)-10

For citation: Pavlov V.A., Novikov B.A. Array Database Internals. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 1, 2018, pp. 137-160. DOI: 10.15514/ISPRAS-2018-30(1)-10

1. Introduction

Recently in many scientific fields database users need to support and process new non-traditional data structures. Among such uncommon structures are different hierarchical structures, graphs, as well as *arrays*. It's worth noting, that such a need is not explained by the subjective preferences of database users, it is fully justified by the real state of things for users and their requirements for processing the data under study. In this paper we will partially consider what is offered to users who need to store and process array data and how storage and processing are made efficient. But first, we let us understand more precisely, with what kind of data such users have to deal with.

The data referred to is also called *multidimensional discrete data* (MDD) or *raster data* [1]. Such data is homogeneous, each element has some index (represented by a vector in a d-dimensional Euclidean space) and, hence, has some adjacent

elements. This data is typically huge. On a more intuitive level, MDD data can be imagined as huge multidimensional cubes. For such a cube, each cell has a discrete multidimensional index and contains a value of a fixed type. An example of a 3D cube may be the following: a series of images[2] obtained from two Huble telescopes cameras for some a period of time, say. a year. As it has been said, in real life those cubes are usually of tera- or even petabyte scale: for instance, Large Hadron Collider (LHC), producing raster cubes during its work, after a day of functioning and filtering produced data generates multidimensional data sizing over 5 terabytes [3].

Granted, such cubes are not just stored as they often demand some kind of analysis. Usually the analysis to be done is not trivial due to the fact that the need for intelligent raster data processing arises in such fields as: natural sciences, medicine, census, multimedia and OLAP. Demand for efficient storage and processing of huge raster data cubes states a problem of devising special tools and algorithms. The specificity of the data in use is another factor increasing the need in a specialized storage systems. Raster data has several peculiarities induced by its properties mentioned above. These peculiarities include: large size of a single raster data value (a single cube may occupy several disk pages instead of a part of a disk page in case of conventional data types, e.g. numeric values); lack of index support due to absence of natural ordering of cubes, etc. Those peculiarities make efficient processing of raster data different from processing conventional data types in terms of storage and optimization techniques.

Fortunately, the problem of optimized storage and efficient processing of raster data has already been faced by authors of special extensions for existing relational/object-relational databases (such as Terralib [4], PostGIS [5], SpatialLite [6], Oracle GeoRaster [7]) and creators of *array databases* standing on specially devised array data models [8]. Today there are several array database management systems (*ADBMS*) such as RasDaMan [9], SciDB [10], which are still maintained and intensively developed with the aim to continuously improve and to conform to rapidly increasing scientific demands. In each ADBMS much attention is paid to optimization as optimizing queries is crucial when processing queries operating with petabyte sized data cubes.

There are two ways of optimization: logical and physical ones. Logical optimization is usually based on formal algebra standing behind the array model. Physical optimization is typically achieved by devising special storage scheme and/or data retrieving order[11], [12]. In the current work we will briefly review theoretical basement and optimization techniques considering three distinct developed ADBMSs : RasDaMan, AML and SciDB.

The reader should be aware of the fact that the aim of this paper is to present and analyze different data models, algebras and optimization techniques used in some array databases and certainly **NOT** to compare those databases in order to determine advantages of one over another. The paper does **NOT** intend to characterize the databases anyhow so that the given characteristics are based on subjective opinions.

2. Diving In

To study the the theoretical basement along with optimization techniques in ADBMSs it is important to understand a generic algorithm following which a new ADBMS can be built.

First, an array term should be formally defined as it is a central object of interest in such systems. Obviously, in array DBMS algebras, the main object of all operations is an *array*. For best of our knowledge, all the existing algebras define an array mostly similarly. Formally, an array is a function defined on index domain D to some value set V. Value sets differ in different systems. Index domain D is represented as Cartesian product of finite amount of ordered sets $I_1, I_2, ..., I_k$. The value k is called an dimensionality ($valence$) of the array. Applying A function to a vector $(i_1, i_2, ..., i_k)$ is associated with getting the array's cell value.

Second, a formal algebra is introduced. Algebras are mathematical structures where several operations with some core objects are defined. Operations with those objects return an object from the same algebra. One of the most important features of algebras is an ability to construct expressions in them by combining application of algebra's operations. As algebras operations are closed, result of an expression evaluation is again an object from the algebra. In simpler words, a formal algebra enables to construct complex expressions value of which do not leave the algebra. In reality in different systems underlying algebras start to differ. Several existing algebras in existing array dbms are described further.

Third, logical optimization rules are introduced. Complex expressions can be overburdened with unnecessary operations and elimination them simplifies the expression benefiting in less execution complexity.

Fourth, physical optimization rules are derived. Logical optimization of an expression is not sufficient for actually executing the query in the most efficient way. In most cases a single query can be executed differently accounting "physical" information which tells how the queried data is actually stored.

Fifth, the query language is introduced to give a user of the system of the system a convenient high-level language taking the user away from lower-level algebra language.

In the current paper we will look at how the first four steps were followed for each of the highlighted array databases, ignoring high level query languages as they do not contribute much to understanding the theoretical essentials of ADBMS.

2.1 Baumann's array algebra

We will now optimization process is organized in RasDaMan ADBMS, explaining its core model and formal algebra - Baumann's array algebra [13]. The overview of optimization techniques is mostly based on PhD thesis [14] of Ronald Ritsch. To present the entire data model the following terms should be explained:

- Multidimensional intervals and spatial domains
- MDD types, values and elementary operations on them
- Derived operations on MDD data
- Extended relational model with MDD support

2.1.1 Multidimensional Intervals and Spatial Domains

An m-interval X is defined as follows: let $D = Z^n$, then $X = [l_1 : h_1,...,l_n : h_n]$ is $\{(x_1, x_2,..., x_n) \in D \mid (l_i \le x_i \le h_i \forall i \in 1..n\}$. Multiple probe functions are defined on the multidimensional intervals:

- Domain function : $dom(X) = D$
- Dimension function : $dim(X) = n$
- Lower bound function : $lo(X) = (l_1,...,l_n)$, where $lo_i(X)$ denotes l_i
- Upper bound function : $hi(X) = (h_1,...,h_n)$, where $hi_i(X)$ denotes h_i
- Extent function : $card(X) = \Pi_{i=1}^{n}(hi_i(X) - lo_i(X) + 1)$

Usually a multidimensional interval represents an index set of a multidimensional array, therefore it is commonly called a *spatial domain*. It is convenient to restrict the possible value type of an interval by introducing a *spatial domain type*. More precisely, a spatial domain type σ^d is a set of admissible d-dimensional domains $\delta^d \subseteq 2^{Z^d}$. Union of all δ^d over all non-negative integers d will be denoted as δ. Then, it is possible to define multiple operations from δ to δ:

- Trimming along dimension i with one dimensional m-interval B
  $trim(X, i, B) = \{(x_1, x_2,..., x_n) \in X \mid lo(B) \le x_i \le hi(B)$
- Slicing along dimension i at point b
  $slice(X, i, b) = \{(x_1,..., x_{i-1}, x_{i+1},...x_n) \mid (x_1,..., x_{i-1}, b, x_{i+1},...x_n) \in X\}$

2.1.2 MDD values and types

An MDD value a over base type T and spatial domain D with $T \in \tau$ and $D \in \sigma$ is a set of (coordinate, value)-tuples defined by a mapping function a: $D \to T$. Then an MDD value a would be $\{(x, a(x)) \mid a(x) \in T, x \in D\}$. Defining an MDD type is accompanied by definition of several probe functions:

- $base(a) = T$

- $sdom(a) = D$

An MDD type M with base type T and spatial domain D with $T \in \tau$ and $D \in \sigma$ is defined as $\{a \mid a$ is MDD value over base type T and spatial domain D, $T \in \tau, D \in \sigma\}$. An MDD type with base type T and spatial domain D is denoted by $[[T, D]]$.

Further, such operation as 'cell access' will be used very frequently on MDD arrays, therefore it should be defined now.

Let $a = \{(x, a(x)) \mid a(x) \in T, x \in D^d\} \in [[T, D^d]]$ be an MDD value and $x \in D^d$ be a d-dimensional point, then the access operator $[]$: $[[T, D^d]] \times (Z)^d \to T$ is defined as: $a[x] = a(x)$.

2.1.3 Core operations

As reported in [14] Baumann's array algebra stands on two basic operators. They are listed below.

- Array constructor $MARRAY(X, x, e_x)$

Let D be an spatial domain, x be a point variable, e_x be an expression with result type T which contain free occurrences of identifier x. Then $MARRAY(X, x, e_x)$ returns an array $a \in [[T, D]]$ for which $A[x] = e_x$.

- Condense operator $COND(op, X, x, e_{x,A})$

Let X be an array, op be a commutative and associative operation on $T \times T \to T$, x be a free identifier, e_x be an expression with result type T, which contains free occurrences of identifier x and MDD array A. Then $COND(op, X, x, e_{x,A})$ returns a scalar value s being equal to $OP_{x \in X} e_{x,A}$

In some works [13], [8], [15] Baumann's array algebra is defined with additional sort operator $SORT(X, i, e)$, which can be defined as follows: let X be an array, i be a dimension number, r be an expression of some type E on which a total ordering is defined. Let S be a one-dimensional array representing a permutation of elements of a set $J = \{lo(X_i), ..., hi(X_i)\}$ sorted by $r(slice(X, i, j)$ where $j \in J$. In less formal words, $SORT$ operator allows to sort hyperplanes of a hypercube along given dimension by value of r expression evaluated on each hyperplane. In [15] it is used for modeling of such a geo-raster operations as $top-k$ and $median$ operations.

It should be mentioned that from optimization point, it is very important to understand what expression e or operation op is provided for an operator to make best possible optimizations. As reported in [14], all operations for $MARRAY$'s expression e are classified into seven disjoint classes:

- CE_1 : Constant expression
- CE_2 : Cell access with probing point x
- CE_3 : Simple expression on cell at probing point x
- CE_4 : Cell access with cluster preserving index expression
- CE_5 : Access to small neighbourhood of probing point x
- CE_6 : Simple expression on two cells at probing point x
- CE_7 : General expression

Those types of operations help to formally classify all probable expressions provided for cell expression and to exploit that knowledge for finer optimization.

Similarly, all the operations for $COND$'s operation op are considered to be from one of the following class:

- CA_1 : Cell access with probing point x
- CA_2 : Simple expression on cell with probing point x
- CA_3 : General expression

Generally speaking, classes CE_1 and CA_7 incorporate expressions to which an optimizer cannot apply any modifications to optimize the $MARRAY$ performance.

2.1.4 Derived operations

When core operation are defined, multiple additional ones are built on top the low-level core operations to have a convenient notation for frequently used typical operations. Such operations are called *derived* operations and there are several types of them: geometric, induced, binary induced, aggregate induced.

Geometric operations

These operations are some special cases of application of $MARRAY$ constructor. Let A be an MDD of type $[[T, D]]$, K be a result of application of $trim$ with some parameters to multidimensional interval D, i be a dimension number, v some valid point along i-th dimension of domain D. Then the following operations can be defined:

- $trimming_K(A) = MARRAY(K, x, A[x])$
- $slice_{(i,v)}(A) = MARRAY(slice(sdom(A), i, v), x, A[x])$

All $MARRAY$ cell expressions in geometric derived operations belong to class CE_2.

Induced operations

These operations are again some special cases of application of $MARRAY$ constructor. Let A be an MDD of type $[[T, D]]$, B be an MDD of type $[[T', D]]$, op be a function $T \to D$, $binOp$ be a function $T \times T' \to T''$. Then the following operations are defined:

- Unary induced operation $\overline{op}(A) : [[T, D]] \to [[T', D]]$ which is the replacement to $MARRAY(D, x, op(A[x]))$
- Binary induced operation: $\overline{binOp}(A, B) : [[T, D]] \times [[T', D]] \to [[T'', D]]$ which is equivalent to $MARRAY(D, x, binOp(A[x], B[x]))$
- Left induced operation: $\overline{op}_{left}(A, const) : [[T, D]] \times T' \to [[T'', D]]$ which is a shorter notation for $MARRAY(D, x, binOp(A[x], const))$
- Right induced operation: $\overline{op}_{right}(const, A) : T' \times [[T, D]] \to [[T'', D]]$ which is equivalent notation for $MARRAY(D, x, binOp(const, A[x]))$
- Note that in case of unary induced operations $MARRAY$ cell expression belongs to CE_3 class and in case of binary induced operations to class CE_6.

Aggregate induced operations

The main convenient operation is $reduce$ operation on which other derived aggregates are based. Let A be an MDD of type $[[T, D]]$, op be an associative and commutative binary operation defined and closed on T. Then $reduce$ operation can be defined as follows: $reduce(op, D, A)$ = $COND(op, D, x, A[x])$. Then several convenient operations can be defined as follows:

- $sum\_cells(A) = reduce(+, sdom(A), x, A[x])$
- $mult\_cells(A) = reduce(*, sdom(A), x, A[x])$

- $avg\_cells(A) = \dfrac{sum\_cells(A)}{card(sdom(A))}$
- $min\_cells(a) = reduce(\min, sdom(A), x, A[x])$
- $max\_cells(a) = reduce(\max, sdom(A), x, A[x])$

2.1.5 Extended relational model

In order to examine MDD specific optimization techniques in combination with set based query processing, the MDD Model is integrated into an adapted Relational Model. The attribute domain of multi-dimensional values can be specified differently:

- Base type is fixed. Spatial is domain unknown
- Base type is fixed. Spatial domain's dimensionality is fixed
- Base type is fixed. Spatial domain is fixed
- Base type is fixed. Spatial domain is fixed with its physical representation (violation of 'hidden physical representation' principle)

The type of attribute domain sets amount of restrictions on MDD values that can be effectively used by an optimizer.

Formally speaking, let $D_i \in \delta$ be spatial domains, $T_i \in \tau$ be types and d_i be positive integers. Let A_i be attributes with $dom(A_i) \in \{[[T_i]], [[T_i, d_i]], [[T_i, D_i]], T_i\}$. Then the mutidimensional relation R is defined as $R \subseteq dom(A_1) \times dom(A_2) \times ... \times dom(A_n)$. When attribute domains are formalized in the model, three relational operations are defined. Let R and S be relations, $cond$ be a predicate on R, op_j be a function defined on R and returning either scalar or multidimensional values for each j. Then relational operations are defied as follows:

- Selection Application σ

  $\sigma_{cond}(R) = \{t \in R \mid cond(t)\}$
- Cross product $\times$

  $\times(R, S) = \{t \mid t = (r_1, r_2, ..., r_n, s_1, s_2, ..., s_n), (r_1, ..., r_n) \in R, (s_1, ..., s_n) \in S\}$
- Application α

  $\alpha_{op_1,...,op_s}(R) = \{t \mid t = (op_1(y), ..., op_s(y)), y \in R\}$

The first two operations are very similar to those in canonical relational model, however, the third operator differs significantly. What it does is application of the provided operators to all of the tuples Relational projection can be expressed through projection operation with special functions returning an element of a tuple at the specific position.

2.1.6 Formalizing Array Query Processing

An array query can be represented as a special graph. Each node of the graph is represented by an operator which comprises the query. This graph is a tree and consists of set trees and element trees. Set trees include relational operations as inner nodes and MDD relations(see definition above) as leaves, whereas element trees' inner nodes are MDD/logical operations. Leaves are MDD constants/iterators. There are also some specific kinds of trees for naming convenience: *condition trees* and *operation trees*. The former are element trees representing boolean multidimensional expressions attached to a select node. The latter are element trees representing some multidimensional expression attached to an application node.

The mere query graph represents data flow between operator nodes. A single edge transfers only particular type of data, thus all edges can be classified into distinct categories depending on type of data flowing through it. The whole edge set is divided into non-intersection sets of three types: relational, dimensional, scalar types. Data edges carrying relations comprises relation sets, edges carrying raster data - dimensional ones, and edges carrying non-dimensional or scalar values are those forming scalar sets. According to this classification, the graph is partitioned in subgraphs of maximal size, each of which contains only one sort of edges. Such subgraphs are called *areas* and are in particular called *relational data areas (RDAs)*, *dimensional data areas (DDAs)* and *scalar data areas (SDAs)*, depending on the type of edges they contain. Optimizing data flow in DDAs is of primary importance in extended relational model for array query processing.

In general, when a query optimizer receives a query it processes it in three stages:

- Rewriting
- Transformation
- Execution

During *rewriting* a query is represented in a *normal query form* which is based on logical optimization rules and the following key principles:

- Eager constant subexpressions evaluation.

This saves computational resources as those expressions might be needed to be computed for each cell of MDD objects

- Boolean expressions normalization aimed at application of optimization rules.

All boolean expressions are transformed to CNF or DNF depending on the predicates in order to let the optimizer detect patterns for application of logical optimization rules

- Prepare induction expressions for the application of optimization rules.

Such a modification leverages associativity and distributivity of induced operations, which can replace MDD operations by scalar ones, dramatically reducing computation cost of the expression

In [14] more than one hundred logical optimization rules are presented. Those rules are mainly based on *load optimization* for geometric operations; exploitation of operations' beneficial properties (such as associativity, distributivity) for induced, binary induced and aggregate operations; movement of individual α, σ subexpressions through cross product operation for set trees.

In the first case, geometric operations are pushed down to multidimensional nodes serving as data sources for upper nodes. This potentially reduces amount of I/O needed to process a single MDD value by an upper node. An example of such a rule is $trimming_D(unOpInduced(A)) \rightarrow unOpInduced(trimming_D(A))$

where $unOpInduced$ is an unary induced operation, A is an MDD value.

In the second case, rewriting expressions leveraging beneficial properties of an operation on which some induced, binary induced or aggregation operation is based may reduce the amount of multidimensional operations in an expression. A remarkable example of such a rule is

$$binOp(binOpInduced(A,b),c) \rightarrow binOpInduced(A,binOp(b,c))$$

In the third case, computation effort is potentially diminished by reducing the amount of multidimensional operations performed. For example, the amount of multidimensional predicate evaluation may be diminished as in case of application of the rule

$$\sigma_{condR and\ condS}(R \times S) \rightarrow \sigma_{condR}(R) \times \sigma_{condS}(S)$$

where R, S are relations, $condR, condS$ - predicates defined on R and S respectively.

In general, query rewriting is reported to be notably profitable if following heuristics are taken into account:

- Perform geometric operations eagerly (load optimization rewriting)
- Reduce number and overall cardinality of Dimensional Data Areas as much as possible

Abiding by this rule lets to diminish the number of edges in DDAs by applying rules that eliminate MDD expressions or transform them into scalar ones.

- Perform applications eagerly

Similar to pushing down projections while using greedy algorithms in relational query processing [16], which is not always the best choice [17]. However, lack of join conditions lets sieving down application into cross product to reduce amount of

tuples for which given functions are applied. Application is an expensive operation as MDD values typically have plenty of cells to operate on.

- Perform selections eagerly

This heuristics also resembles the common heuristics in relational query processing as noted in [18]. Selections are pushed down to diminish operation sets as early as possible. Scalar predicates are given priority over MDD ones.

- Search for common subexpressions

Common subexpressions are stored as intermediate results. Doing one unit of work several times is useless and even costly when operating on large MDD values.

During *transformation* logical plan operations are mapped to physical plan's operations. Such a mapping is not the distinctive feature of RasDaMan, for example, AML (described in Sec. 2.2) also maps logical operations to physical ones. Typically multiple physical plans are valid and semantically equivalent. Those might be analytically compared via usage of special array cost models which were devised for corresponding algebras in [19], [14]. However, being able to just compare physical plans using cost functions is not always sufficient, it is crucial that some physical plan refinement techniques are exploited whose aim is to try to reduce the cost of a physical plan by accounting physical layout of the processed MDD values and to adjust the iteration order correspondingly. In RasDaMan system each type operation is considered separately.

Transformation of induced and aggregate operations is pretty straightforward. Main idea of transformation in such cases is to provide parallel tile processing. However, transformation of binary induced operations is a bit more tricky and we will focus on them in more detail further.

Binary induced operations are optimized by trying to find the optimal tile traversing order over tiles comprising binary induced operands. Finding optimal tile traversing order minimizes disk reads as the main bottleneck during MDD values processing, leveraging efficient exploitation of main memory. The problem can be formalized using graph terminology. Let $G = (V, E)$ be a graph, where $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \varnothing$, so that V_1, V_2 represent tile sets forming the first and the second operands of the binary induced operation respectively. The edge $(v_1, v_2) \in E$ if and only if tiles corresponding to v_1, v_2 need to be processed simultaneously during binary induced operation performance. The result graph is a bipartite graph, as any edge from E has the one end in V_1 and another in V_2. An edge in the graph may intuitively be perceived as an indicator of represents a need for holding two tiles from different sets in main memory for faster processing. However, to process the tiles in main memory those tiles should be, obviously, loaded into the main memory first, unless they are already in place. Loading of a tile is an expensive operation as it is directly related to expansive disk I/O, hence the amount loads should be diminished as much as possible. If there is no cache and only two tiles can be held in main memory simultaneously at a time then the

problem of minimization of disk access can be formulated as follows: find a vertex traversing order $k_1, k_2, \mathrm{K}\ k_n$, minimizing amount of disk access.

The tile traverse algorithm for binary induced operations has been pondered in [14]. Disk access minimization problem is reduced to the problem of minimization of a special *cost* function that is defined as $cost(k_1, k_2, ..., k_i) = cost(k_1, k_2, ..k_{i-1}) + (isAdj(k_i, k_{i-1})\ and\ k_i > 1)\ ?\ 1 : 2$.

The minimum is obtained when the sequence $k_1, k_2, \mathrm{K}\ , k_n$ forms a Hamilton cycle. In generic case determination of whether such a sequence of vertices exist is known to be an NP-complete problem [20].

However, the restriction to visit each vertex only once can be relaxed. Such an approach has been used in [21] for facing the problem of finding the optimal tile traversing order for array join - a special case of binary induces operation. In this paper the requirement of exclusive visit of each vertex is replaced with requirement of visiting each edge only once with intention to minimize multiple reads of a tile. Authors exploit Hierhozer and Weiner necessary and sufficient condition to find an Euler circuit, i.e. a circuit that visits each edge in the graph only once. As the Hierholzer and Wiener criteria claims [22] the presence of Euler circuit in a graph is equivalent to its connectivity and all vertices having even degrees. Authors split the graph G into connected components, augmenting each with extra auxiliary edges so that each vertex has an odd degree and Euler circuit exists. For each component an Euler circuit is computed and tile traverse path is determined. The result traverse path is built from a random permutation of components' paths. The authors report that it is still an open question how this approach may be adapted to either a distributed environment where each tile load may have its own cost and or to a presence of an arbitrary sized cache.

2.2 AML algebra

2.2.1 Array abstraction

In *Array Manipulation Language (AML)* [23] an array A is set by its shape S, domain D (which is conceptually similar to Array's algebra value set), and the mapping M that establishes the link between the array's shape and domain. $S[i]$ represents the extent of A along i-th dimension, as all cells of an AML array has lower bound equal to zero. A vector x is said to be in array's shape S iff $\forall i 0 \le x[i] < S[i]$. Said that, we can define the mapping M more formally as a function that returns some value from D for every $x \in S$ and a special $null \notin D$ element otherwise.

2.2.2 Algebra operations

AML makes use of *bit patterns* and several probe functions defined on those patterns (*index* and *count*). The $index(P,k)$ function returns the index of the $k-1$-th 1 in the bit pattern P if it does not equal to (0), otherwise the function returns 0. The $count(P,k)$ function will return number of 1 up to k-th position in the bit pattern B.

Let A,B be the multidimensional arrays with different shapes S_1, S_2 and common domain D with d dimensions, P, P_i be some bit patterns, D_f, R_f be some shapes called *domain* and *range* boxes respectively; then the core operations on arrays in AML algebra are:

- Subsection. $C = SUB_i(A,P)$

The SUB_I operator iterates over hyperplanes along i-th dimension and, if allowed by the corresponding bit in P, concatenates the hyperplane to the result array.

- Merging $C = MERGE_i(A,B,P,\delta)$

The $MERGE$ operator cyclically glues together hyperplanes cut along i-th dimension according to pattern P. Hyperplane are taken from A for set bit in the pattern and from B otherwise. If source array has no values hyperplane is filled with δ value.

- Function application. $APPLY(A, f, D_f, R_f, P_0, \mathrm{K}, P_{d-1})$

Iterates over slabs of shape D_f from A, checks whether that slab is "allowed" by all of the P_i patterns (by looking through corresponding bits over all patterns and rejecting the slab if some of the bits is not set). If the slab is "allowed" function f is applied to the slab and a slab of shape R_f is obtained. The result slab is "glued" to the result array from the side indicated by iteration position (more formally to ($count(P_0,i_1),\mathrm{K},count(P_{d-1},i_{d-1})$ where vector (i_1,K,i_n) represents the iteration vector)).

The $APPLY$ operator might seem to resemble the $MARRAY$ operator defined in Baumann's array algebra. Indeed, $MARRAY$ operator can be called as a more restrictive version of $MARRAY$ operator with range boxes equalling a single array cell. In AML function f is a black box just as an expression of class CE_7 for $MARRAY$ operator.

2.2.3 Optimization

AML optimization techniques, similarly to those in Baumann's algebra, can be classified into logical and physical ones. In AML those techiques are applied in three phases, so called *rewriting phase* and *plan generation phase*, *plan refinement phase*.

During *rewriting phase* an AML optimizer receives an AML query, builds query expression tree, and transforms it to semantically equivalent one using algebra's transformation rules. The transformed expression is guaranteed to be evaluated to the same result as the intial one, however the transformed query is hoped to be executed faster for some reason (e.g. due to operating on smaller amount of cells as in case of *load optimization* in Baumann's array algebra, see Sec. 2.1.6). In [19], [23] several algebraic rules are presented. The set of logical optimization rules is not so diverse compared to those devised in [14]. The approaches used by AML optimizer is similar to those exploited by Baumann's Algebra optimzer for *load optimization*. Examples of AML optimizer's query transformations are: merging several subsample operators into one, pushing subsample through merge in some cases, pushing subsamples through apply in some cases, etc. As AML's algebra operators by nature are superior to Baumann's Algebra ones in terms of flexibility (e.g. bit pattern support in all elementary operators, differently sized range boxes for function application operator), logical optimization rules become more complex, accompanied with extra initial conditions and overburdened by auxiliary bit pattern calculations. For example, some generic optimizations exploited by a Baumann's algebra optimizer, like reducing the cardinality of processed cells set before function application, cannot be simply borrowed by AML optimizer for use. This occurs due to the fact that additional logic is introduced by bit patterns and possibly different sizes of range and domain boxes. However, overall relative complexity of AML's logical transformation rules does not diminish the AML optimization potential. At *plan generation phase* rewritten expression tree is mapped to a physical plan just as during *Transformation* phase of Baumann's optimizer. The physical plan is represented by a directed graph where a vertex represents a logical operator, while an edge depicts a data flow. Every operator expects a stream of non-overlapping chunks (*tiles* in RasDaMan terminology) of some particular shape and in some particular order as an input. Operators produce non-overlapping array chunks of particular shape and in some particular order as an output. An operator may have some parameters which specifies its behavior. The summary on the physical operators is provided in table 1.

Table 1. AML physical operations

Name	Input Stream	Parameters	Description
APPLY_P	1	function	Applies a function to an array
LEAF_P	0	array	Reads array from disk
COMBINE_P	> 0	combination	Maps input slabs (in each

		map	dimension) to output slabs. Used for *MERGE* and *SUB*
REGROUP_P	1	-	changes input chunks' shapes
REORDER_P	1	-	permutes input chunks

The building of the physical operator tree is based on recursive top-down traversing of the expression tree. The algorithm step can be determined depending on the currently processed node of the expression tree. Algorithm of building the physical plan tree is summarized in tab. 2.

Table 2. AML physical plan tree building algorithm

Current tree root node is ...	Action
Nonleaf *APPLY* with domain box D_f and range box R_f	Add **APPLY_P** and **REGROUP_P** operators where * **REGROUP_P** precedes **APPLY_P** * **APPLY_P** input chunk shape matches D_f ; output – R_f * Application mask are taken from *APPLY* patterns
SUB or *MERGE* with n leaves	Find max tree of *SUB* and *MERGE* rooted at current node. Translate it to n-ary **COMBINE_P** and n **REGROUP_P** operators
Leaf *APPLY*	Translate it to **LEAF_P**

The main aims of the optimizer on *plan refinement phase* are to remove no-op operators and specify chunk ordering of each operator. Chunk iteration order directly affects the amount of data buffed by an operator, therefore the optimizer tries to minimize the memory requirement so that try to execute entirely in memory not to spent effort on materializing intermediate results of evaluation. For d-dimensional array consisting of q chunks there are $q!$ iteration orders. If a plan consists of multiple operators consuming several arrays then considering all iteration orders becomes exponentially expensive. AML authors decided that the optimizer should consider only $\overline{d}$ iteration orders for each operator, where $\overline{d}$ is the maximum dimensionality of an array consumed in the plan. Each of those $\overline{d}$ iteration order differs in the primary dimension by which all the input chunks are ordered. Other sort dimensions are taken in dimension number increasing order. Authors claim that *Hilbert* curve [24] or Z-order [25] could be considered by an optimizer as those orders might be related to secondary storage scheme, but those types of orders are neglected for the sake of simplicity.

For physical plan consisting of w operators there will be $\overline{d}^{w}$ iteration orders. There is another problem that even if an optimal iteration order is found for some operator it does not mean that optimizer is done with this operator. There might be another dependent operator expecting output chunk stream of the just considered operator in a completely different order. However, instead of examination of another possible chunk ordering it might be more beneficial to insert a reorder operator between the producer and the consumer operators. Having addressed the aforementioned problems, AML authors devised a cost-based algorithm reducing the complexity of assigning chunk iteration orders to operators down to $O(w\overline{d}^{2})$. The algorithm is briefly discussed below.

Let $C_i(x)$ be cost of choosing output to be returned in i-th order for operator x. Let $Y(x)$ be set of x neighbors. If the x is **LEAF_P** then cost function is equal to operator memory cost (defined for each operator separately), otherwise: $C_i(x) = c_i(x) + \sum_{y \in Y(x)} \left(\min\left(C_i(y), \min_{i \neq j} \left(C_j(y) + reorderCost(j,i,x,y)\right)\right)\right)$

Here $reorderCost(j,i,x,y)$ shows the additional cost augmented after inserting a **REORDER_P** operator between x and y operator reordering j-th ordered chunk stream into i-th order one.

2.3 SciDB

To the best of our knowledge the formal SciDB algebra description with possible derived optimization techniques has not been yet published. This may be explained by the fact that, formally speaking, there is no formal algebra in SciDB, as it will be seen below, and all operators are initially considered to be user defined functions (*UDF*s). There are some built-in operators, but they do not form any algebra, and are still considered UDFs, as SciDB pays much attention to extensibility. Despite this fact, here we try to formalize our knowledge about SciDB and structure it using the plan used for AML and Baumann's array algebra. First we define an array, then list built-in elementary operations of SciDB and finally try to explore how SciDB query optimizer works.

2.3.1 Array abstraction

SciDB operates on collection of n-dimensional arrays each of which again may be represented as a mapping

$$A: I_1 \times I_2 \times \mathrm{K} \times I_n \rightarrow (V_1 \times V_2 \times \mathrm{K} \times V_k) \cup \{NULL\}$$

Sets I_1, I_2, , I_n are closed subsets of Z^n just as in case of Baumann's array algebra. Value associated with each index vector (i_1, K, i_n) represents an array's *cell*. What needs more attention here is the nature of a SciDB array's cell. As it can

be seen from formal SciDB array definition the cell value is actually a heterogeneous *tuple* or a special *NULL* value. The presence of *NULL* value gives lets SciDB to store *sparse* arrays just out of the box. Cells which are mapped to *NULL* are called *empty*. Each tuple element can be addressed by so called *attribute* name. It should be mentioned, there are some constraints on value sets V_j, which restrict a tuple value to be of one of some predefined types: fixed length string, number, etc. However, users of SciDB are given an opportunity to compose custom types (user defined types – *UDT*s). One of the features of the SciDB array data model is that it is *nested*, allowing an array cell contain another array.

2.3.2 Algebra operators

One of the most distinguishable feature of SciDB is that as reported in [26] it has no built-in operators, forming some rigor algebraic system. Authors claim that all operators are in fact UDFs and SciDB has some embedded UDFs, which to some extent may be perceived as elementary operators forming SciDB base algebra. However, this might seem as contradiction to the SciDB ideology. In [27] the term 'algebra' is used, but formalism is avoided and just built-in operator usage examples are provided. Below we describe SciDB built-in operators mentioned in [27] and specified in online SciDB documentation [28].

Let

- A be an n-dimensional SciDB array, B be m-dimensional array
- V be a *array index values* for A,
  where *array index values* can be defined as a set of pairs $\{(i_1, v_1), \mathrm{K}, (i_p, v_p)\}$ where for every $0 \le j < n$: $0 \le i_j < n$, i_j values are distinct and v_j is an admissible index value for array's dimension j.
- Q be a predicate containing free occurrence of A cell's dimensional index
- E be a predicate containing free occurrences of A cell's attributes
- F be a function mapping A cell attributes values to some admissible cell type

Then the set of the following operators may be described:

- $Slice(A,V)$

Gets out the subarray of dimensionality $dim(A) - |V|$ from array A, taking A as buffer array and sequentially cutting out hyperplanes from buffer array based on elements in V.

- $Subsample(A,Q)$

Gets out the subarray from array A building it from those cells whose index over dimension j satisfies the predicate Q for every admissible j.

- $SJoin(A,B)$

Combines the cells' values of two input arrays if cells have the identical dimensional indices.

- $Filter(A,E)$

Gets an array of the same dimensionality as A where each cell is taken from A *iff* E evaluates to true on it, or considered *empty* otherwise.

- $Apply(A,F)$

Gets a new array applying F to a cell (substituting cell's corresponding attribute values to F) and storing the result of the calculation in the result array's cell with the same dimension index.

It should be mentioned that SciDB orients on high extensibility, which explains the shift of SciDB towards UDFs (and UDTs). ScidDB provides a special facility that enables to extend the aforementioned set of operators with custom ones, written in C++. Custom operators are required to take array input(s) and return array output(s). Moreover, SciDB supports defining own aggregates (user defined aggregates – *UDA*s), increasing the degree of extensibility even more.

2.3.3 Optimization

The shift to the paradigm 'everything is defined by a user' makes query planning a much harder task. The SciDB optimizer operates on 'blackbox' operators which may theoretically be optimized, but in general case their nature is too generic for the optimizer to determine optimizations it might perform. However, compared to AML and RasDaMan systems, SciDB tries to overcome this difficulty with optimizations basing more on physical data storage and parallelism. Those optimizations are discussed below.

When a query is got an optimizer builds a logical plan ding all the required semantic checks. As it is stated in [27] the optimizer will produce a complete physical plan corresponding to the built logical plan, where possible. Otherwise it will split the query plan into subplans consisting of pipelinable operators and execute them (in parallel, taking into account the physical structure, discussed further). One logical optimization of SciDB query optimizer mentioned in [29] is detecting commuting operations and pushing them down in the query tree. However, due to generic nature UDFs finding such operators is typically a luck.

When physical information comes in use, SciDB optimizer starts to 'breathe easier'. A SciDB instance is supposed to run on multiple nodes, adhering to *shared nothing*

design. A central *system catalog* exists, storing meta information about user-defined extentions, data distribution, etc. By such SciDB enables to provide a high level of parallelizm and related performance improvements accounting the fact that array data manipulations are known to be CPU bound([14], [29]). SciDB optimizer makes use of distributed architecture and performs several related optimizations discussed in [29]. For example, the optimizer examines the logical tree for blocking operations, i.e. those which require a temporary array to be constructed (e.g. operations demanding redistribution of data in order to execute). In [29] built-in SciDB optimizer is reported to be an incremental and cost-based one. It means that the optimizer picks the best choice for the first subtree to execute making use of a cost model for plan evaluation. The same paper states the SciDB optimizer can be called a 'simple optimizer' which tries to minimize amount of date movement and increase the level of parallelism.

However, different optimization techniques has been recently proposed, which might be used in SciDB environment. Those include devising *iterative array processing* model for a parallel array engine proposed in [30], optimization of SciDB's *Filter* operator proposed in [31], shuffle join optimization framework for the SciDB array data model presented in [32], etc.

3. Possible directions of investigation

Based on the overview of the optimization process provided above we summarize some directions that are available for further investigation. Under no circumstances should this list be perceived as complete one. The list below is just a set of noticed future work directions which is actually much larger.

- **Baumann's Array Algebra. Array Query Processing. Commutativity of slice and trimming is not accounted during optimization**

How can this property be exploited for load optimization?

- **Baumann's Array Algebra. Array Cost Model. Approximating selectivity of predicates containing MDD expressions using common techniques for AQP approach**.

In relational query processing there are three well-known techniques: Sampling, Parametric, Histogram-based Techniques. RasDaMan creators opted for Histogram-based approach, saying that parametric techniques have a problem that real distributions (especially those of operations results) are seldom accurately approximated by mathematical distributions. The problem is very serious in case of raster image data. Sampling techniques are reports to be very flexible and tolerant to updates. The main disadvantage of such an approach is that sampling has have considerable I/O and CPU overhead and lacks computation result reusability. How serious is that overhead and how disadvantages overweigh advantages?

- **Baumann's Array Algebra. Cell expressions analyzing**

It is possible to analyze cell expressions for the elementary operations (*MARRAY* , *COND*) in order to optimize disk access, e.g. detect tiles which should be cached iteration over MDD value.

- **Baumann's Array Algebra. Optimization of binary induced operations. Array join problem. Optimization for relaxed restrictions**

When the tile graph is built as in [21] the cost of fetching a tile from disk is considered the same for all tiles. However, this might be not the case for a distributed environment or in presence of cache with size allowing to hold more than 1 tile of each operand. The approach presented in [33] might be considered.

- **AML. Plan refinement. Iteration order**

Authors claim that *Hilbert* curve or Z -order could be accounted by an optimizer during plan refinement as those orders might be related to storage scheme, but those types of orders are dismissed from consideration for the sake of simplicity. Can such a simplification be revisited?

4. Conclusion

In the current paper we have investigated the theoretical background of array databases exploring three different mature array database management systems: RasDaMan, AML, SciDB. We have looked at those database from a fixed perspective: firstly, we explore the data model the system uses to simply define an *array*; secondly, we examine what formal algebra the system constructs above arrays; thirdly, we take a closer look on algebraic optimizations (logical optimizations) and those applied when information about physical storage and retrieval of data is taken into account (physical level optimizations). We collect some possible directions of further investigation for considered array databases. The list of directions mainly contain ones outlined by the authors of the array databases themselves.

References

[1]. Peter Baumann. Raster Data Management and Multi-Dimensional Arrays, pages 2332-2339. Springer US, Boston, MA, 2009.
[2]. Hubble telescope images. https://www.spacetelescope.org/images/.
[3]. Large hadron collider storage. http://lhcb-public.web.cern.ch/lhcb-public/en/Data
[4]. Gilberto Câmara, Lúbia Vinhas, Karine Reis Ferreira, Gilberto Ribeiro De Queiroz, Ricardo Cartaxo Modesto De Souza, Antônio Miguel Vieira Monteiro, Marcelo Tílio De Carvalho, Marco Antonio Casanova, and Ubirajara Moura De Freitas. TerraLib: An Open Source GIS Library for Large-Scale Environmental and Socio-Economic Applications, pages 247-270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
[5]. PostGIS official web page. https://postgis.net/.
[6]. SpatialLite web page. https://www.gaia-gis.it/fossil/libspatialite/home.
[7]. Oracle GeoRaster documentation. https://docs.oracle.com/cd/B19306_01/appdev.102/b14254/geor_intro.htm.

[8]. Baumann P. and Holsten S. A comparative analysis of array models for databases. In Database Theory and Application, Bio-Science and Bio-Technology. Communications in Computer and Information Science, volume 258, 2011.
[9]. Rasdaman home page. http://www.rasdaman.org/.
[10]. Paul G. Brown. Overview of scidb: large scale array storage, processing and analysis. In SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 963-968, 2010.
[11]. Paulo Jorge Pimenta Marques. Arbitrary tiling of multidimensional discrete data cubes in the rasdaman system. 1998.
[12]. L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. Information Systems, 20(2):155-183, 1995. Scientific Databases.
[13]. Peter Baumann. A database array algebra for spatio-temporal data and beyond. 06 1999.
[14]. Roland Ritsch. Optimization and evaluation of array queries in database management systems. 12 1999.
[15]. A. G. Gutierrez and P. Baumann. Modeling fundamental geo-raster operations with array algebra. In Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007), pages 607-612, Oct 2007.
[16]. Frank P. Palermo. A data base search problem. 01 1974.
[17]. Joseph M. Hellerstein. Optimization techniques for queries with expensive methods. ACM Trans. Database Syst., 23(2):113-157, June 1998.
[18]. A. Swami. Optimization of large join queries: Combining heuristics and combinatorial techniques. SIGMOD Rec., 18(2):367-376, June 1989.
[19]. Arunprasad P. Marathe and Kenneth Salem. Query processing techniques for arrays. SIGMOD Rec., 28(2):323-334, June 1999.
[20]. Hamiltonian cycle. http://mathworld.wolfram.com/HamiltonianCycle.html.
[21]. P. Baumann and V. Merticariu. On the efficient evaluation of array joins. In 2015 IEEE International Conference on Big Data (Big Data), pages 2046-2055, Oct 2015.
[22]. Ethan Kim. Comp 251: Data structures and algorithms. https://ethkim.github.io/TA/251/eulerian.pdf.
[23]. Arunprasad P. Marathe and Kenneth Salem. A language for manipulating arrays. In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, pages 46-55, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
[24]. Hilbert curve. http://www4.ncsu.edu/ njrose/pdfFiles/HilbertCurve.pdf.
[25]. Z-curve general information. http://wiki.gis.com/wiki/index.php/Z-order_(curve).
[26]. P. Cudre-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. DeWitt, B. Heath, D. Maier, S. Madden, J. Patel, M. Stonebraker, and S. Zdonik. A demonstration of scidb: A science-oriented dbms. Proc. VLDB Endow., 2(2):1534-1537, August 2009.
[27]. Paul G. Brown. Overview of scidb: Large scale array storage, processing and analysis. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, pages 963-968, New York, NY, USA, 2010. ACM.
[28]. Scidb doucumentation. https://paradigm4.atlassian.net/wiki/spaces/ESD/overview.
[29]. Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The architecture of scidb. In Proceedings of the 23rd International Conference on Scientific and Statistical Database Management, SSDBM'11, pages 1-16, Berlin, Heidelberg, 2011. Springer-Verlag.

[30]. Emad Soroush, Magdalena Balazinska, Simon Krughoff, and Andrew Connolly. Efficient iterative processing in the scidb parallel array engine. In Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM '15, pages 39:1-39:6, New York, NY, USA, 2015. ACM.

[31]. Sangchul Kim, Seoung Gook Sohn, Taehoon Kim, Jinseon Yu, Bogyeong Kim, and Bongki Moon. Selective scan for filter operator of scidb. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM '16, pages 28:1-28:4, New York, NY, USA, 2016. ACM.

[32]. Jennie Duggan, Olga Papaemmanouil, Leilani Battle, and Michael Stonebraker. Skew-aware join optimization for array databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, pages 123-135, New York, NY, USA, 2015. ACM.

[33]. Weijie Zhao, Florin Rusu, Bin Dong, and Kesheng Wu. Similarity join over array data. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, pages 2007-2022, New York, NY, USA, 2016. ACM.

Базы данных для обработки массивов: взгляд изнутри

В.А. Павлов <vlad.pavlov24@gmail.com>
Б.А. Новиков < b.novikov@spbu.ru >
Санкт-Петербургский государственный университет,
199034, Россия, Санкт-Петербург, Университетская набережная, д. 13б

Аннотация. После появления огромного количества научных данных, которые необходимо было хранить и обрабатывать, в мире баз данных возникла задача поддержки больших многомерных массивов. Стала необходимой разработка специальных баз данных, которые основывались бы на модели данных, "сердцем" которой было понятие массива (array). Разработка хорошо организованной системы управления базой данных, базирующейся на нетрадиционной модели данных, требовала решения следующих задач: формальное определение модели данных, основывающейся на понятии массива; построение формальной алгебры, работающей с объектами модели; разработка правил оптимизации запросов на логическом уровне, а затем и на физическом. Эти задачи уже решались создателями специальных баз данных, настроенных на обработку и хранение массивов (array databases). В данной работе рассматриваются понятия массива, формальные алгебры и методы оптимизации запросов в таких развитых базах данных, как RasDaMan, AML, SciDB – базах данных, ориентированных на хранение и обработку крупных многомерных массивов.

Ключевые слова: базы данных для обработки массивов; обзор; формальная алгебра баз данных для обработки массивов; обработка запросов к базам данных для обработки массивов; оптимизация запросов к базам данных для обработки массивов; AML; RasDaMan; SciDB

DOI: 10.15514/ISPRAS-2018-30(1)-10

Для цитирования: Павлов В.А., Новиков Б.А. Базы данных для обработки массивов: взгляд изнутри. Труды ИСП РАН, том 30, вып. 1, 2018 г., стр. 137-160. DOI: 10.15514/ISPRAS-2018-30(1)-10

Список литературы

[1]. Peter Baumann. Raster Data Management and Multi-Dimensional Arrays, pages 2332-2339. Springer US, Boston, MA, 2009.

[2]. Hubble telescope images. https://www.spacetelescope.org/images/.

[3]. Large hadron collider storage. http://lhcb-public.web.cern.ch/lhcb-public/en/Data

[4]. Gilberto Câmara, Lúbia Vinhas, Karine Reis Ferreira, Gilberto Ribeiro De Queiroz, Ricardo Cartaxo Modesto De Souza, Antônio Miguel Vieira Monteiro, Marcelo Tílio De Carvalho, Marco Antonio Casanova, and Ubirajara Moura De Freitas. TerraLib: An Open Source GIS Library for Large-Scale Environmental and Socio-Economic Applications, pages 247-270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[5]. PostGIS official web page. https://postgis.net/.

[6]. SpatialLite web page. https://www.gaia-gis.it/fossil/libspatialite/home.

[7]. Oracle GeoRaster documentation. https://docs.oracle.com/cd/B19306_01/appdev.102/b14254/geor_intro.htm.

[8]. Baumann P. and Holsten S. A comparative analysis of array models for databases. In Database Theory and Application, Bio-Science and Bio-Technology. Communications in Computer and Information Science, volume 258, 2011.

[9]. Rasdaman home page. http://www.rasdaman.org/.

[10]. Paul G. Brown. Overview of scidb: large scale array storage, processing and analysis. In SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pages 963-968, 2010.

[11]. Paulo Jorge Pimenta Marques. Arbitrary tiling of multidimensional discrete data cubes in the rasdaman system. 1998.

[12]. L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems. Information Systems, 20(2):155 - 183, 1995. Scientific Databases.

[13]. Peter Baumann. A database array algebra for spatio-temporal data and beyond. 06 1999.

[14]. Roland Ritsch. Optimization and evaluation of array queries in database management systems. 12 1999.

[15]. A. G. Gutierrez and P. Baumann. Modeling fundamental geo-raster operations with array algebra. In Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007), pages 607-612, Oct 2007.

[16]. Frank P. Palermo. A data base search problem. 01 1974.

[17]. Joseph M. Hellerstein. Optimization techniques for queries with expensive methods. ACM Trans. Database Syst., 23(2):113-157, June 1998.

[18]. A. Swami. Optimization of large join queries: Combining heuristics and combinatorial techniques. SIGMOD Rec., 18(2):367-376, June 1989.

[19]. Arunprasad P. Marathe and Kenneth Salem. Query processing techniques for arrays. SIGMOD Rec., 28(2):323-334, June 1999.

[20]. Hamiltonian cycle. http://mathworld.wolfram.com/HamiltonianCycle.html.

[21]. P. Baumann and V. Merticariu. On the efficient evaluation of array joins. In 2015 IEEE International Conference on Big Data (Big Data), pages 2046-2055, Oct 2015.

[22]. Ethan Kim. Comp 251: Data structures and algorithms. https://ethkim.github.io/TA/251/eulerian.pdf.

[23]. Arunprasad P. Marathe and Kenneth Salem. A language for manipulating arrays. In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB '97, pages 46-55, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[24]. Hilbert curve. http://www4.ncsu.edu/ njrose/pdfFiles/HilbertCurve.pdf.
[25]. Z-curve general information. http://wiki.gis.com/wiki/index.php/Z-order_(curve).
[26]. P. Cudre-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. DeWitt, B. Heath, D. Maier, S. Madden, J. Patel, M. Stonebraker, and S. Zdonik. A demonstration of scidb: A science-oriented dbms. Proc. VLDB Endow., 2(2):1534-1537, August 2009.
[27]. Paul G. Brown. Overview of scidb: Large scale array storage, processing and analysis. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, pages 963-968, New York, NY, USA, 2010. ACM.
[28]. Scidb doucumentation. https://paradigm4.atlassian.net/wiki/spaces/ESD/overview.
[29]. Michael Stonebraker, Paul Brown, Alex Poliakov, and Suchi Raman. The architecture of scidb. In Proceedings of the 23rd International Conference on Scientific and Statistical Database Management, SSDBM'11, pages 1-16, Berlin, Heidelberg, 2011. Springer-Verlag.
[30]. Emad Soroush, Magdalena Balazinska, Simon Krughoff, and Andrew Connolly. Efficient iterative processing in the scidb parallel array engine. In Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM '15, pages 39:1-39:6, New York, NY, USA, 2015. ACM.
[31]. Sangchul Kim, Seoung Gook Sohn, Taehoon Kim, Jinseon Yu, Bogyeong Kim, and Bongki Moon. Selective scan for filter operator of scidb. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, SSDBM '16, pages 28:1-28:4, New York, NY, USA, 2016. ACM.
[32]. Jennie Duggan, Olga Papaemmanouil, Leilani Battle, and Michael Stonebraker. Skew-aware join optimization for array databases. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, pages 123-135, New York, NY, USA, 2015. ACM.
[33]. Weijie Zhao, Florin Rusu, Bin Dong, and Kesheng Wu. Similarity join over array data. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, pages 2007-2022, New York, NY, USA, 2016. ACM.