Noname manuscript No.
(will be inserted by the editor)

Do the middle letters of "OLAP" stand for Linear Algebra ("LA")?

Hugo Daniel Macedo · José Nuno Oliveira

Received: date / Accepted: date

Abstract Inspired by pointfree relational data processing, this paper addresses the foundations of an alternative roadmap for parallel online analytical processing (OLAP) based on a separation of concerns: rather than depending on standard database technology and heavy machinery, OLAP operations are performed by encoding data in matrix format and relying thereupon solely on LA operations.

The paper investigates, in particular, how the generation of aggregation operations such as cross tabulations and data cubes in OLAP can be expressed in terms of matrix multiplication, transposition and the Khatri-Rao variant of the Kronecker product.

This offers much potential for parallel OLAP, as such matrix operations have a well-defined parallelization theory. Last but not least, the approach offers a formal semantics for data aggregation which is useful in reasoning about OLAP operation as a whole.

Keywords Foundations of decision support systems · Linear algebra · OLAP · Data cube

1 Introduction

This paper finds its motivation in the need to perform data mining and online analytical processing (OLAP) [5, 25, 13] in an efficient way. These techniques

Hugo Daniel Macedo
MAPi PhD Student
High-Assurance Software Lab (HASLab)
Minho University, Braga, Portugal
E-mail: hmacedo@di.uminho.pt

José Nuno Oliveira
High-Assurance Software Lab (HASLab)
Minho University, Braga, Portugal
E-mail: jno@di.uminho.pt

are very useful for summarizing huge amounts of information in the form of histograms, sub-totals, cross tabulations (vulg. *pivot tables*), roll-up/drill down transformations and data cubes, whereby new trends and relationships hidden in raw data can be found. The need for this kind of operation concerns not only large companies generating huge amounts of data every day (the "big data" trend), which need to be consolidated overnight, but also the laptop spreadsheet user who wants to make sense of the data stored in a particular workbook.

OLAP is resource-demanding and calls for parallelization. With the advent of multi-core personal machines, code parallelization has become a wide concern, ranging from large main-frames to laptops. Placed at this end of the spectrum, even the anonymous Microsoft Excel user might in fact legitimately ask: *is the generation of pivot tables in Excel actually taking advantage of the underlying multi-core hardware? How parallel is such a construction?*

There are essentially two ways to reach parallel OLAP. One is the development of dedicated systems such as eg. PARSIMONY [11], which provides a parallel and scalable infrastructure for multidimensional analysis and data mining targeting at distributed memory parallel machines such as the IBM SP-2, for instance. Still in this trend, existing tools intended for other areas (eg. scientific computing) have been adapted to scalable data cube construction [35].

The other way proceeds by splitting the problem in two steps: first, one finds a mathematical framework exhibiting well-known and developed potential for parallel execution and encodes OLAP operations in such a framework; second, one relies on such parallelization theories and reuses general purpose software already available, possibly taken from another application area. This second approach is beneficial in the sense that there is a *separation of concerns* — one does not need to "think parallel" in the first place, parallelism coming "for free" thanks to the general theory.

A roadmap to parallel OLAP. In this paper we follow the latter strategy, finding inspiration in relatively recent developments in the remote area of digital signal processing (DSP) which, as is well-known, relies on linear algebra (LA). Generation of fast (parallel) code for DSP has witnessed great advances in recent years under the motto *"can we teach computers to write fast libraries?"* [26]. Domain specific languages (DSLs) and systems such as (respectively) SPL and SPIRAL [27], for instance, have shown how automatic generation of high performance libraries for LA applications relies on very high-level *specification scripts* written in index-free matrix algebra, in which matrix multiplication plays a major role, given its amenability to parallelization via *divide-and-conquer* algorithms [32, 29].

Our approach is similar: we will show how to translate OLAP into linear algebra, the benefits being two-fold: not only one is able to reason about OLAP in this way, thanks to the well-known calculus of matrices, but also it (indirectly) provides a mainstream way to achieve parallelism in OLAP.

For easy illustration, the proposed translation is supported by a small set of combinators extending the widespread MATLAB [1] library of matrix operations.

Contribution. The ideas presented in this paper derived from the authors' work on typing linear algebra [16] which eventually drove them into the proposed synergy between linear algebra and OLAP.

Such a synergy is, to the best of their knowledge, novel in the field. Rather than relying on standard OLAP state of the art developments, a cross field perspective is put forward that may open new ways of looking at this body of knowledge.

Remark concerning terminology: the more widespread acronym OLAP is preferred to ROLAP ('relational OLAP") [25] throughout the paper, even knowing that ROLAP is most often intended.

2 Background

Parallelism is intimately related to so-called *divide and conquer* algorithms, or *breakdown-rules* [27] which are naturally adapted for execution in multiprocessor machines. It turns out that the construction of such algorithms is the "natural" way to write programs in the so-called *functional programming* style [2,14]. Thus parallelism blends well with this programming discipline, evidence of this being, for instance, how easily Google's MAPREDUCE is expressed using functional combinators [15,7]. Functional programming has witnessed great advances over the years in many respects, namely in the development of an *algebra of programming* (AoP) [2] which puts emphasis on the "type structure" which is central to modern functional languages such as Haskell, for instance [14].

The authors have shown in a recent paper [16] how close to the AoP a *"matrices as arrows" typed* approach to linear algebra is. This is easy to understand after all since functions are special cases of binary relations which in turn are nothing but Boolean matrices [2]. Elsewhere, it has been shown how to take advantage of binary relation algebra in reasoning about data dependencies in databases [20,22] and data transformation in general [21]. Needless to say, relations play a major role in data processing since Codd's pioneering work on the foundations of the *relational data model* theory [4].

Given this proximity between relation and matrix algebra, the question arises: how much gain can one expect from translating results from one side to the other? In this paper we will show how a particular construction in relation algebra — that of building binary *relational projections*, used in [20,22] to reason about functional dependencies in databases — translates into building cross tabulations (pivot tables) which are central to OLAP and data-mining.

[1] MATLAB [TM] is a trademark of The MathWorks ®.

[2] Indeed, relation algebra and matrix algebra can be regarded as instances of the *allegory* concept [9], the latter under some restrictions on the algebra of matrix elements.

On the relational side, such projections are always of the form

$$f \cdot R \cdot g^{\circ} \tag{1}$$

where R is the binary relation being projected and f and g are *observing* functions, usually associated to attributes. The dot $(\cdot)$ between the symbols denotes relational composition [3] and $(\_)^{\circ}$ expresses the converse operation, whereby pair (b, a) belongs to relation R° iff pair (a, b) belongs to R.

Pattern (1) turns up very often in relation algebra [2]. In its particular use to express data dependencies, such projections take the form

$$f_A \cdot [\![T]\!] \cdot f_B^{\circ} \tag{2}$$

where T is a database file, or table (set of data records, or tuples), A and B are attributes of the schema of T, f_A (resp. f_B) is the function which captures the semantics of attribute A (resp. B) [4], and $[\![T]\!]$ captures the semantics of T in the form of a binary relation known as a *coreflexive* [2]: $[\![T]\!] = \{(t, t) \mid t \in T\}$. However strange and redundant this construction may look like, it proves essential to the reasoning, as shown in [20,22]. Expressed in set-theoretical notation, projection (2) is set-comprehension

$$\{(t[\![A]\!], t[\![B]\!]) \mid t \in T\}$$

where $t[\![A]\!]$ (resp. $t[\![B]\!]$) denotes the value of attribute A (resp. B) in tuple t.

Essential to (2) is its emphasis on the very basic combinators of relation algebra: composition and converse. These generalize to matrix multiplication and transposition, respectively, which are easy to parallelize. The following law of the calculus of (blocked) matrices

$$\left[\, R \middle| S \,\right] \cdot \left[\frac{U}{V}\right] = R \cdot U + S \cdot V \tag{3}$$

— where R, S, U, V are matrix-blocks — is given in [16] to capture the essence of (parallelizable) divide-and-conquer matrix multiplication.

Under this motivation, we will show below that cross tabulations in OLAP can be expressed by a formula similar to (2),

$$t_A \cdot [\![T]\!]_M \cdot t_B^{\circ} \tag{4}$$

where M is a *measure* and A and B are the *dimensions* chosen for the particular cross tabulation to build. Notation t_A (resp. t_B) expresses the *membership* matrix of the column addressed by dimension A (resp. B) whose construction will be explained later. Also explained later, $[\![T]\!]_M$ means the diagonal matrix capturing column M of T.

The construction of matrices t_A, t_B and $[\![T]\!]_M$ will be first illustrated with examples. Cross tabulations will be pictured as displayed by Microsoft Excel.

[3] Recall from discrete maths that, given two relations R and S, pair (c, a) will be in the composition $R \cdot S$ iff there is some b such that (c, b) is in R and (b, a) is in S.

[4] That is, given a tuple $t \in T$, $f_A(t)$ yields the value of attribute A in t, usually denoted by $t[A]$ (similarly for attribute B).

Structure of the paper. The remainder of this paper is structured as follows. Section 3 introduces cross tabulations, one of the kernel operations of OLAP. Section 4 gives a brief overview of the *typed linear algebra* notation adopted in the paper, taken from [16]. Section 5 expresses cross tabulations solely in terms of linear algebra matrix operations. Section 6 builds up on cross-tabbing and "rolls-up" on functional dependencies, introducing dimension hierarchies into the game. Section 7 proves that construction of cross-tabulations is incremental. Section 8 goes higher-dimensional into the LA construction of OLAP cubes. Finally, section 9 reviews related work and Section 10 draws some conclusions, giving a prospect of future work.

3 Cross-tabulations

In data processing, a cross tabulation (or pivot table) provides a particular summary or view of data extracted from a raw data source. As example of raw data consider the table displayed in Figure 1 where each row records the number of vehicles of a given model and color sold per year.

Model	Year	Color	Sales
Chevy	1990	Red	5
Chevy	1990	Blue	87
Ford	1990	Green	64
Ford	1990	Blue	99
Ford	1991	Red	8
Ford	1991	Blue	7

Fig. 1 Collection of raw data (adapted from [10]).

In general, the raw-data out of which cross tabulations are calculated is not normalized and is collected into a central database (termed a data *warehouse*, or decision support database) containing huge amounts of information obtained from disparate sources. Such a central warehouse — typically, a table with an absurd number of lines — is not easy (if at all possible) to manually inspect and analyse. To obtain useful information from it one needs to summarize the data by selecting attributes of interest and exhibiting their inter-relationships.

Different summaries answer to different questions such as, for instance *"how many vehicles were sold per color and model?"*, For this particular question, the attributes $Color$ and $Model$ are selected as *dimensions* of interest, $Sales$ is regarded as *measure* and the corresponding cross tabulation is depicted in Figure 2, as generated via the pivot table menu in Excel.

Large scale cross tabulation generation is an essential part of OLAP. Broadly speaking, OLAP refers to the technique of performing sophisticated analysis over the information stored in a data warehouse, whose complexity is well-known [24]. As mentioned in [5], numerous SQL extensions are offered by

Sum of Sales	Model		
Color	Chevy	Ford	Grand Total
Blue	87	106	193
Green		64	64
Red	5	8	13
Grand Total	92	178	270

Fig. 2 Pivot table in Excel extracted from the data in Figure 1.

many vendors of OLAP products trying to address this problem. The solution we put forward in this paper does not try to solve it inside the OLAP and data warehousing technologies, but rather calls for a synergy with the field of linear algebra application, where satisfactory solutions have been found for similarly complex operations in other domains such as eg. computer graphics and DSP [27].

The key resides in expressing OLAP operations in the form of matrix algebra expressions which can be parallelized [1,32]. In the particular case of reporting multi-dimensional analyses of data, one should be able to build three matrices, according to the hint given by formula (4): two associated to the dimensions (attributes) A and B being analysed and a third recording which *measure* or *metric* attribute is to be considered for consolidation.

This encoding of data into LA is quite smooth if matrix operations are *typed* in the way presented in [16]. For self-containedness we give a very brief overview of such *typed LA* notation below.

4 Typed linear algebra

Matrices as arrows. A matrix A with n rows and m columns is a function $A(r, c)$ which tells the value occupying each cell (r, c), for $1 \leqslant r \leqslant n$, $1 \leqslant c \leqslant m$. In this paper we will follow the arrow notation of [16] and write $n \xleftarrow{A} m$ to denote that matrix A is of type $n \longleftarrow m$ (m columns, n rows). Thus matrix multiplication can be expressed by arrow composition:

$$n \xleftarrow{A} m \xleftarrow{B} k \qquad C = A \cdot B \tag{5}$$

For every n there is a matrix of type $n \longleftarrow n$ which is the unit of composition. This is nothing but the identity matrix of size n, denoted by $n \xleftarrow{id_n} n$ or $n \xleftarrow{1} n$, indistinguishably. Therefore:

$$id_m \cdot A \;=\; A \;=\; A \cdot id_n \tag{6}$$

$$\begin{array}{ccc} n & \xleftarrow{id_n} & n \\ {\scriptstyle A}\downarrow & \swarrow_{A} & \downarrow{\scriptstyle A} \\ m & \xleftarrow[id_m]{} & m \end{array}$$

Subscripts m and n can be omitted wherever the underlying diagrams are assumed.

Vectors as arrows. Vectors are special cases of matrices in which one of the dimensions is 1, for instance

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \qquad \text{and} \qquad w = \begin{bmatrix} w_1 \ \ldots \ w_n \end{bmatrix}$$

Column vector v is of type $m \longleftarrow 1$ (m rows, one column) and row vector w is of type $1 \longleftarrow n$ (one row, n columns). Our convention is that lowercase letters (eg. v, w) denote vectors and uppercase letters (eg. A, M) denote arbitrary matrices.

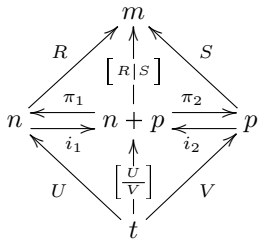
Types. Matrix types (the end points of arrows) can be generalized to arbitrary (finite) sets thanks to addition and multiplication being commutative and associative. This ensures unambiguous definition of matrix composition because the summation inside the inner product of two vectors can be calculated in any order. Typewise, our convention is that lowercase letters (eg. n, m) denote the traditional dimension types (natural numbers), letting uppercase letters denote other types.

Converse of a matrix. One of the kernel operations of linear algebra is *transposition*, whereby a given matrix changes shape by turning its rows into columns and vice-versa. Given matrix $n \xleftarrow{A} m$, notation $m \xleftarrow{A^\circ} n$ denotes its transpose, or converse. The following idempotence and contravariance laws hold:

$$(A^\circ)^\circ = A \tag{7}$$

$$(A \cdot B)^\circ = B^\circ \cdot A^\circ \tag{8}$$

Block notation. Matrices can be built of other matrices using block notation. Two basic binary combinators are identified in [16] for building matrices out of other matrices, say A and B, regarded as blocks, either stacking these vertically, $\begin{bmatrix} A \\ \hline B \end{bmatrix}$, or horizontally, $\left[\, A \middle| B \,\right]$. Dimensions should agree, as shown in the diagram below, taken from [16], where m, n, p and t are types:



Special matrices i_1, i_2, π_1 and π_2 are fragments of the identity matrix and play an important role in explaining the semantics of the two combinators. This, however, can be skipped for the purposes in the current paper. (The interested reader is referred to [16] for details.)

The *exchange law*

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = \left[\left[\begin{array}{c} A \\ \hline C \end{array}\right] \middle\| \left[\begin{array}{c} B \\ \hline D \end{array}\right]\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{9}$$

tells the equivalence between row-major and column-major construction of matrices by blocks. Thus the four-block notation on the right [16].

Direct sums. Given two arbitrary matrices A and B, the *direct sum* of A and B is defined as follows, using block notation:

$$A \oplus B = \left[\begin{array}{c|c} A & 0 \\ \hline 0 & B \end{array}\right] \tag{10}$$

Mind the types (dimensions):

$$\begin{array}{ccc} n & m & n+m \\ {\scriptstyle A}\downarrow & {\scriptstyle B}\downarrow & \downarrow{\scriptstyle A \oplus B} \\ k & j & k+j \end{array}$$

Direct sum is a standard linear algebra operator enjoying many useful properties [16]. The following equation, termed the *absorption law*, specifies how block operator $\left[\,|\,\right]$ absorbs direct sum $\oplus$, for suitably typed matrices A, B, C and D:

$$\left[\, A \middle| B \,\right] \cdot (C \oplus D) = \left[\, A \cdot C \middle| B \cdot D \,\right] \tag{11}$$

Khatri-Rao matrix product. Given matrices $n \xleftarrow{A} m$ and $p \xleftarrow{B} m$, the so-called Khatri-Rao [28] matrix product of A and B, denoted $n \times p \xleftarrow{A \odot B} m$ is a column-wise Kronecker product,

$$\begin{array}{c} u \odot v = u \otimes v \\ \left[\, A_1 \middle| A_2 \,\right] \odot \left[\, B_1 \middle| B_2 \,\right] = \left[\, A_1 \odot B_1 \middle| A_2 \odot B_2 \,\right] \end{array} \tag{12}$$

where u, v are column-vectors and A_i, B_i are suitably typed matrices. As an example of operation relying on this product consider row vector

$$s = \left[5\ 87\ 64\ 99\ 8\ 7\right]$$

of type $1 \xleftarrow{s} 6$, capturing the transposition of the *Sales* column of Figure 1. Then Khatri-Rao product $s \odot id$ is the corresponding diagonal matrix:

$$6 \xleftarrow{s \odot id} 6 \;=\; \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 87 & 0 & 0 & 0 & 0 \\ 0 & 0 & 64 & 0 & 0 & 0 \\ 0 & 0 & 0 & 99 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix} \tag{13}$$

This conversion is essential to the LA encoding of cross tabulations, as shown next.

5 Cross tabulations in LA

Recall that the core of cross tabulation generation is formula (4), which is the matrix counterpart to relational projection (2). This section explains this construct starting by showing how the move from relations to matrices is obtained by encoding functions as matrices.

Building projection functions. Let A be an attribute of raw-data table T and let n be the number of records in T (vulg. rows, or lines in a spreadsheet). We write $T(A)$ to denote the column of T identified by attribute A, $T(A, y)$ to denote the element occupying the y-th position (row) in such a column, and $|A|$ to denote the range of values which can be found in $T(A)$. Column $T(A)$ can be regarded as a function which tells, for each row number $1 \leqslant r \leqslant n$, which value x of $|A|$ can be found in row r of such a column. Such a function can be encoded as an elementary matrix of type $|A| \xleftarrow{t_A} n$, defined as follows [5]:

$$t_A(x, r) = \begin{cases} 1 & \text{if } T(A, r) = x \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In our running example (Figures 1 and 2), $n = 6$ and we want to build these matrices for attributes *Model* and *Color*. The projection $|Model| \xleftarrow{t_{Model}} n$ associated to dimension *Model* is matrix

	1	2	3	4	5	6
Chevy	1	1	0	0	0	0
Ford	0	0	1	1	1	1

and projection $|Color| \xleftarrow{t_{Color}} n$ associated to dimension *Color* is matrix

	1	2	3	4	5	6
Blue	0	1	0	1	0	1
Green	0	0	1	0	0	0
Red	1	0	0	0	1	0

[5] These projections can be identified with the *bitmaps* of [34], regarded as matrices.

Note that, typewise, the composition of matrices t_{Color} and t°_{Model} makes sense, leading to a matrix of type $|Color| \prec\!\!\longleftarrow |Model|$,

$$t_{Color} \cdot t^{\circ}_{Model} = \begin{array}{r|cc} & Chevy & Ford \\ \hline Blue & 1 & 2 \\ Green & 0 & 1 \\ Red & 1 & 1 \end{array} \tag{15}$$

which essentially counts the number of sale records per colour and model. This situation (*counting*), which is what Excel outputs wherever the measure attribute chosen in pivot table calculation is not numeric, corresponds to formula (4) wherever the middle matrix is the identity.

The diagonal construction. In order to sum up the number of vehicles sold rather than just counting sale records we need to identify a *measure* attribute, that is, a numeric attribute of T to be used for consolidation. In the case of Figure 1 only *Sales* applies. Because such numeric data have to become available for both projection matrices, the column chosen is converted into a diagonal matrix, as already shown as an illustration of Khatri-Rao (13).

Notation $[\![T]\!]_M$ will be used to denote the diagonal matrix representation of measure attribute M in T. Index-wise, this corresponds to the following definition:

$$[\![T]\!]_M(j,i) = \begin{cases} T(M,j) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

LA script for cross tabulation. We are now in position to run formula (4) for T as in Figure 1, $A = Colour$ and $B = Model$, obtaining another matrix of type $|Color| \prec\!\!\longleftarrow |Model|$:

$$t_{Color} \cdot [\![T]\!]_{Sales} \cdot t^{\circ}_{Model} \quad = \quad \begin{array}{r|cc} & Chevy & Ford \\ \hline Blue & 87 & 106 \\ Green & 0 & 64 \\ Red & 5 & 8 \end{array} \tag{16}$$

If compared to Figure 2, cross tabulation (16) misses the two row and column grand totals. These are easily obtained via *"bang" matrices*. Let us explain what these are and our choice of terminology.

In functional programing, the popular "bang" function, which is of type $1 \leftarrow A$ and usually denoted by symbol "!", is a polymorphic constant function yielding the unique value which inhabits the singleton type 1 [6]. The encoding of this function in LA format will be matrix $1 \prec\!\!\overset{!_A}{\longleftarrow} A$ wholly filled up with 1s — a row vector. For instance, $!_{|Model|}$ will be the row vector with $|Model|$-many positions all holding number 1. The MATLAB equivalent to $1 \prec\!\!\overset{!_n}{\longleftarrow} n$ is `ones(1,n)`, see Listing 1 in the appendix.

[6] See [2]. In Haskell, both this type and its inhabitant are denoted by "()". For the purposes in this paper, 1 can be regarded as the singleton set {ALL}.

Clearly, the composition of row vector $1 \xleftarrow{!} A$ with any column vector of type $A \xleftarrow{v} 1$ computes the singleton vector holding the sum of all cells in v. Thus one can extend formula (4) with *bang* vectors so as to equip cross tabulations with grand totals, by defining

$$\begin{aligned} & ctab_{A,B;M}(T) : |A|+1 \leftarrow |B|+1 \\ & ctab_{A,B;M}(T) \;=\; \left[\frac{t_A}{!}\right] \cdot [\![T]\!]_M \cdot \left[\frac{t_B}{!}\right]^{\circ} \end{aligned} \tag{17}$$

which computes the cross tabulation of raw data table T with respect to dimensions A, B and measure M. Note that types (dimensions) have been added a new entry (1), which can be understood as a singleton type containing a distinguished element, say ALL, labelling grand totals. This corresponds, in our running example, to enriching (16) with the extra row and column corresponding to the added *bang* vectors, both labeled with ALL,

$$ctab_{Color,Model;Sales}(T) \;=\; \begin{array}{r|ccc} & Chevy & Ford & \text{ALL} \\ \hline Blue & 87 & 106 & 193 \\ Green & 0 & 64 & 64 \\ Red & 5 & 8 & 13 \\ \text{ALL} & 92 & 178 & 270 \end{array} \tag{18}$$

finally achieving the effect of Figure 2 with LA operations only. The MATLAB script for formula (17) is given in Listing 1 provided in the appendix.

Among the many properties of "bang" matrices we single out

$$\left[\,!\,|\,!\,\right] = !\tag{19}$$

and

$$! \odot A = A = A \odot !\tag{20}$$

which tells ! the unit of Khatri-Rao product. Since this is associative too, we can rely on its extension to n argument matrices A_i $(1 \leqslant i \leqslant n)$ by writing $\bigodot_{i=1}^{n} A_i$ or even

$$\bigodot_{i \leftarrow s} A_i \tag{21}$$

where s is a sequence of indices. This extension will be useful in the generation of data cubes given in Section 8. Prior to this, we address below another operation central to OLAP: *roll-up*.

6 "Rolling-up" on functional dependencies

It can be shown via blocked matrix algebra [16] that the matrix composition of (17) unfolds into four blocks, namely

$$\left[\begin{array}{c|c} t_A \cdot [\![T]\!]_M \cdot t_B^{\circ} & t_A \cdot [\![T]\!]_M \cdot !^{\circ} \\ \hline ! \cdot [\![T]\!]_M \cdot t_B^{\circ} & ! \cdot [\![T]\!]_M \cdot !^{\circ} \end{array}\right] \tag{22}$$

whose pre- and post-compositions with "bang matrices" can already be regarded as examples of the OLAP operation known as *roll-up*.

Rolling-up means replacing a dimension by another which is more general in some sense (eg. grouping, classification, containment). The latter is therefore "higher" in a dimension hierarchy which somehow acts as a *classification* or *taxonomy* of data records.

A simple way of seeing roll-up at work is the acknowledgement of functional dependencies (FDs) [17] in data. Let us, for instance, augment the raw data of our running example with two new columns recording the month and season of each sale, as displayed in Figure 3.

Model	Year	Color	Sales	Month	Season
Chevy	1990	Red	5	March	Spring
Chevy	1990	Blue	87	April	Spring
Ford	1990	Green	64	August	Summer
Ford	1990	Blue	99	October	Autumn
Ford	1991	Red	8	January	Winter
Ford	1991	Blue	7	January	Winter

Fig. 3 Augmented collection of raw data.

Look, for instance, at the column labelled $Season$ in Figure 3, telling in which season ($Spring$, $Summer$, $Autumn$ or $Winter$) the particular sales took place. It is clear that FD $Season \leftarrow Month$ holds, as every month belongs to one and only one season. In other words, $Season$ is higher in the dimension hierarchy than $Month$ [7].

Roll-up matrices. In general, functional dependency $B \leftarrow A$ will hold in a table T iff no pair of rows can be found in which the values of attribute A are the same and those of attribute B differ ("B is determined by A"). That is, B acts as a *classifier* for A, meaning that every cross tabulation involving A can be *rolled-up* into another (less detailed) involving B instead.

Interestingly, the *roll-up* matrix $|B| \xleftarrow{t_{B \leftarrow A}} |A|$ associated to FD $B \leftarrow A$ is simply given by

$$t_{B \leftarrow A} = t_B \cdot t_A^\circ \tag{23}$$

(We hope (23) convinces the reader of the advantage of writing FDs the other way round, namely $B \leftarrow A$ instead of the more conventional $A \rightarrow B$ [17].) For instance, the roll-up matrix calculated from FD $Season \leftarrow Month$ is:

$$t_{Season} \cdot t_{Month}^\circ = \begin{array}{r|ccccc} & January & March & April & August & October \\ \hline Spring & 0 & 1 & 1 & 0 & 0 \\ Summer & 0 & 0 & 0 & 1 & 0 \\ Autumn & 0 & 0 & 0 & 0 & 1 \\ Winter & 2 & 0 & 0 & 0 & 0 \end{array} \tag{24}$$

[7] The fact that T is not normalized in general reflects the preparation process of merging into the same data warehouse different tables of a (normalized) database.

Note how matrix (24) is "functional" in the sense that at most one non-zero cell can be found in each column.

So, given a cross tabulation matrix $|A|+1 \xleftarrow{X} |C|+1$, the effect of rolling it up across a given FD $B \leftarrow A$ is another cross tabulation given by matrix

$$(t_{B \leftarrow A} \oplus id) \cdot X$$

of type $|B|+1 \xleftarrow{\quad} |C|+1$. Notice how $(\oplus\ id)$ handles the tabulation's ALL field not present in the roll-up matrix. Converse (transpose) caters for the same effect on the right-hand side: rolling X up across another FD $C \leftarrow D$ is matrix

$$X \cdot (t_{D \leftarrow C} \oplus id)^{\circ}$$

of type $|A|+1 \xleftarrow{X} |D|+1$. We illustrate this below by instantiating X with a cross tabulation from $Model$ to $Month$

$$ctab_{Month,Model;Sales}(T) =$$

	Chevy	*Ford*	ALL
January	0	15	15
March	5	0	5
April	87	0	87
August	0	64	64
October	0	99	99
ALL	92	178	270

which, once composed with roll-up matrix (24) extended with totals, yields the expected rolling up effect:

$$(t_{Season \leftarrow Month} \oplus id) \cdot ctab_{Month,Model;Sales}(T) =$$

	Chevy	*Ford*	ALL
Spring	92	0	92
Summer	0	64	64
Autumn	0	99	99
Winter	0	15	15
ALL	92	178	270

Checking for FDs. Construction (23) enables us to check for functional dependencies. In general, FD $B \leftarrow A$ will hold wherever matrix $t_{B \leftarrow A}$ is *functional*, or *simple*. This terminology is imported from relational algebra and allegory theory [9]: a $\mathbb{N}_0$-valued matrix S will be said to be simple iff its *image* $S \cdot S^{\circ}$ is diagonal.

It can be checked that the image of *roll-up* matrix (24) is diagonal

	Spring	*Summer*	*Autumn*	*Winter*
Spring	2	0	0	0
Summer	0	1	0	0
Autumn	0	0	1	0
Winter	0	0	0	4

while that of (15)

	Blue	*Green*	*Red*
Blue	5	2	3
Green	2	1	1
Red	3	1	2

is not. Thus, FD $Color \leftarrow Model$ does not hold.

Of course, projections are functional (simple) — in fact, they are matrix representations of surjections (surjective functions), recall (14). A simple matrix A is said to be a *surjection* iff the sum of each column of A is 1, in which case $A \cdot A^\circ = id$. So, ! matrices are surjections and the following *natural* law holds:

$$! \cdot A = ! \quad \Leftarrow \quad A \text{ is a function} \tag{25}$$

This law is enough to ensure the following property: *roll-up* preserves cross tabulation grand totals.

Further developments. The matrix representation of FDs opens further perspectives on the *roll-up* OLAP operation, as the following matrix

	Jan	*Feb*	*Mar*	*Apr*	*May*	*Jun*	*Jul*	*Aug*	*Sep*	*Oct*	*Nov*	*Dec*
Spring	0	0	0.3	1	1	0.7	0	0	0	0	0	0
Summer	0	0	0	0	0	0.3	1	1	0.7	0	0	0
Autumn	0	0	0	0	0	0	0	0	0.3	1	1	0.7
Winter	1	1	0.7	0	0	0	0	0	0	0	0	0.3

of type $Season \prec\!\!\!\longleftarrow Month$ shows. In this case, FD $Season \leftarrow Month$ does not strictly hold, for equinoctial and solsticial months are doubly classified in the seasons they border, in different proportions (70% for the season which ends, 30% for the one which starts up).

Perhaps one might say that a "fuzzy" data dependency holds in this situation. In spite of the possible complexity that this extension of the previous situation might raise in the traditional OLAP perspective, in our setting it doesn't change anything, as such "fuzzy" months-into-seasons roll-up process would work precisely in the same way, always relying on matrix multiplication and transposition.

7 Incremental construction

Cross tabulations as defined by formula (17) are amenable to incremental construction under certain conditions. For instance, suppose one is given yesterday's cross tab and today's new data. Then today's cross tab (in matricial form) will be obtained by adding to the former (matrix-wise) the cross tab of the latter.

As an illustration of how LA support helps in proving facts about data mining operations, we give below the proof of incremental cross tabulation construction. Let T be yesterday's raw data and T' be the new data. Assuming that T has remained the same (no updates, no deletes), let $T'' = T;T'$ denote the append of the two data sources. Then the following facts hold,

$$t''_A = \left[t_A | t'_A \right] \tag{26}$$

$$t''_B = \left[t_B | t'_B \right] \tag{27}$$

$$[\![T;T']\!]_M = [\![T]\!]_M \oplus [\![T']\!]_M \tag{28}$$

where $\oplus$ builds a diagonal matrix by direct sum (10) of two diagonal matrices.

The proof that cross tabulation is incremental

$$ctab_{A,B;M}(T;T') = ctab_{A,B;M}T + ctab_{A,B;M}T' \tag{29}$$

stems from its definition (17) and follows by simple equational reasoning, using the laws of matrix algebra:

$$\begin{array}{ll}
& ctab_{A,B;M}(T;T') \\
\Leftrightarrow & \quad \{\ (17)\ \} \\
& \left[\frac{t''_A}{!}\right] \cdot [\![T;T']\!]_M \cdot \left[\frac{t''_B}{!}\right]^\circ \\
\Leftrightarrow & \quad \{\ (26)\ ;\ (27) \text{ and } (28)\ \} \\
& \left[\frac{\left[t_A \middle| t'_A\right]}{!}\right] \cdot ([\![T]\!]_M \oplus [\![T']\!]_M) \cdot \left[\frac{\left[t_B \middle| t'_B\right]}{!}\right]^\circ \\
\Leftrightarrow & \quad \{\ (19) \text{ twice }\ ;\ \text{exchange law } (9) \text{ twice}\ \} \\
& \left[\left[\frac{t_A}{!}\right] \middle| \left[\frac{t'_A}{!}\right]\right] \cdot ([\![T]\!]_M \oplus [\![T']\!]_M) \cdot \left[\left[\frac{t_B}{!}\right] \middle| \left[\frac{t'_B}{!}\right]\right]^\circ \\
\Leftrightarrow & \quad \{\ \text{absorption } (11)\ \} \\
& \left[\left[\frac{t_A}{!}\right] \cdot [\![T]\!]_M \middle| \left[\frac{t'_A}{!}\right] \cdot [\![T']\!]_M\right] \cdot \left[\frac{\left[\frac{t_B}{!}\right]^\circ}{\left[\frac{t'_B}{!}\right]^\circ}\right] \\
\Leftrightarrow & \quad \{\ \text{divide and conquer matrix multiplication } (3)\ \} \\
& \left[\frac{t_A}{!}\right] \cdot [\![T]\!]_M \cdot \left[\frac{t_B}{!}\right]^\circ + \left[\frac{t'_A}{!}\right] \cdot [\![T']\!]_M \cdot \left[\frac{t'_B}{!}\right]^\circ \\
\Leftrightarrow & \quad \{\ (17) \text{ twice}\ \} \\
& ctab_{A,B;M}T + ctab_{A,B;M}T'
\end{array}$$

In retrospect, this proof establishes *ctab* as a structure preserving map between raw data collection and (cross tabulation) matrix addition.

8 Higher-dimensional OLAP

In this section we proceed beyond cross tabulations generation to achieve higher-dimensionality. The aim is to formulate a general LA theory for n-dimensional OLAP, dealing with all data summary levels presented in [12], from 0 to 3-dimensional summaries, respectively: aggregate, group-by, cross-tab and cube. The approach goes further by allowing any number n of dimensions.

The proposed generalization depends on the Khatri-Rao product (12) that works as a Cartesian product operator on the types of the matrix, thus a Cartesian product of the dimensions. As an illustration, remember the projections of our running example and apply this product to t_{Model} and t_{Color}. The outcome is a matrix bearing type $|Model \times Color| \leftarrow\!\!- 6$:

		1	2	3	4	5	6
Chevy	*Blue*	0	1	0	0	0	0
Chevy	*Green*	0	0	0	0	0	0
Chevy	*Red*	1	0	0	0	0	0
Ford	*Blue*	0	0	0	1	0	1
Ford	*Green*	0	0	1	0	0	0
Ford	*Red*	0	0	0	0	1	0

It tells in which rows the particular dimension pairs appear, compare with Figure 1. Put in other words, this matrix is the higher-rank projection $t_{Model \times Color}$ of the Cartesian product of the two dimensions. In general,

$$t_{A \times B} = t_A \odot t_B \tag{30}$$

Thus

$$\begin{aligned} & t_{Model \times Year \times Color} \\ = \; & t_{Model} \odot t_{Year} \odot t_{Color} \\ = \; & \end{aligned} \tag{31}$$

			1	2	3	4	5	6
Chevy	1990	*Blue*	0	1	0	0	0	0
Chevy	1990	*Green*	0	0	0	0	0	0
Chevy	1990	*Red*	1	0	0	0	0	0
Chevy	1991	*Blue*	0	0	0	0	0	0
Chevy	1991	*Green*	0	0	0	0	0	0
Chevy	1991	*Red*	0	0	0	0	0	0
Ford	1990	*Blue*	0	0	0	1	0	0
Ford	1990	*Green*	0	0	1	0	0	0
Ford	1990	*Red*	0	0	0	0	0	0
Ford	1991	*Blue*	0	0	0	0	0	1
Ford	1991	*Green*	0	0	0	0	0	0
Ford	1991	*Red*	0	0	0	0	1	0

is the projection capturing the whole dimensional part of the raw-data table of Figure 1.

Multidimensional cross tabulations are obtained via the same formula (17) just by supplying higher-rank projections, for instance

$$ctab_{Model \times Color, Year; Sales}(T) =$$

		1990	1991	ALL
Chevy	*Blue*	87	0	87
Chevy	*Green*	0	0	0
Chevy	*Red*	5	0	5
Ford	*Blue*	99	7	106
Ford	*Green*	64	0	64
Ford	*Red*	0	8	8
ALL		255	15	270

corresponding to $A = Model \times Color$ and $B = Year$ in (17). Furthermore, by composing $[\![T]\!]_{Sales}$ with the projection of all dimensions which, as we

have seen (31), is obtained by the Khatri-Rao product, we obtain the three dimensional part of the CUBE operator:

$$t_{Model \times Year \times Color} \cdot [\![T]\!]_{Sales} \cdot !^{\circ} =$$

			Sales
Chevy	1990	*Blue*	87
Chevy	1990	*Green*	0
Chevy	1990	*Red*	5
Chevy	1991	*Blue*	0
Chevy	1991	*Green*	0
Chevy	1991	*Red*	0
Ford	1990	*Blue*	99
Ford	1990	*Green*	64
Ford	1990	*Red*	0
Ford	1991	*Blue*	7
Ford	1991	*Green*	0
Ford	1991	*Red*	8

A generalization follows from this example. Given an ordered set of dimensions D and a measure M, to calculate the corresponding cube iterate over the powerset of D and, for each set of dimensions s in the powerset (regarded as a sequence induced by the predefined order on dimensions) build the corresponding projection using an iteration of (30) over s — recall also (21). Finally, multiply by the measure and bang converse as presented in:

$$cube_{D;M}(T) = \bigoplus_{s \in 2^D} (\bigodot_{d \leftarrow s} t_d) \cdot [\![T]\!]_M \cdot !^{\circ} \tag{32}$$

Remember that $\bigoplus$, the n-ary extension of vertical blocking (recall Section 4), stacks blocks vertically and is therefore just a glue of the intermediate results provided by the outermost iteration.

CUBE as a MATLAB *script.* To close the illustration of our approach, we detail an implementation of formula (32) made available as a MATLAB script `Cube.m`. The core of our experimental script (see Listing 2 in the appendix) is a function with the same name which receives as input an array of the projections for each dimension (*proj*) and the measure diagonal (*dnum*). It then outputs the result of the CUBE operator. It is tuned for the example in paper [10], and to run it one needs to pass as parameters the projection matrices as defined in the current paper. By running

```
>> CleanAndShow(Cube({m,y,c},d))
```

in MATLAB, where variables m,y,c and d respectively hold t_{Model}, t_{Year}, t_{Color} and $T(Sales)$, we will obtain the result displayed below.

			Sales
Chevy	1990	*Blue*	87
Chevy	1990	*Red*	5
Ford	1990	*Blue*	99
Ford	1990	*Green*	64
Ford	1991	*Blue*	7
Ford	1991	*Red*	8
Chevy	1990	*ALL*	92
Ford	1990	*ALL*	163
Ford	1991	*ALL*	15
Chevy	*ALL*	*Blue*	87
Chevy	*ALL*	*Red*	5
Ford	*ALL*	*Blue*	106
Ford	*ALL*	*Green*	64
Ford	*ALL*	*Red*	8
ALL	1990	*Blue*	186
ALL	1990	*Green*	64
ALL	1990	*Red*	5
ALL	1991	*Blue*	7
ALL	1991	*Red*	8
Chevy	*ALL*	*ALL*	92
Ford	*ALL*	*ALL*	178
ALL	1990	*ALL*	255
ALL	1991	*ALL*	15
ALL	*ALL*	*Blue*	193
ALL	*ALL*	*Green*	64
ALL	*ALL*	*Red*	13
ALL	*ALL*	*ALL*	270

Generic Matricial Aggregation. A general formula for calculating aggregations on a given ordered set D of dimensions and measure M from a database table T is given by

$$\bigoplus_{j \in \mathsf{F}(D)} (\bigodot_{d \leftarrow j} t_d) \cdot [\![T]\!]_M \cdot !^{\circ}$$

where generic set construct $\mathsf{F}(D)$ tells how dimensions in D are handled. Different operations are obtained by instantiating $\mathsf{F}(D)$. For instance, by making $\mathsf{F}(D) = \varnothing$ one obtains an AGGREGATE [12] ; for $\mathsf{F}(D) = \{D\}$ the result is a GROUP-BY; for $\mathsf{F}(D)$ issuing the set of prefixes of D, one gets a ROLL-UP; finally, for F providing the powerset of D one obtains, as shown in (32), a data CUBE.

9 Related Work

An overview of data warehousing and OLAP technology can be found in [3]. Since Gray et al delivered their seminal data cube paper in 1996 [12], most work in the field has been concerned with techniques for efficient OLAP, given the small time window (usually at night) when warehouses can go offline for data refreshing.

Yang et al [35] focus on the problem of data cube construction and show how a cluster middleware, called ADR (originally developed for scientific data intensive applications) can be used for carrying out scalable data cube construction implementation.

Bearing the ideal of making OLAP "truly online", Ng et al [19] develop a collection of parallel algorithms directed towards online and offline creation of data cubes using low cost PC clusters to parallelize computations.

Goil and Choudhary [11] address scalability in multidimensional systems for OLAP and multidimensional analysis and describe the PARSIMONY system providing a parallel and scalable infrastructure for multidimensional online analytical processing, used for both OLAP and data mining. Sparsity of data sets is handled by using chunks to store data either as a dense block using multidimensional arrays or as sparse representation using a bit encoded sparse structure. Parallel algorithms are developed for data mining on the multidimensional cube structure for attribute-oriented association rules and decision-tree-based classification. Performance results for high dimensional data sets on a distributed memory parallel machine (IBM SP-2) show good speedup and scalability.

Recent publications in "end-to-end" system proposals for parallel OLAP servers are scarce. SIDERA [6] is one such proposal, providing OLAP-specific functionality gathering recent results in a common framework: *"the most comprehensive OLAP platform described in the current research literature"* [6].

Closer to our approach, Sun and others [30,31] introduce a technique based on the use of tensors in the area of pattern discovery. (Tensors generalize vectors and matrices, as happens in the mathematical domain, and can be used to represent data-cubes.) To capture temporal evolution one uses tensor streams or sequences that are time indexed structures of tensors, the advantage of this kind of streams being the generalization of traditional streams and sequences.

On the background stays *singular value decomposition* (SVD), whose matricial expression conspicuously resembles our starting point (4) and suggests a link between the two approaches which we intend to study in the future.

Our work also intersects with the area of index based database query (response time) optimization, namely in what respects *bitmap* indices [34]. Clearly, the projection matrices built in the current paper are bitmaps regarded as matrices. Bitmaps were first implemented in IBM's Model 204 [23], becoming a "de facto" device after compression techniques solved their outrageous memory space demands. They are still in use in today's commercial database systems, see [34] for details.

Concerning LA kernels for parallel machines, Bell and Garland [1] explore the design of efficient sparse matrix-vector kernels for throughput oriented processors and implement these kernels in a parallel computing architecture developed by NVIDIA. The Optimized Sparse Kernel Interface (OSKI) Library [32] is a collection of low-level C primitives that provide automatically tuned computational kernels on sparse matrices, for use in solver libraries and

applications. OSKI has a BLAS-style interface, providing basic kernels like sparse matrix-vector multiply and sparse triangular solve, among others.

Last but not least, Yang et al [36] propose architecture-aware optimizations for sparse matrix multiplication on GPUs and study the impact of their efforts on graph mining. This work is another piece of evidence suggesting that future OLAP and data mining should rely on linear algebra.

10 Conclusions and Future Work

This paper proposes a separation of concerns in approaching parallel OLAP. The strategy consists in first encoding OLAP functionality solely in terms of LA (matrix) operations, and then relying on the theory of parallel sparse matrix/matrix and matrix/vector multiplication to achieve parallelism [32]. All operations in the approach, namely

- the conversion of dimension attributes into projection matrices
- the conversion of measure attributes into diagonal matrices
- the calculation of cross tabulations
- the calculation of cubes

are *embarassingly parallel* [8]. Both projection and diagonal matrices are sparse, therefore calling for suitably optimization in a parallel environment [32].

The paper focuses on the first part of the approach — something one might call LAOLAP (for "linear algebra OLAP") — letting the actual parallel implementation over multi-core LA kernels for future work, as explained below. Moreover, it illustrates how the kinship between relation and matrix algebra suggests how the LA approach to OLAP should proceed. To the best of the authors' knowledge this technique is novel in the field and deserves further attention.

Although the overall strategy can be regarded as a *noSQL* [18], or "SQL-free" approach to OLAP data processing, an LA semantics could be developed for SQL accordingly, think for instance of syntactic constructs such as GROUP BY. Further to these and to the matricial projections dealt with already in the current paper, the Khatri-Rao product can be used to perform *selections* using attribute membership vectors as representations of sets of values.

Future work. Given the two separate steps of our approach, future work will proceed in two independent directions, one going deeper into the LA encoding of data mining operations and the other actually implementing the overall approach and benchmarking it with respect to state-of-the-art parallel OLAP technology.

On the foundations side, much work has to be carried out, namely in providing a proper "justification" of the approach. In particular, we have to cross-check our matrix encoding of OLAP (and FDs) with already existing OLAP formal models, such as given in [5, 24] and likely elsewhere. Mimicking OLAP algebra (whatever this means) in terms of linear algebra may provide better

and simpler proofs for existing results and possibly generate new ones, as our experience in pointfree calculation already shows, in the relational algebra field [20,22]. And, of course, a closer look at [30] is also in the research agenda.

Extending LA support for other forms of data consolidation such as eg. averaging is at immediate reach. For instance, averaging rather than summing up measure vectors is obtained once again via *bang* matrices, as the following formula shows,

$$avg\ v = \frac{! \cdot v}{! \cdot !^\circ}$$

for $n \xleftarrow{v} 1$ and $1 \xleftarrow{!} n$, reducing v into the scalar which holds its sum. Averaging holds since $(! \cdot !^\circ)$ is $1 \xleftarrow{n} 1$, also a scalar. This generalizes to weighted averaging: given two vectors $n \xleftarrow{v,w} 1$ where w records weights,

$$\overline{v}_w = \frac{x}{y} \quad \textbf{where} \quad \left[\frac{x}{y}\right] = \left[\frac{v^\circ}{!}\right] \cdot w$$

Back to our case study, it is easy to see that obtaining cross tabulations consolidated by averaging is a question of augmenting equation (4) with the (index-wise) division of the cross tabulation matrix by the corresponding counting matrix:

$$\frac{t_A \cdot [\![T]\!]_M \cdot t_B^\circ}{t_A \cdot t_B^\circ}$$

Extremes (min and max) are easy to calculate, achievable by changing the semantics of the multiplication and sum of elements in the matrix. But calculating more exotic data consolidation forms as eg. population's standard deviation is a challenge to overcome due to the complexity of the formulas. We have been able to achieve it with intensive use of Khatri-Rao products and other non-trivial matrix operations, but further research is needed to evaluate the practicality of such usage.

On the practical side, our main expectations reside in actually implementing our LA OLAP formulæ over a fast kernel for parallel implementation of sparse matrix algebra, such as eg. [1,32], and benchmark the overall result with respect to standard parallel OLAP implementations such as eg. PARSIMONY [11].

It will be interesting to see how much of current *bitmap* technology [34] can be used in our approach to obtain optimal projection function implementations. We point out that further study of the relationship between bitmap compression and sparse matrix representation techniques is of interest to the whole spectrum of this research field.

The prospect of extending such techniques to spreadsheet software running on multi-core laptops is also of interest [8]. We are currently getting involved in

[8] See project SSaaPP: *Spread Sheets as a Programming Paradigm* in the HASLab project portfolio:
`http://wiki.di.uminho.pt/twiki/bin/view/DI/FMHAS/Projects`.

an open source initiative targeting at incorporating our ideas into the calculation of summary tables in OpenOffice taking advantage of lap-top multi-core hardware architecture.

Last but not least, it would be interesting to see how much could be gained from implementing our approach in computational software programs used in scientific engineering such as eg. *Mathematica* [9] [33], namely by putting its matrix tools and support for sparse arrays in the service of packages such as eg. *BEST Viewpoints 7*, everything tuned for multi-core platforms.

Acknowledgements This research was carried out in the context of the MONDRIAN Project funded by the Portuguese Science and Technology Foundation (FCT) contract PTDC/EIA-CCO/108302/2008. Hugo Macedo holds a PhD FCT grant number SFRH/BD/33235/2007. Comments by Orlando Belo on an earlier version of this paper are gratefully acknowledged.

References

1. Bell, N., Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, pp. 18:1–18:11. ACM, New York, NY, USA (2009). DOI http://doi.acm.org/10.1145/1654059.1654078. URL `http://doi.acm.org/10.1145/1654059.1654078`
2. Bird, R., de Moor, O.: Algebra of Programming. Series in Computer Science. Prentice-Hall International (1997). C.A.R. Hoare, series editor
3. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Rec. **26**, 65–74 (1997). DOI http://doi.acm.org/10.1145/248603.248616. URL `http://doi.acm.org/10.1145/248603.248616`
4. Codd, E.: A relational model of data for large shared data banks. CACM **13**(6), 377–387 (1970)
5. Datta, A., Thomas, H.: The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. Decis. Support Syst. **27**(3), 289–301 (1999). DOI http://dx.doi.org/10.1016/S0167-9236(99)00052-4
6. Eavis, T., Dimitrov, G., Dimitrov, I., Cueva, D., Lopez, A., Taleb, A.: Parallel OLAP with the Sidera server. Future Generation Computer Systems **26**(2), 259 – 266 (2010). DOI DOI:10.1016/j.future.2008.10.007. URL `http://www.sciencedirect.com/science/article/B6V06-4TS6SF9-2/2/e47655496e565ae368b2c2d460262e1f`
7. Fokkinga, M.: Mapreduce — a two-page explanation for laymen (2008). URL `http://www.cs.utwente.nl/~fokkinga/mmf2008j.pdf`. Unpublished Technical Report
8. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
9. Freyd, P., Scedrov, A.: Categories, Allegories, *Mathematical Library*, vol. 39. North-Holland (1990)
10. Goil, S., Choudhary, A.: High performance olap and data mining on parallel computers. Data Mining and Knowledge Discovery **1**, 391–417 (1997). URL `http://dx.doi.org/10.1023/A:1009777418785`. 10.1023/A:1009777418785
11. Goil, S., Choudhary, A.: Parsimony: An infrastructure for parallel multidimensional analysis and data mining. Journal of Parallel and Distributed Computing **61**(3), 285 – 321 (2001). DOI DOI:10.1006/jpdc.2000.1691. URL `http://www.sciencedirect.com/science/article/B6WKJ-457CHXT-3G/2/24e539f3a0fa852750073235beabd01a`

[9] *Mathematica* TM is a registered trademark and Mathematica Player is a trademark of Wolfram Research, Inc.

12. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: S.Y.W. Su (ed.) Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana, pp. 152–159. IEEE Computer Society (1996)
13. Jensen, C.S., Pedersen, T.B., Thomsen, C.: Multidimensional Databases and Data Warehousing. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2010)
14. Jones, S.P.: Haskell 98 Language and Libraries. Cambridge University Press, Cambridge, UK (2003). Also published as a Special Issue of the Journal of Functional Programming, 13(1) Jan. 2003.
15. Lämmel, R.: Google's mapreduce programming model — revisited. Sci. Comput. Program. **68**(3), 208–237 (2007). DOI http://dx.doi.org/10.1016/j.scico.2007.07.001
16. Macedo, H., Oliveira, J.: Matrices As Arrows! A Biproduct Approach to Typed Linear Algebra. In: Mathematics of Program Construction, *Lecture Notes in Computer Science*, vol. 6120, pp. 271–287. Springer (2010)
17. Maier, D.: The Theory of Relational Databases. Computer Science Press (1983)
18. Meijer, E., Bierman, G.: A co-relational model of data for large shared data banks. Queue **9**, 30:30–30:48 (2011). DOI http://doi.acm.org/10.1145/1952746.1961297. URL `http://doi.acm.org/10.1145/1952746.1961297`
19. Ng, R.T., Wagner, A., Yin, Y.: Iceberg-cube computation with PC clusters. SIGMOD Rec. **30**, 25–36 (2001). DOI http://doi.acm.org/10.1145/376284.375666. URL `http://doi.acm.org/10.1145/376284.375666`
20. Oliveira, J.: Functional dependency theory made 'simpler'. Technical Report DI-PURe-05.01.01, DI/CCTC, University of Minho, Braga (2005). PUReCafe, 2005.01.18 [talk]; available from `http://wiki.di.uminho.pt/twiki/bin/view/Research/PURe/PUReCafe`
21. Oliveira, J.: Transforming Data by Calculation. In: GTTSE'07, *LNCS*, vol. 5235, pp. 134–195. Springer (2008)
22. Oliveira, J.: Pointfree foundations for (generic) lossless decomposition (2009). (Submitted to MSCS)
23. O'Neil, P.: Model 204 architecture and performance. In: D. Gawlick, M. Haynie, A. Reuter (eds.) High Performance Transaction Systems, *Lecture Notes in Computer Science*, vol. 359, pp. 39–59. Springer Berlin / Heidelberg (1989)
24. Park, C.S., Kim, M.H., Lee, Y.J.: Finding an efficient rewriting of OLAP queries using materialized views in data warehouses. Decision Support Systems **32**(4), 379 – 399 (2002). DOI DOI:10.1016/S0167-9236(01)00123-3. URL `http://www.sciencedirect.com/science/article/B6V8S-44M2FGN-1/2/2b2bb0f96ec0f2bddd0e3642fd7cdb42`
25. Pedersen, T.B., Jensen, C.S.: Multidimensional database technology. Computer **34**, 40–46 (2001). DOI http://dx.doi.org/10.1109/2.970558. URL `http://dx.doi.org/10.1109/2.970558`
26. Püschel, M.: Can we teach computers to write fast libraries? In: GPCE '07: Proceedings of the 6th international conference on Generative programming and component engineering, pp. 1–2. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1289971.1289973
27. Püschel, M., Moura, J.M.F., Johnson, J., Padua, D., Veloso, M., Singer, B., Xiong, J., Franchetti, F., Gacic, A., Voronenko, Y., Chen, K., Johnson, R.W., Rizzolo, N.: SPIRAL: Code generation for DSP transforms. Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation" **93**(2), 232– 275 (2005)
28. Rao, C., Rao, M.: Matrix algebra and its applications to statistics and econometrics. World Scientific Pub Co Inc (1998)
29. Solomonik, E., Demmel, J.: Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. Tech. Rep. UCB/EECS-2011-10, University of California at Berkeley (2011)
30. Sun, J., Tao, D., Faloutsos, C.: Beyond streams and graphs: dynamic tensor analysis. In: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 374–383. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1150402.1150445
31. Sun, J., Tao, D., Papadimitriou, S., Yu, P.S., Faloutsos, C.: Incremental tensor analysis: Theory and applications. ACM Trans. Knowl. Discov. Data **2**, 11:1–11:37

(2008). DOI http://doi.acm.org/10.1145/1409620.1409621. URL http://doi.acm.org/10.1145/1409620.1409621
32. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. Parallel Comput. **35**, 178–194 (2009). DOI 10.1016/j.parco.2008.12.006. URL http://portal.acm.org/citation.cfm?id=1513001.1513318
33. Wolfram, S., et al.: Mathematica: a system for doing mathematics by computer. Addison-Wesley New York (1988)
34. Wu, K., Otoo, E.J., Shoshani, A.: Optimizing bitmap indices with efficient compression. ACM Trans. Database Syst. **31**, 1–38 (2006). DOI http://doi.acm.org/10.1145/1132863.1132864. URL http://doi.acm.org/10.1145/1132863.1132864
35. Yang, G., Jin, R., Agrawal, G.: Implementing data cube construction using a cluster middleware: Algorithms, implementation experience, and performance evaluation. In: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID '02, pp. 84–. IEEE Computer Society, Washington, DC, USA (2002). URL http://portal.acm.org/citation.cfm?id=872748.873269
36. Yang, X., Parthasarathy, S., Sadayappan, P.: Fast sparse matrix-vector multiplication on GPUs: implications for graph mining. Proc. VLDB Endow. **4**, 231–242 (2011). URL http://portal.acm.org/citation.cfm?id=1938545.1938548

A Listings of Matlab scripts

Listing 1 provides both the calculation of cross tabulation for projections A, B and measure M and the generation of a bang matrix for size r.

```
function R = bang(r)
        R = ones(1,r);
end

function B = ctab(A,M,B)
    [m, k] = size(M);
    B = [ A ; bang(m) ] * M * [ B ; bang(k)]';
end
```

Listing 1 MATLAB encodings of *bang* (!) and cross table calculation (*ctab*).

The script presented in Listing 2 consists of a loop that builds a row partitioned matrix, as prescribed by the $\ominus$ operator and its indices. It iterates over the result sets generated by the powerset construction 2^D, corresponding to $P(proj)$ in the script.

We observe that a richer type system along the lines proposed by [16] would be useful by letting MATLAB to infer $(size(meas, 1))$ in the penultimate line, or even eliminate its need if in the construction of the partitioned matrix $[C; kr(psj)]$ a bang of size 6 were inferred as the empty Khatri-Rao matching the type.

The final result relies on a Khatri-Rao product implementation kr made available in the MATLAB Central repository by Laurent Sorber from K.U. Leuven. The script *CleanAndShow* removes 0 valued entries and shows the types of the result to increase the visualization of the result.

Although useful to put our ideas to work and as a validation of our approach, we have to disclaim and acknowledge the limitations of this script that was handcrafted to capture this particular example, but easily generalizable. Our intention is to exemplify and make it a hands-on resource to aid in the understanding of the approach.

The function encoding the powerset construction for the example data cube is presented in Listing 3. It generates the powerset from the bigger sets with three components to the smaller ones. The empty set is not included because it is hard coded in the `Cube.m` script.

```
function C = Cube(proj,meas)
    C = [];
    ps = P(proj);
    for j=1:size(ps,2)-1
        C = [C ; kr(ps{j})];
    end
    C = [C; bang(size(meas,1))];
    C = C * meas * bang(size(meas,1))';
end
```

Listing 2 MATLAB encodings of the Cube operator.

```
function PS = P(set)

    length = size(set,2);
    PS = cell(1,2^length);

    PS = {};
    x = 1;
    for i=length:-1:0

        ind = nchoosek(1:length,i);

        for j=1:size(ind,1)
            PS{x} = set(ind(j,:));
            x=x+1;
        end
    end


end
```

Listing 3 MATLAB encodings a powerset operator.

Listing 4 gives the script that labels the resulting vector of using `Cube.m` to the running example presented. It contains the labels hardcoded, joins them with the resulting vector and in the end iterates over the result by removing zero sale lines.

```
function column = CleanAndShow(totals)

    labels = ['Chevy␣1990␣Blue␣␣',
              'Chevy␣1990␣Green␣',
              'Chevy␣1990␣Red␣␣␣',
              'Chevy␣1991␣Blue␣␣',
              'Chevy␣1991␣Green␣',
              'Chevy␣1991␣Red␣␣␣',
              'Ford␣␣1990␣Blue␣␣',
              'Ford␣␣1990␣Green␣',
              'Ford␣␣1990␣Red␣␣␣',
              'Ford␣␣1991␣Blue␣␣',
              'Ford␣␣1991␣Green␣',
              'Ford␣␣1991␣Red␣␣␣',
              'Chevy␣1990␣ALL␣␣␣',
              'Chevy␣1991␣ALL␣␣␣',
              'Ford␣␣1990␣ALL␣␣␣',
              'Ford␣␣1991␣ALL␣␣␣',
              'Chevy␣ALL␣␣Blue␣␣',
              'Chevy␣ALL␣␣Green␣',
              'Chevy␣ALL␣␣Red␣␣␣',
              'Ford␣␣ALL␣␣Blue␣␣',
              'Ford␣␣ALL␣␣Green␣',
              'Ford␣␣ALL␣␣Red␣␣␣',
              'ALL␣␣␣1990␣Blue␣␣',
              'ALL␣␣␣1990␣Green␣',
              'ALL␣␣␣1990␣Red␣␣␣',
              'ALL␣␣␣1991␣Blue␣␣',
              'ALL␣␣␣1991␣Green␣',
              'ALL␣␣␣1991␣Red␣␣␣',
              'Chevy␣ALL␣␣ALL␣␣␣',
              'Ford␣␣ALL␣␣ALL␣␣␣',
              'ALL␣␣␣1990␣ALL␣␣␣',
              'ALL␣␣␣1991␣ALL␣␣␣',
              'ALL␣␣␣ALL␣␣Blue␣␣',
              'ALL␣␣␣ALL␣␣Green␣',
              'ALL␣␣␣ALL␣␣Red␣␣␣',
              'ALL␣␣␣ALL␣␣ALL␣␣␣'
              ];
    columnaux = [labels num2str(totals)];

    column = [];
    for i=1:size(columnaux,1)
        if(strcmp(columnaux(i,19:20),'␣0') == 0)
            column = [column; columnaux(i,:)];
        end
    end
end
```

Listing 4 MATLAB encodings of the CleanAndShow operator.