Can we teach computers to generate fast OLAP code?

(Research Note, May 2010)

Version of July 20, 2010

Hugo Daniel Macedo[1] and José Nuno Oliveira[1]

High-Assurance Software Lab (HASLab)
CCTC - Minho University, Portugal
{hmacedo,jno}@di.uminho.pt

Abstract. Inspired by previous pointfree relational approaches to data processing, we investigate how the generation of pivot tables in data mining (such as those produced by Microsoft Excel) can be expressed using matrix multiplication and transposition. This generalizes relational projections and provides a linear algebra approach to the drill-down/roll-up operations typical of OLAP. In this context, the prospect of using SPIRAL to generate fast OLAP code is analysed.

1 Introduction

This research note finds its motivation in the need to generate fast running code for data mining and other database analysis techniques such as OLAP (On-line Analytical Processing) [2]. These techniques are very useful for summarizing huge amounts of information in the form of *pivot tables* 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, but also the laptop spreadsheet user who wants to make sense of the data stored in a particular worksheet.

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

In areas such as digital signal processing (and linear algebra applications in general), generation of fast (parallel) code has witnessed great advances in recent years under the motto *"can we teach computers to write fast libraries?"* [11]. Domain specific languages (DSLs) and systems such as (respectively) SPL and SPIRAL [12], for instance, have shown how automatic generation of high performance libraries for linear algebra 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.

Parallelism blends well with functional programming, a discipline in which the construction of divide-and-conquer algorithms is the natural way to write programs. Functional programming too has witnessed great advances all over the years in many respects, namely in the development of an algebra of programming (AoP) [1] which puts emphasis on the "type structure" which is central to modern functional languages such as Haskell, for instance [4].

In a recent paper [5], we have shown how close to the AoP a *"matrices as arrows"* (typed!) approach to linear algebra is, easy to understand after all since functions are special cases of binary relations which in turn are nothing but Boolean matrices [1]. Elsewhere, one of us has shown how to take advantage of binary relation algebra in reasoning about data dependencies in databases [7, 9] and data transformation in general [8].

Given this proximity between relation and matrix algebra, the question arises: how much gain can we expect from translating results from one side to the other? In this note we show how a particular construction in relation algebra — that of building binary *relational projections*, used in [7, 9] to calculate with functional dependencies in databases — translates to building *pivot tables* which are central to OLAP and data-mining. 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. The dot $(\cdot)$ between the symbols denotes relational composition 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), A and B are attributes of the schema of T, f_A (resp. f_B) means the function which captures the semantics of attribute A (resp. B), and $[\![T]\!]$ captures the semantics of T in the form of a binary relation known as a *coreflexive* [1]:

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

[1] Indeed, relation algebra and matrix algebra can be regarded as instances of the *allegory* concept [3], the latter under some restrictions on the algebra of matrix elements.

[2] For instance, for $R = id$, (1) expresses a *tabulation* [3, 1] wherever the paring of f and g is injective. For functions f and g inductive on the same input type (such functions are known as *catamorphisms* [1]), expression $f \cdot g^{\circ}$ even captures the "divide-and-conquer" algorithm whose divide step is carried out by g° and the conquer step by f.

However strange (and redundant!) this construction may look like, it proves essential to the reasoning, as shown in [7, 9].

Essential to (2) is its emphasis on the very basic combinators of relation algebra, composition and converse. These translate to matrix multiplication and transposition, respectively, which are easy to parallelize by code generating systems such as SPIRAL. Under this motivation, we show below that the construction of *pivot tables* in data mining and OLAP can be expressed by a formula similar to (2),

$$t_A \cdot [\![T]\!] \cdot t_B^\circ \tag{3}$$

paving the way to fast generation of such tables, where A and B are the attributes (dimensions) of the table to build and $[\![T]\!]$ is the diagonal matrix capturing the attribute where calculations take place. The construction of matrices t_A and t_B is detailed below with an example. Tables are pictured as generated by Microsoft Excel.

About this document. We warn the reader that the ideas jotted in this note have not yet been checked against related work. Anyone knowing about similar applications of linear algebra to OLAP and data-mining, please report to the authors.

2 Calculating Pivot Tables

In data processing, a 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 (adapted from [14]) where each row records the number of units of a given product shipped to costumers in a given region at a specific shipping date. Other attributes of each shipping record in the table include price, cost and the gender and style classification of the corresponding product.

Region	Gender	Style	Ship Date	Season	Units	Price	Cost
East	Boy	Tee	01/31/05	Winter	12	11.04	10.42
East	Boy	Golf	01/31/05	Winter	12	13	12.6
East	Boy	Fancy	01/31/05	Winter	12	11.96	11.74
East	Girl	Tee	03/20/05	Spring	10	11.27	10.56
East	Girl	Golf	03/20/05	Spring	10	12.12	11.95
East	Girl	Fancy	03/20/05	Spring	10	13.74	13.33
West	Boy	Tee	07/31/05	Summer	11	11.44	10.94
West	Boy	Golf	07/31/05	Summer	11	12.63	11.73
West	Boy	Fancy	07/31/05	Summer	11	12.06	11.51
West	Girl	Tee	07/31/05	Summer	15	13.42	13.29
West	Girl	Golf	10/25/05	Autumn	15	11.48	10.67
North	Boy	Golf	10/25/05	Autumn	16	11.34	11.24
South	Boy	Golf	10/25/05	Autumn	17	10.35	11.11

Fig. 1. Collection of raw data

In general, the raw-data out of which pivot tables are built is not normalized and arises by collecting into a central database (termed a data *warehouse*, or

decision support database) huge amounts of information obtained from disparate databases. Such a central warehouse — typically, a table with an absurd number of lines — is not easy (if at all possible) to 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 units were shipped per region and ship date?"*, leading to the corresponding *pivot table*. For this particular question, the attributes *Region* and *ShipDate* are selected as dimensions of interest. The corresponding pivot table, as generated by Excel, as depicted in Figure 2.

Sum of Units	Ship Date				
Region	01/31/05	07/31/05	10/25/05	03/20/05	Grand Total
East	36			30	66
North			16		16
South			17		17
West		48	15		63
Grand Total	36	48	48	30	162

Fig. 2. A pivoted table

Pivot table generation is 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 [10]. As mentioned in [2], numerous SQL extensions are offered by many vendors of OLAP products trying to address this problem. The solution we put forward in this note 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.

The key resides in expressing OLAP operations in terms of matrix algebra expressions which can then be parallelized using tools such as SPIRAL [12]. In the particular case of reporting multi-dimensional analysis of data in the form of pivot tables, we have to be able to build, according to the hint given by formula (3), three matrices: two associated to the dimensions (attributes) A and B being analysed and the other recording which *measure* or *metrics* of T is to be considered.

As an example, let us now see how to generate the pivot table of Figure 2 using matrices.

Building projection functions. Let A be an attribute of raw-data table T and $|A|$ denote the range of values which can be found in column A of T. Let n be the number of records in T (rows, or lines in a spreadsheet).

Note that each column in T "is" a function which tells, for each row, which value of $|A|$ can be found in such column. Such a function can be encoded as an

elementary matrix of type [3] $n \xrightarrow{t_A} |A|$, defined as follows:

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

In our running example, $n = 13$, $A = Region$ and $B = ShipDate$. Figures 3 and 4 depict projection matrices t_{Region} and $t_{ShipDate}$, respectively.

	1	2	3	4	5	6	7	8	9	10	11	12	13
East	1	1	1	1	1	1	0	0	0	0	0	0	0
West	0	0	0	0	0	0	1	1	1	1	1	0	0
North	0	0	0	0	0	0	0	0	0	0	0	1	0
South	0	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 3. *Region* projection

	1	2	3	4	5	6	7	8	9	10	11	12	13
01/31/05	1	1	1	0	0	0	0	0	0	0	0	0	0
03/28/05	0	0	0	1	1	1	0	0	0	0	0	0	0
07/31/05	0	0	0	0	0	0	1	1	1	1	0	0	0
10/25/05	0	0	0	0	0	0	0	0	0	0	1	1	1

Fig. 4. *ShipDate* projection

Note that, typewise, the composition of matrices $|Region| \xleftarrow{t_{Region}} n$ and $n \xleftarrow{t^{\circ}_{ShipDate}} |ShipDate|$ already makes sense, leading to the matrix depicted in Figure 5, which essentially counts the number of shipping records per region and date. This situation (*counting*), which is what Excel outputs wherever the third attribute chosen is not numeric, corresponds to formula (3) where the middle matrix is the identity.

	01/31/05	03/20/05	07/31/05	10/25/05
East	3	3	0	0
West	0	0	4	1
North	0	0	0	1
South	0	0	0	1

Fig. 5. Matrix $t_{Region} \cdot t^{\circ}_{ShipDate}$ (counting)

In order to sum up the number of units shipped rather than just counting shippings we need to supply a numeric attribute of T which will be used for

[3] Hereafter we stick to the arrow notation of [5] in typing matrices.

consolidation. In the case of T (Figure 1) any of $Unit$, $Price$ and $Cost$ apply. Because such numeric data has to become available for both projection matrices, the column chosen is converted into a diagonal matrix.

The diagonal construction. Let $T(A)$ denote the column of raw-data table T identified by attribute A and $T(y, A)$ denote the element occupying the y-th position in such column. The conversion of column $T(Unit)$ into the corresponding diagonal matrix of type $n \prec\!\!\longleftarrow n$ is a basic construction:

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

Matrix $[\![T(Unit)]\!]$ is given in Figure 6 and the outcome of pivot table calculation using matrix operations

$$t_{Region} \cdot [\![T(Unit)]\!] \cdot t^{\circ}_{ShipDate} \tag{4}$$

is given in Figure 7.

12	0	0	0	0	0	0	0	0	0	0	0	0
0	12	0	0	0	0	0	0	0	0	0	0	0
0	0	12	0	0	0	0	0	0	0	0	0	0
0	0	0	10	0	0	0	0	0	0	0	0	0
0	0	0	0	10	0	0	0	0	0	0	0	0
0	0	0	0	0	10	0	0	0	0	0	0	0
0	0	0	0	0	0	11	0	0	0	0	0	0
0	0	0	0	0	0	0	11	0	0	0	0	0
0	0	0	0	0	0	0	0	11	0	0	0	0
0	0	0	0	0	0	0	0	0	15	0	0	0
0	0	0	0	0	0	0	0	0	0	15	0	0
0	0	0	0	0	0	0	0	0	0	0	16	0
0	0	0	0	0	0	0	0	0	0	0	0	17

Fig. 6. Diagonal matrix recording metric column $Unit$

	01/31/05	03/20/05	07/31/05	10/25/05
East	36	30	0	0
West	0	0	48	15
North	0	0	0	16
South	0	0	0	17

Fig. 7. Pivot table calculated by matrix expression (4)

Grand totals. If compared to Figure 2, the table of Figure 7 misses the two row and column grand totals. These are very easily obtained via *"bang" matrices*. We explain what these are and our choice of terminology: in functional programing, the (popular) "bang" function, which is of type $A \to 1$ and usually denoted by

symbol "!", is a polymorphic constant function yielding the unique value which inhabits the singleton type 1. (In Haskell, both this type and its inhabitant are denoted by "()".) The encoding of this function in matricial form will be matrix $1 \xleftarrow{!_A} A$ wholly filled up with 1s. For instance, $!_{|Region|}$ will be the row vector with $|Region|$-many positions all holding number 1.

Clearly, post-composing our pivot table with $!_{|Region|}$ will yield the lower grand-total row of Figure 2 and pre-composing the same table with the converse (transpose) of $!_{|ShipDate|}$ will yield the column grand total on the right. The bottom-right cell holding number 162 is nothing but the singleton matrix

$$1 \xleftarrow{!_n \cdot [\![T(unit)]\!] \cdot !_n{}^\circ} 1$$

3 "Rolling-up" on functional dependencies

The pre- and post-composition of a given pivot table with the "bang matrices" of the previous section is already an example of the OLAP operation known as *roll-up*.

Rolling-up means replacing a dimension of a pivot table by another which is more general, in some sense (eg. 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) [6] in raw-data. Let us, for instance, look at the column labelled *Season* in table T (Figure 1), telling in which season (*Spring*, *Summer*, *Autumn* or $Winter$) a particular shipping took place. FD $Season \leftarrow ShipDate$ clearly holds, as every date occurs in one and only one season. In other words, *Season* is higher in the dimension hierarchy than *ShipDate* [4].

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 pivot table involving A can be *rolled-up* to another (less detailed) involving B instead.

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

$$t_{B \leftarrow A} = t_B \cdot t_A \tag{5}$$

(We hope (5) convinces the reader of the advantage of writing FDs the other way round, namely $B \leftarrow A$ instead of the more conventional $A \rightarrow B$ [6].) Figure 8 depicts the roll-up matrix calculated from FD $Season \leftarrow ShipDate$ [5].

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

[5] Notice that Figure 8 in fact "depicts a function". More generally, construction (5) enlables us to check for functional dependencies: FD $B \leftarrow A$ will hold wherever

	01/31/05	03/20/05	07/31/05	10/25/05
Autumn				1
Spring		1		
Summer			1	
Winter	1			

Fig. 8. Roll-up matrix $t_{Season \leftarrow ShipDate}$

So, given a pivot matrix $|A| \prec\!\!\xleftarrow{M} |C|$, the effect of rolling it up across a given FD $B \leftarrow A$ is pivot matrix

$$t_{B \leftarrow A} \cdot M$$

of type $|B| \prec\!\!\!\!- |C|$. Converse (transpose) caters for the same effect on the right-hand side: rolling M up across another FD $D \leftarrow C$ is pivot matrix

$$M \cdot t^{\circ}_{D \leftarrow C}$$

Further developments. The matrix representation of FDs opens further perspectives on the *roll-up* OLAP operation, as the matrix in Figure 9 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.

	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

Fig. 9. Seasons as "fuzzy" sets of months

Perhaps one might say that a "fuzzy" data dependency holds, in this case. 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 composition and transposition.

4 Related Work

As written above, thus far we have not done a proper inspection of the literature on linear algebra application to data-mining and OLAP. Being novices in OLAP

matrix $t_{B \leftarrow A}$ is *simple*, a terminology imported from relational algebra and allegory theory[3] — a $\mathbb{N}_0$-valued matrix S will be said to be simple iff its *image* $S \cdot S^{\circ}$ is diagonal.

technology, we are still unable to evaluate the novelty of the ideas jotted in this note, whose single purpose is to gather feedback on their opportunity as far as the parallelization of OLAP code is concerned.

The prospect of extending such techniques to spreadsheet software running on multi-core laptops is also of interest [6].

Indeed, there may be work around which we don't know about. In a quick search using Google, a particular reference has attracted our attention already: in [13] Jimeng Sung and others 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.

The paper generalizes the technique of (PCA) Principal Component Analysis, which reduces the number of elements of the dimensions. The typical example is the factorization of a matrix into two "smaller" matrices, whose multiplication is the matrix which has the minimal difference to the original according to a least squares distance.

The generalized technique is termed DTA (Dynamic Tensor Analysis) and a stream version of it STA: *"Intuitively, it projects and matricizes along each mode; and it performs PCA to find the projection matrix for that mode."* Here mode refers to a way of (matricizing) transforming a tensor into a matrix.

On the background stays *singular value decomposition* (SVD), whose matricial expression conspicuously resembles our starting point (3). Thus we have something to study, already.

5 Future work

Our intention thus far has been to illustrate how the kinship between relation and matrix algebra can suggest ways of expressing resource intensive data processing such as OLAP, using linear algebra kernels of the kind systems such as SPIRAL handle in a very efficient way.

On the practical side, our main expectations reside in checking whether SPIRAL can be used in areas other than DSP, in this case in data-mining and OLAP, based on the strategy outlined in this note.

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 [2, 10] and very 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.

[6] 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`.

Anyone wishing to provide feedback on our sketchy ideas is very welcome.

References

1. R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997. C.A.R. Hoare, series editor.
2. Anindya Datta and Helen Thomas. 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.
3. P.J. Freyd and A. Scedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.
4. S.L. Peyton Jones. *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.
5. H.D. Macedo and J.N. Oliveira. Matrices as arrows! a biproduct approach to typed linear algebra, 2010. Accepted by MPC'10.
6. D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.
7. J.N. Oliveira. Functional dependency theory made 'simpler'. Technical Report DI-PURe-05.01.01, DI/CCTC, University of Minho, Gualtar Campus, Braga, 2005. PUReCafé, 2005.01.18 [talk]; available from `http://wiki.di.uminho.pt/twiki/bin/view/Research/PURe/PUReCafe`.
8. J.N. Oliveira. Transforming Data by Calculation. In *GTTSE'07*, volume 5235 of *LNCS*, pages 134–195. Springer, 2008.
9. J.N. Oliveira. Pointfree foundations for (generic) lossless decomposition, 2009. (Submitted to MSCS).
10. Chang-Sup Park, Myoung Ho Kim, and Yoon-Joon Lee. Finding an efficient rewriting of olap queries using materialized views in data warehouses. *Decision Support Systems*, 32(4):379 – 399, 2002.
11. Markus Püschel. Can we teach computers to write fast libraries? In *GPCE '07: Proceedings of the 6th international conference on Generative programming and component engineering*, pages 1–2, New York, NY, USA, 2007. ACM.
12. Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nicholas Rizzolo. SPIRAL: Code generation for DSP transforms. *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, 93(2):232– 275, 2005.
13. Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383, New York, NY, USA, 2006. ACM.
14. Wikipedia. Pivot table — wikipedia, the free encyclopedia, 2010. [Online; accessed 12-May-2010].