Optimisation of a Linear Algebra Approach to OLAP

Filipe Oliveira - A57816
a57816@alunos.uminho.pt

Sérgio Caldas - A57779
a57779@alunos.uminho.pt

July 20, 2016

Abstract

Online Analytical processing (OLAP) systems, perform multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling. All the referred analysis depends on Relational Algebra, which lack algebraic properties, and qualitative and quantitative proofs for all the relational operator. The proposed solution focus on a typed linear algebra approach, encoding OLAP functionality solely in terms of Linear Algebra operations (matrices).

*It has been argued that linear algebra (LA) is better suited than standard relational algebra for formalizing and implementing queries in on-line multidimensional data analysis [**?**] [**?**]. This can be achieved over a small LA sparse matrix kernel which, further to multiplication and transposition, offers the Kronecker, Khatri-Rao and Hadamard products. We give preliminary experimental results obtained with one cluster of Search6(Ref) using queries of the TPC-H Benchmark(Ref).*

I. Introduction

The design and development of systems that generate, collect, store, process, analyse, and query large sets of data is filled with significant challenges both hardware and software. Combined, these challenges represent a difficult landscape for software engineers.

The relation database is the current solution for big data storage. Data dependencies in databases can be seen as a binary two-way associative relation. Regarding that lema, prior efforts have been made [**?**] [**?**] in the research project "Linear Algebra approach to OLAP", in order to fully represent relational algebra in terms of linear algebra operators.

OLAP is resource-demanding and calls for parallelisation. Regarding the challenge of High Performance Computing, we implemented from the start a typed linear algebra solution given special importance to the data modelling which, if done poorly, limits the attainable efficiency in data-intesive systems like OLAP that tends to access massive amounts of data and is thus time consuming.

With respect to performance evaluation and results validation in a real work scenario, the used datasets were produced with TPC-H Benchmark, in which is workload consists of multiple query runs. In order to obtain realistic and meaningful results large datasets were considered, ranging from 1 to 64GB.

In order to infer conclusions and compare relational and linear algebra the object-relational database management system PostgreSQL version 9.6, with roots in open source community, was chosen in order to represent the relational algebra approach.

Given this proximity between database relations and linear algebra, the question arises: does the linear algebra approach presents performance improvements when compared with the relational one?

The report is organised as follows. Section 2 introduces XXXX. Section 3 presents XXXX. Section 4 gives experimental results. Section 5 presents related work. Section 6 concludes.

II. Linear Algebra Strategy

I. Towards a linear algebra semantics for SQL

Inspired by point-free relational data processing, an alternative roadmap for parallel online analytical processing (OLAP) can be achieved based on encoding data in matrix format and relying thereupon solely on LA operations[**?**].

I.1 Encoding data in matrix format

As example of raw data consider the displayed table 1 where each row records the order key, quantity, return flag, line status and ship date from an given TPC-H benchmark lineitem table. In order to facilitate data association the column number in the corresponding lineitem table was included above each data column.

Table 1: *Collection of raw data (adapted from TPC-H benchmark lineitem table).*

#1 l_orderkey	#5 l_quantity	#9 l_returnflag	#10 l_linestatus	#11 l_shipdate
1	17	N	O	1996-03-13
1	36	N	O	1996-04-12
1	8	N	O	1996-01-29
1	28	N	O	1996-04-21
1	24	N	O	1996-03-30
1	32	N	O	1996-01-30
2	38	N	O	1997-01-28
3	45	R	F	1994-02-02
3	49	R	F	1993-11-09
3	27	A	F	1994-01-16

To obtain useful information from raw data, which in OLAP systems escalated to Terabytes of information, we need to summarise the data by selecting attributes of interest and exhibiting their inter-relationships.

Consider the following simplified TPC-H query 1: "How many items were sold per return flag and line status?". For this particular question, the necessary attributes to answer the query are present on table lineitem, being **return flag**, **line status**, and **quantity**. In relational algebra that question could be easily answered by the following SQL code:

Listing 1: *SQL code for the simplified TPC-H query 1*

```
SELECT l_returnflag, l_linestatus,  sum(↩
    l_quantity)
FROM LINEITEM_SAMPLE
GROUP BY l_returnflag, l_linestatus;
```

which would produce the result presented on table 2

Table 2: *Simplified query-1 relational algebra result from the collection of raw data (adapted from TPC-H benchmark lineitem table).*

#9 l_returnflag	#10 l_linestatus	#5 l_quantity
N	O	183
R	F	94
A	F	27

Aggregations like the presented on the prior simplified query occur in all TPC-H queries, hence performance of group-by and aggregation is quite important. That matter will be addressed in the later sections of the report.

Regarding expressing the OLAP in therms of LA, the key resides in expressing operations in the form of matrix algebra expressions. In this particular example, we should be able to build three matrices, one for each attribute, with each matrix being correlated to relational algebra as the row storage of columns #5, #9, and #10. However, in order to do so, we need to find a two-way association between the presented string on the rows #9 and #10 and an unique integer identifier. Our proposed solution encodes strings recurring to Gnome **Gquarks [?]** - a two-way association between a non-zero unsigned int and a char * - based on a thread safe hashtable.

By order of appearance, each unique string will be associated to an unique unsigned integer. Repeated strings will be associated to the prior corresponding unsigned integer. The resulting unsigned integer value range will start in the number 1. The value 0 in GQuarks is associated to NULL. Since in the proposed solution row and column numbers will respect C-style arrays notation both column and row numbering will start on 0. The GQuark unsigned integer value decremented by 1 will represent the row position on the matrix, and the register number, starting at 0, will represent the column position on the matrix.

We can now verify that at most one non-zero cell can be found in each column the matrix in order to maintain the two-way association between an register and its corresponding string (matrix is "functional"). There is also the possibility of direct associating an register number with a value, wether integer or floating point. The referred matrices will be diagonal matrices, in which the element value is directly associated with the register.

Two types of matrices have now been presented:

1. Projection Matrices - which recur to Gnome Gquarks, in which the Gquark decremented by 1, represents the row position of the matrix, and the register number, starting at 0, will represent the column position on the matrix.

2. Measure Matrices - direct associating an register number with a value, wether integer or floating point, which the element value is directly associated with the register number, and consequently the column and row position.

We can now build the necessary matrices for the given examples, presented from figure 1 to 3. The two-way association between char* and unsigned integer is also presented in table 3 in order to aid the example comprehension.

Table 3: *Two-Way association between GQuarks and Row Number, for rows #9 and #10 presented in the collection of raw data (adapted from TPC-H benchmark lineitem table).*

String	GQuark	Row #
N	1	0
R	2	1
A	3	2
O	4	3
F	5	4

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
r0	17	0	0	0	0	0	0	0	0	0
r1	0	36	0	0	0	0	0	0	0	0
r2	0	0	8	0	0	0	0	0	0	0
r3	0	0	0	28	0	0	0	0	0	0
r4	0	0	0	0	24	0	0	0	0	0
r5	0	0	0	0	0	32	0	0	0	0
r6	0	0	0	0	0	0	38	0	0	0
r7	0	0	0	0	0	0	0	45	0	0
r8	0	0	0	0	0	0	0	0	49	0
r9	0	0	0	0	0	0	0	0	0	27

Figure 1: *Dense Measure Matrix produced from the collection of raw data (adapted from TPC-H benchmark lineitem table), from column #5 (quantity column).*

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
r0	1	1	1	1	1	1	1	0	0	0
r1	0	0	0	0	0	0	0	1	1	0
r2	0	0	0	0	0	0	0	0	0	1

Figure 2: *Dense Projection Matrix produced from the collection of raw data (adapted from TPC-H benchmark lineitem table), from column #9 (return flag column) with 2-way association achieved recurring to GQuarks.*

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
r0	0	0	0	0	0	0	0	0	0	0
r1	0	0	0	0	0	0	0	0	0	0
r2	0	0	0	0	0	0	0	0	0	0
r3	1	1	1	1	1	1	1	0	0	0
r4	0	0	0	0	0	0	0	1	1	1

Figure 3: *Dense Projection Matrix produced from the collection of raw data (adapted from TPC-H benchmark lineitem table), from column #10 (line status column) with 2-way association achieved recurring to GQuarks.*

II. Projection Operation

Regarding the projection statement of the example query:

Listing 2: *SQL code for the simplified TPC-H query 1 only for the SELECT and GROUP BY statements*

```
SELECT l_returnflag, l_linestatus
FROM LINEITEM_SAMPLE
GROUP BY l_returnflag, l_linestatus;
```

the typical case to reproduce a SELECT and GROUP BY statement on a relational algebra approach would be to remove certain columns of the lineitem table. In the LA approach the process consists of joining the projection matrices of the selected attributes. The final LA result should hold all possible combinations of the projected attributes and no duplicated tuples. That can be achieved by doing several Khatri-Rao products, one for each tuple of attributes present in the statement.

The corresponding LA encoding of the SQL statement from listing 2 is given by the equation presented in equation 1.

$$Projection\ Matrix\ =\ ReturnFlag\ \otimes LineStatus \tag{1}$$

which would produce the matrix presented in figure 4.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
r0	0	0	0	0	0	0	0	0	0	0
r1	0	0	0	0	0	0	0	0	0	0
r2	0	0	0	0	0	0	0	0	0	0
r3	1	1	1	1	1	1	1	0	0	0
r4	0	0	0	0	0	0	0	0	0	0
r5	0	0	0	0	0	0	0	0	0	0
r6	0	0	0	0	0	0	0	0	0	0
r7	0	0	0	0	0	0	0	0	0	0
r8	0	0	0	0	0	0	0	0	0	0
r9	0	0	0	0	0	0	0	1	1	0
r10	0	0	0	0	0	0	0	0	0	0
r11	0	0	0	0	0	0	0	0	0	0
r12	0	0	0	0	0	0	0	0	0	0
r13	0	0	0	0	0	0	0	0	0	0
r14	0	0	0	0	0	0	0	0	0	1

Figure 4: *Projection Matrix produced from the Khatri-Rao product between return flag and line status columns from lineitem table, from columns #9 and #10.*

Given the matrices A with dimensions $(m \times n)$ and B with dimensions $(o \times p)$, the produced projection matrix has dimensions $(q \times r)$, being $q = m = o$ and $r = m \times o$. The produced matrix dimensions can also be expressed as $(\ (\ m \times o\ ) \times n)$ as visible on figure 4.

In order to produce the desired tuples holding all possible combinations, the association between row number and tuple has to be created. Given two original matrices A with dimensions $(m \times n)$ and B with dimensions $(o \times p)$, and one produced projection matrix C via the Khatri-rao product, in order re-obtain both strings that produce each tuple, for every row R_C from the C matrix, the corresponding R_A from matrix A, and R_B from matrix B, are given by the expression:

$$\begin{aligned} R_A &= R_C / o \\ R_B &= R_C \,\%\, o \end{aligned} \tag{2}$$

We can now build the corresponding association between tuples and the rows of matrix C, as shown on table 4.

Table 4: *Association between the produced projection matrix from the Khatri-Rao product between return flag and line status columns from lineitem table, from columns #9 and #10, and the corresponding produced tuples for every row of matrix C.*

Row C	Column C	R_A	R_B	String A	String B	Tuple
r 3	0	0	3	N	O	(N , O)
r 3	1	0	3	N	O	(N , O)
r 3	2	0	3	N	O	(N , O)
r 3	3	0	3	N	O	(N , O)
r 3	4	0	3	N	O	(N , O)
r 3	5	0	3	N	O	(N , O)
r 3	6	0	3	N	O	(N , O)
r 9	7	1	4	R	F	(R , F)
r 9	8	1	4	R	F	(R , F)
r 14	9	2	4	A	F	(A , F)

III. Aggregation Operation

As you can state the SELECT SQL statement was reproduced, however the GROUP BY operation still needs to be applied. In order to do so we need to define an operator, a row vector commonly know as bang[**?**].

Given an matrix C with $(\ (\ m\ \times\ o\ )\ \times\ n)$, the bang vector, in order to be correctly typed, would have dimension n and be totally filled with the value 1. Figure 5 illustrates the produced bang vector to be multiplied by matrix C in order to group the elements located in the same row of matrix C.

$$\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \end{array} \begin{array}{c} c0 \\ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \end{array}$$

Figure 5: *Projection Matrix produced from the Khatri-Rao product between return flag and line status columns from lineitem table, from columns #9 and #10.*

The grouped LA encoding of the SQL statement from listing 2 is given by the equation 3:

$$Projection\ Vector\ =\ (\ ReturnFlag\ \otimes LineStatus\ ) \times bang \quad (3)$$

resulting in the projection vector presented in figure 6.

$$\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \\ r10 \\ r11 \\ r12 \\ r13 \\ r14 \end{array} \begin{array}{c} c0 \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{array}$$

Figure 6: *Projection Vector produced from the product between Khatri-Rao product shown on figure 4 and bang row vector show on figure 6.*

Considering the value 0 as the non existing relation between atributes we can simplify the given projection vector and produce the association between return flag and line status columns from lineitem table based on expression 2, as visible on table 5.

Table 5: *Association between the produced projection Vector from the Khatri-Rao product between return flag and line status columns from lineitem table, from columns #9 and #10, and the corresponding tuples.*

Row C #	R_A	R_B	String A	String B	Tuple
r 3	0	3	N	O	(N , O)
r 9	1	4	R	F	(R , F)
r 14	2	4	A	F	(A , F)

In order to produce the results show on listing 1 via the linear algebra approach, we need to associate to every distinct tuple the corresponding quantity. Expression 4 adds the quantity measure matrix to the LA encoding. All measure matrices, shall therefore be represented between double square brackets in order to assist differentiation from the projection matrix type.

$$((\ ReturnFlag\ \otimes LineStatus\ ) \times\ [\![Quantity]\!])\ \times\ bang \quad (4)$$

In addition to the association between row number and tuple, with the values 1 and 0 representing the existence or non-existence of the attributes association respectively, we now have a value associated to each tuple. Figure 7 illustrates the resultant matrix of the dot product between the Khatri-Rao product between return flag and line status columns, and the quantity measure matrix, from the given lineitem example table.

$$
\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \\ r10 \\ r11 \\ r12 \\ r13 \\ r14 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} c0 & c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
17 & 36 & 8 & 28 & 24 & 32 & 38 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 49 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 27
\end{pmatrix}
\end{array}
$$

Figure 7: *Resultant matrix of the dot product between the Khatri-Rao product between return flag and line status columns, and the quantity measure matrix, produced from the LA expression 4.*

The dot product between the resultant matrix defined in figure 7 and the row bang vector defined in figure 6 would produce the final row vector presented on figure 8.

$$
\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \\ r10 \\ r11 \\ r12 \\ r13 \\ r14 \end{array}
\begin{array}{c}
\begin{array}{c} c0 \end{array} \\
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 183 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 94 \\ 0 \\ 0 \\ 0 \\ 0 \\ 27
\end{pmatrix}
\end{array}
$$

Figure 8: *Final Row Vector produced from the product between Khatri-Rao product shown on figure 7 and bang row vector show on 6.*

We have now defined the necessary operations to reproduce through linear algebra the relational algebra results shown on table 2. Table 6 associates the generated tuples via the Khatri-Rao operation shown on figure 7 and the row vector bang as shown on expression 4.

Table 6: *Association between the generated tuples via the Khatri-Rao operation shown on figure 7 and the row vector bang as shown on expression 4.*

Row C #	R_A	R_B	String A	String B	Tuple	Value
r 3	0	3	N	O	(N , O)	183
r 9	1	4	R	F	(R , F)	94
r 14	2	4	A	F	(A , F)	27

IV. Selection Operation

Given the obtained results, there is still one linear operation required to fully respond to a new query, more close to the TPC-H query 1: "How many items were sold per return flag and line status, between the dates 1996-04-12 and 1997-01-28?". For this particular question, the necessary attributes to answer the query remain present on the table lineitem, being the pior analysed **return flag**, **line status**, and **quantity**, with the addition of the selection attribute – **shipdate**. In relational algebra that question could be easily answered with the extension of the prior SQL code presented on listing 1:

Listing 3: *SQL code for the simplified TPC-H query 1*

```
SELECT l_returnflag, l_linestatus,  sum(↩
    l_quantity)
FROM LINEITEM_SAMPLE
WHERE l_shipdate >= "1996-04-12" AND l_shipdate ↩
    <= "1997-01-28"
GROUP BY l_returnflag, l_linestatus;
```

which would produce the result presented on table 7:

Table 7: *Query-1 relational algebra result from the collection of raw data (adapted from TPC-H benchmark lineitem table).*

#9 l_returnflag	#10 l_linestatus	#5 l_quantity
N	O	102

The same result could be computed regarding solely on linear algebra operations strictly restricting the values present on the matrix shown on figure 7. By producing an measure selection matrix initially with its diagonal cells filled with value 1, and by re-obtaining the string based on the corresponding GQuark of the selection column, comparing it with both keys (**1996-04-12** and **1997-01-28**), we can simply replace the cells of the measure selection matrix that do not pass in the restriction with the value 0.

On subsection V the non validation of the comparison would result in the elimination of the element from the sparse representation, as we shall address later. For a matter of simplification of the visualisation process of the algorithm this method shall be used on this section.

Expression 5 adds the selection measure matrix to the LA encoding. We now have projection, selection, and aggregation operations defined solely in terms of linear algebra.

$$(((\ ReturnFlag \otimes LineStatus\ ) \times [\![Selection]\!]\ ) \times [\![Quantity]\!]) \times bang \qquad (5)$$

Figure 9 illustrates the resultant measure selection matrix of the restriction of shipdate column, from the given lineitem example table. In order to assist visualisation table 8 associates the row value of shipdate for every register with the result of the selection process.

Table 8: *Association between row value of shipdate for every register with the result of the selection process, for a restriction of values between **1996-04-12** and **1997-01-28**.*

Row #	l_shipdate	Value
r 0	1996-03-13	FALSE
r 1	1996-04-12	**TRUE**
r 2	1996-01-29	FALSE
r 3	1996-04-21	**TRUE**
r 4	1996-03-30	FALSE
r 5	1996-01-30	FALSE
r 6	1997-01-28	**TRUE**
r 7	1994-02-02	FALSE
r 8	1993-11-09	FALSE
r 9	1994-01-16	FALSE

$$\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \end{array} \begin{array}{c} \begin{array}{cccccccccc} c0 & c1 & c2 & c3 & c4 & c5 & c6 & c7 & c8 & c9 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 9: *Measure Selection Matrix produced from the restriction of shipdate column to values between **1996-04-12** and **1997-01-28**, from the given lineitem example table, from column #11.*

Denote that the selection process could occur both in the projection or the aggregation process with the restriction to be realised before the bang operation. In order to otimize the operations the selection should be realized in the initial stages of the query, since it reduces the amount of computation required later.

Reproducing the LA operations presented on expression 5 the product between Khatri-Rao product shown on figure 7, the measure selection matrix define in figure 9, the measure quantity matrix define in figure 1, and the row bang vector defined in figure 6 would produce the final row vector presented on figure 10.

$$\begin{array}{c} \\ r0 \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \\ r6 \\ r7 \\ r8 \\ r9 \\ r10 \\ r11 \\ r12 \\ r13 \\ r14 \end{array} \begin{array}{c} \begin{array}{c} c0 \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 102 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{array}$$

Figure 10: *Final Row Vector produced from the product between Khatri-Rao product shown on figure 7, the measure selection matrix define in figure 9, the measure quantity matrix define in figure 1, and the row bang vector defined in figure 6.*

We have now defined the necessary operations to reproduce through linear algebra the relational algebra results shown on table 7. Table 9 associates the generated selected tuples via expression 5.

Table 9: *Association between the generated selected tuples via expression 5.*

Row C #	R_A	R_B	String A	String B	Tuple	Value
r 3	0	3	N	O	(N , O)	102

V. Encoding data in Sparse Matrix Format

The matrices used to encode OLAP information will state an high degree of both sparsity and irregularity. Regarding the minor dataset used in the experimental results, figures 11 trough 14, analyse the sparsity pattern of the necessary columns in order to respond to the simplified TPC-H query-1 show on expression 5.

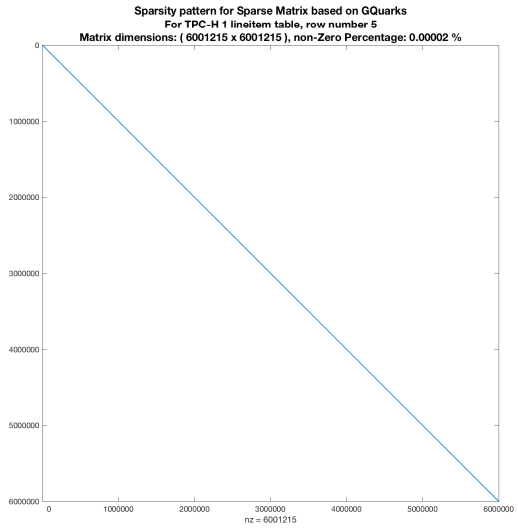

Figure 11: *Sparsity pattern analysis for attribute quantity, from the TPC-H dataset 1GB lineitem table(column #5).*

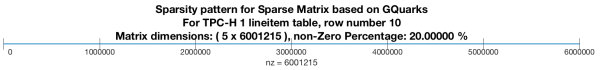

Figure 13: *Sparsity pattern analysis for attribute line status, from the TPC-H dataset 1GB lineitem table(column #10).*

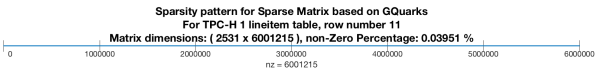

Figure 14: *Sparsity pattern analysis for attribute shipdate, from the TPC-H dataset 1GB lineitem table(column #11).*

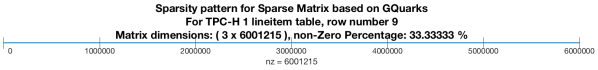

Figure 12: *Sparsity pattern analysis for attribute return flag, from the TPC-H dataset 1GB lineitem table(column #9).*

As visible on the prior sparsity analysis a matrix dimensions $(m \times n)$ will have in the worst case a non-zero number equal to n, since each column has at most one element. Therefore, to fully realise the potential of the processing units, it is very important that this memory bottleneck is overcome.

Given the example matrix of figure 11 the number of zero elements that would have to be stored and processed would be $(n-1)^2$. Taking into account that for the TPC-H dataset 32GB the number o element arises to 192000551, the necessary amount of memory to store only one single precision dense matrix would be 147 456 846 GB. Regarding the smaller data

chunks the hierarchy of memory consists of small fast memory at the top, closest to the arithmetic unit but getting larger and slower moving further away. Efforts should therefore be done in order to most of the memory system requests will be satisfied by the fast, high levels of memory, maximise occurrence of temporal and spatial locality so that the design of the code is consistent with the design of the caches. The used matrices must also be compressed to take advantage of their mathematical structure. With the prior statements in mind the storage formats chosen rely on the Compressed Sparse Column (CSC) and Compressed Sparse Row (CSR) formats [**?**].

Since each column has at most element, the CSC format construction is thereby simplified, giving place to an sequential column pointer array, bigger in size in one element when compared to the value and row index arrays. Denote that after the LA selection operation the column pointer array no longer keeps the illustrated property.

Regarding the usage of CSR format, it was chosen due to the creation of an alternative version to the LA operations that took CSC formatted matrices as input. This alternative version took CSR formatted matrices as input and relied on Intel® Math Kernel Library version 11.3.

VI. Incremental construction

Projection Matrices, as defined on section I.1 are amenable to incremental construction due to the GQuarks lema, where each unique string will be associated to an unique unsigned integer. If the new data does not exist in terms of a GQuark, a new association will be created and a new unique unsigned integer will be associated to it. Regarding the consequent matrix column increase, due to register addition, no problem arises from that action.

Dimension Matrices, also defined on section I.1, only depend on the matrix column increase due to the direct association of the cell value, having also no problem regarding incremental construction. Yesterday's data will still be valid and have zero migration cost with the addition of today's data.

III. Experimentation

Regarding the prior described lemas, we shall now assess wether if the linear algebra approach presents performance improvements when compared with the relational one. Before we discuss the measured performance results in the following section, we will briefly summarise characteristics of the multicore platform in our test suite, and present an overview of the performed tunings.

Throughout all the experiments, the same platform was used. The system, referenced as compute node 652-1, has two Intel® Xeon® E5-2670v2 (Ivy Bridge architecture) and features 64 GB of DDR3 RAM, supported at a frequency of 1333 MHz divided in 4 memory channels. Table 10 fully characterises the hardware features of the test platform:

System	compute-652-1
# CPUs	2
CPU	Intel® Xeon® E5-2670v2
Architecture	Ivy Bridge
# Cores per CPU	10
# Threads per CPU	20
Clock Freq.	2.5 GHz
L1 Cache	320KB 32KB per core
L2 Cache	2560KB 256KB per core
L3 Cache	25600KB shared
Inst. Set Ext.	SSE4.2 & AVX
#Memory Channels	4
Vendors Announced Peak Memory BW	59.7 GB/s
Measured[1] Peak Memory BW	58.5GB/s

Table 10: *Architectural characteristics of the evaluation platform.*

Regarding the software used for both relational and linear algebra, the corresponding versions are stated bellow:

- Linear Algebra:
  - Compiler: ICC version 16.0.0 (GCC version 4.4.6 compatibility)
    - no vectorization: -O3 -std=c99 -no-vec -farray-notation
    - vectorization: -O3 -std=c99 -farray-notation -xAVX -vec-report7
  - Intel® MKL Version 11.3
    - Link line: -lmkl_intel_lp64 -lmkl_core -lmkl_sequential -lpthread -lm
- Relational Algebra (PostgreSQL version 9.6+);
  - Built with the following dependencies:
    - GCC version 4.9.0
    - Python 2.6.6

Denote that PostgreSQL was compiled specifically for the test platform in order to fully take advantage of the available hardware.

I. Tuning the relational algebra engine

Database, application, and storage servers ship with a large number of configuration parameters like buffer cache sizes, number of I/O daemons, and parameters input to the database query optimiser. Finding good settings for these parameters is a challenging task because of the complex ways in which parameter settings can affect performance. The parameters

shared_buffers, effective_cache_size, and work_mem, were adjusted accordingly, with shared_buffers begin set to 2GB, effective_cache_size being set to 64GB, and work_mem being set to 25MB. In order to further speed up RA queries, indexes were created for all the database tables.

II. Tuning sparse CSC and CSR methods to assist data level parallelism

Regarding SSE vectorisation several efforts were made in order to introduce it in both versions of the linear algebra approach. The usage of the performance Library Intel Math Kernel library fully assisted vectorisation on the method responsible for the dot product between sparse matrices and sparse matrix vector. The compiler was also instructed via auto vectorisation hints, and user mandated vectorisation, of the non existence of vector dependencies when that case was verified.

Regarding the memory allocation alignment, all Data is aligned during creation accordingly to cache line size. Regarding the access alignment the compiler was instructed to to assume that all CSC and CSR arrays are aligned on an 32-byte boundary.

III. Results from the tuning of relational and linear algebra approaches

We conducted experiments on the following TPC-H simplified query-1:

IV. Parallelization

V. Experimetal Results

VI. Future Work

VII. Conclusion

VIII. Acknowledgements

References

@articlemacedo2015linear, title=A linear algebra approach to OLAP, author=Macedo, Hugo Daniel and Oliveira, José Nuno, journal=Formal Aspects of Computing, volume=27, number=2, pages=283–307, year=2015, publisher=Springer @articleda2015benchmarking, title=Benchmarking a Linear Algebra Approach to OLAP Master Thesis, author=da Costa Pontes, Rogério António, year=2015 @articlemacedo10matrices, title=Matrices as arrows! a biproduct approach to typed linear algebra, 2010, author=Macedo, HD and Oliveira, JN, journal=Submitted to MPC, volume=10 @techreportmacedo2011middle, title=Do the middle letters of ?OLAP? stand for linear algebra (?LA?), author=Macedo, Hugo Daniel and Oliveira, José Nuno, year=2011, institution=Technical Report TR-HASLab: 04: 2011, INESC TEC and University of Minho, Gualtar Campus, Braga