Parallel Data Cube Construction for High Performance On-Line Analytical Processing

Sanjay Goil and Alok Choudhary

Technical Report No. CPDC-TR-97-01

December 1997

Center for Parallel and Distributed Computing
Department of Electrical & Computer Engineering
Northwestern University
Technological Institute
2145 Sheridan Road, Evanston IL-60208

Parallel Data Cube Construction for High Performance On-Line Analytical Processing

Sanjay Goil Alok Choudhary
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60201

Abstract

Decision support systems use On-Line Analytical Processing (OLAP) to analyze data by posing complex queries that require different views of data. Traditionally, a relational approach (ROLAP) has been taken to build such systems. More recently, multi-dimensional database techniques (MOLAP) have been applied to decision-support applications. Data is stored in multi-dimensional arrays which is a natural way to express the multi-dimensionality of the enterprise and is more suited for analysis.

Precomputed aggregate calculations in a Data cube can provide efficient query processing for OLAP applications. In this paper we present algorithms and results for in-memory data cube construction on distributed memory machines.

1 Introduction

On-line Analytical Processing (OLAP) systems enable analysts and managers to gain insight into the performance of an enterprise through a wide variety of views of data organized to reflect the multi-dimensional nature of the enterprise [1] . OLAP gives insight into data through fast, consistent, interactive access to a wide variety of possible views of information. In contrast to traditional databases, it answers questions like "*what if ?*" and "*why ?*" in addition to "*who ?*" and "*what ?*". OLAP is used to build decision support systems which try extracting knowledge from data.

Typically, large amounts of data are analyzed for OLAP applications. Ad-hoc analytical queries are posed by analysts who expect the system to provide real-time query performance. By using parallel processing more memory is available and a larger database can be handled without going to disk. Scalable parallel algorithms can be developed to reduce the amount of time taken by a query. In this paper we consider main-memory databases. Extensions to disks using parallel I/O will be future work.

Distributed Memory Parallel Computers (shared-nothing machines) consist of a set of processors (tens to a few hundred) connected through an interconnection network. The memory is physically distributed among the processors and interaction between processors is through explicit message passing. Popular interconnection topologies are 2D meshes (Intel Paragon, Delta), 3D meshes (Cray T3D), hypercubes (nCUBE), fat tree (CM5) and multistage networks (IBM-SP2). Table 1 shows the collective communication operations used in our algorithms. Further details on these operations can be found in [6].

Section 2 gives an overview of the data cube and its operators. Section 3 presents the issues in data cube construction. Section 4 presents parallel data cube construction for MOLAP. Section 5 gives results for a OLAP benchmark suite on the IBM-SP2 parallel machine. Section 6 concludes the paper.

2 Data cube

Data cube computes the aggregates along all possible combinations of dimensions. This operation is useful for answering OLAP queries which use aggregation on different combinations of attributes. Data can be organized into a data cube by calculating all possible combinations of GROUP-BYs. For a dataset with k attributes this would lead to 2^k GROUP-BY calculations. In this paper we present algorithms for calculating the data cube using multi-dimensional arrays on a distributed memory parallel computer.

The cube treats each of the k aggregation attributes as a dimension in k-space. An aggregate of a particular set of attribute values is a point in this space. The set of points form a k-dimensional cube. Aggregate functions are classified into three categories [2]. *Distributive* functions allow partitions of the input set to be aggregated separately and later combined, e.g. *sum(), count()*. *Algebraic* functions can be expressed in terms of other distributive functions, e.g. *average()* can be expressed as *sum()* divided by *count()*. *Holistic* functions, such as *median()* cannot be computed in parts and combined. We do not consider holistic functions in this paper.

Operation	Description	Time
Broadcast	One processor sends a message of size m to all other processors.	$t_{bcast}m$
Combine	A binary associative operation applied on a vector of size m on every processor.	$t_{comb}m$
Gather	Collects data of size m from p processors and stores the resulting vector. of size mp in one of the processors.	$t_{gather}m$
All-to-All Communication	Each processor sends a distinct message of size at most m to every processor.	$t_{all_to_all}$m

Table 1: Collective communication operations

2.1 Operations on the data cube

Data Cube [2] operators generalize the histogram, cross-tabulation, roll-up, drill-down and sub-total constructs required by financial databases. The following operations can be defined on the data cube. **Pivoting**, also called *rotating*, involves rotating the cube to change the dimensional orientation of a report or page on display. **Slicing-dicing** involves selecting some subset of the cube. It can be visualized as *slice* of the data in a 3D-cube. **Roll-up** performs aggregations on different levels of hierarchy defined on the dimensions. For example, aggregating the dimension up the hierarchy ($day \rightarrow month \rightarrow quarter$..) is a roll-up operation. **Drill-down** traverses the hierarchy from lower to higher levels of detail. **Trend analysis** over sequential time periods is another operation.

Consider a multi-dimensional database with *product*, *date*, *supplier* as *dimensions* and *sales* as a *measure*. Partitioning a dataset into dimensions and measures is a design choice. Dimensions usually have a hierarchy associated with them, which specify aggregation levels and the granularity of viewing data. For example $day \rightarrow month \rightarrow quarter \rightarrow year$ is a hierarchy on date.

An aggregate of an attribute is represented by introducing a new value *ALL* [2]. The right hand side table in Figure 1 illustrates a data cube for the set of rows given in the left hand side.

The Data cube operator is the n-dimensional generalization of the group-by operator. Consider the following query which uses the cube operator.

SELECT Model, Year, Color, SUM(sales) AS Sales
FROM Sales
WHERE Model in 'Ford', 'Chevy'
AND Year BETWEEN 1990 AND 1992
GROUP BY CUBE(Model, Year, Color);

$2^N - 1$ aggregate calculations are needed for a N-dimensional data cube. For example, $2^3 = 8$ group-bys are calculated for the above query: {Model, Year, Color}, {Model, Year}, {Model, Color}, {Year, Color}, {Model}, {Year}, {Color} and ALL.

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

Model	Year	Color	Sales
Chevy	1990	Blue	87
Chevy	1990	Red	5
Chevy	1990	ALL	92
Chevy	ALL	Blue	87
Chevy	ALL	Red	5
Chevy	ALL	ALL	92
Ford	1990	Blue	99
Ford	1990	Green	64
Ford	1990	ALL	163
Ford	1991	Blue	7
Ford	1991	Red	8
Ford	1991	ALL	15
Ford	ALL	Blue	106
Ford	ALL	Green	64
Ford	ALL	Red	8
ALL	1990	Blue	186
ALL	1990	Green	64
ALL	1991	Blue	7
ALL	1991	Red	8
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

Figure 1: Data cube Illustration

2.2 OLAP Alternatives

Traditionally, OLAP systems have been build on top of a relational database system. These are referred to as relational OLAP (ROLAP). An OLAP engine is built on top of a relational database. This generates analytical queries in SQL which may become cumbersome for complex queries and affect performance. Relational systems have to embed a multi-dimensional view on the underlying data. Alternatively, multi-dimensional OLAP (MOLAP) systems have appeared only recently. These use multi-dimensional databases modeled as multi-dimensional arrays for OLAP operations. An intuitive view of data provides a sophisticated analytical functionality and support for complex queries. Data relationships are modeled more naturally and intuitively.

3 Data cube Construction

Several optimizations can be done over the naive method of calculating each aggregate separately from the initial data for the data cube operator [2].

1. **Smallest Parent:** For computing a group-by this selects the smallest of the previously computed group-bys from which it is possible to compute the group-by. Consider a four attribute cube

($ABCD$). Group-by AB can be calculated from $ABCD$, ABD and ABC. Clearly sizes of ABC and ABD are smaller than that of $ABCD$ and are better candidates. The actual choice will be made by picking the smaller of ABC and ABD.

2. **Effective use of Cache** This aims at using the cache effectively by computing the group-bys in such an order that the next group-by calculation can benefit from the cached results of the previous calculation. This can be extended to disk based data cubes by reducing disk I/O and caching in main memory. For example, after computing ABC from $ABCD$ we compute AB followed by A. In MOLAP systems the sizes of the intermediate levels are fixed and the order can be determined easily.

3. **Minimize inter-processor Communication:** Communication is involved among the processors to calculate the aggregates. The order of computation should minimize the communication among the processors because inter-processor communication costs are typically higher than computation costs. For example, $BC \rightarrow C$ will have a higher communication cost to first aggregate along B and then divide C among the processors in comparison to $CD \rightarrow C$ where a local aggregation on each processor along D will be sufficient.

A lattice framework to represent the hierarchy of the group-bys was introduced in [4]. A lattice for the group-by calculations for a four-dimensional cube ($ABCD$) is shown in Figure 2.

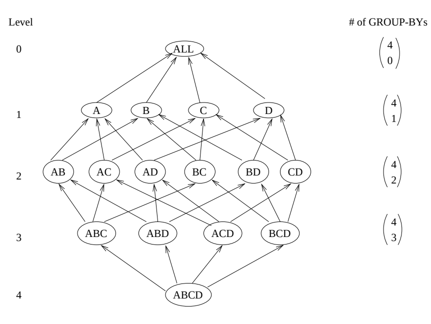

Figure 2: Lattice for cube operator

Calculation of the order in which the GROUP-BYs are created depends on the cost of deriving a lower order (one with a lower number of attributes) group-by from a higher order (also called the *parent*) group-by. For example, between ABD $\rightarrow$ BD and BCD $\rightarrow$ BD one needs to select the one with the lower cost. Cost estimation of the aggregation operations can be done by establishing a cost model. Let t_{op} be the cost of an addition, t_{copy} the cost of a copying a byte. Communication is modeled by collective communication operations as described in a previous section. We will analyze the costs of calculating the group-bys by factoring in these costs.

We assume that the total available memory on the processors is large enough to hold the datasets in memory. This is not an unfair assumption since most parallel machine these days have more than 128MB main memory per node. With 16 nodes we can handle databases of size 2GB and larger datasets can be handled by increasing the number of processors. Hence, it is important to develop scalable algorithms to handle larger databases.

4 Parallel Data cube Construction for MOLAP

We assume that data is provided as a set of rows stored in a file and the number of distinct elements are given for each attribute. For illustration purposes, let A, B, C and D be the attributes in a dataset with D_a, D_b, D_c and D_d as the number of their distinct values, respectively. Without loss of generality, let $D_a \geq D_b \geq D_c \geq D_d$. If this is not the case, it can be made true by a simple renaming of the attributes. Let p be the number of processors, numbered $P_0 \ldots P_{p-1}$, and N be the number of rows.

First, the rows are partitioned on p processors in a partitioning step. The *partitioning* phase is followed by a *loading* phase in which a multi-dimensional array is loaded on each processor from the rows acquired after the partitioning phase. This creates the *base cube*. Loading can either be done by a hash-based method or a sort-based method. We have implemented both and compare its scalability properties. This is followed by a *aggregation* phase which calculates the various aggregate sub-cubes. We describe each phase in the next few subsections.

4.1 Partitioning

A sample based partitioning algorithm is used to partition the rows among the processors. Attribute A is used in this partitioning. This is done to ensure the partitioning of data at the coarsest grain possible. This divides A nearly equally onto the processors and also establishes an order on A. If $A_x \in P_i$ and $A_y \in P_j$ then $A_x \leq A_y$ for $i < j$. In fact, in the partitioning scheme used here, rows end up being sorted locally on a processor.

4.2 Loading

Once rows are partitioned on processors, they are loaded into a multi-dimensional array (md-array). The size of the md-array in each dimension is the same as the number of unique values for the attribute represented in that dimension. A row is represented as a cell in the md-array indexed by the values of each of the attributes. Hence, each *measure* needs to be loaded in the md-array from the rows. We describe two methods to perform this task, a hash based method and a sort-based method. Figure 3 describes the hash-based cube loading algorithm.

4.2.1 Sort-based method

The sort-based method provides regularity of access over the sorted hash tables since the attributes probing them are sorted. The sorted hash-tables are scanned only in one direction for all the rows which have the same value for all the attributes in dimensions which come earlier in the sorted order, i.e have more unique values, since the order in which the attributes are sorted is descending with respect to their number of unique values. Due to lack of space we do not describe the algorithm here.

4.3 Aggregation

We illustrate the costs of calculating the various GROUP-BYs from the base cube for a three attribute (or 3D) cube in Table 2. Some calculations are *local* and some are *non-local* and need multiple processors to exchange data leading to communication among processors. Even with local calculations the costs can differ depending on how the calculation is made. Calculating AC from ABC requires summing on the B dimension and calculating AC from ACD requires aggregation on D. Depending on how the multi-dimensional array is stored these costs could be different since the stride of the access can affect the cache performance. As can be seen from the cost calculations, we see that the cost of calculating aggregates from a parent are lower if the order of the attributes in the aggregate is a prefix of the parent. Calculating ABCD $\rightarrow$ ABC $\rightarrow$ AB $\rightarrow$ A is a local calculation. Each cube is distributed across the processors since dimension A is distributed.

For a global aggregate calculation, first a local aggregation is done along dimension A. This is followed by a **Combine** operation on BCD. Each processor then keeps the corresponding portion of BCD, distributing B evenly among processors.

Clearly, as can be observed from the aggregation calculations shown in the table above, there are multiple ways of calculating a GROUP_BY. For calculating the aggregate at a level where aggregates can be calculated from multiple parents, we need to pick up an assignment such that the cost is minimized. In [3], this is solved by posing it as a graph problem and using minimum cost matching in a bipartite graph. Again, refer to Figure 2. Suppose we want to calculate aggregates

Source	Target	Cost
ABC $\rightarrow$	AB	$(\frac{D_a}{p} D_b D_c))t_{op}$
	AC	$(\frac{D_a}{p} D_b D_c)t_{op}$
	BC	$(\frac{D_a}{p} D_b D_c)t_{op} + (D_b D_c)t_{comb} +$ $(\frac{D_b}{p} D_c)t_{copy}$
AB $\rightarrow$	A	$(\frac{D_a}{p} D_b)t_{op}$
	B	$(\frac{D_a}{p} D_b)t_{op} + D_b t_{comb} + \frac{D_b}{p} t_{copy}$
AC $\rightarrow$	A	$(\frac{D_a}{p} D_c)t_{op}$
	C	$(\frac{D_a}{p} D_c)t_{op} + D_c t_{comb} + \frac{D_c}{p} t_{copy}$
BC $\rightarrow$	B	$(\frac{D_b}{p} D_c)t_{op}$
	C	$(\frac{D_b}{p} D_c)t_{op} + D_c t_{comb} + \frac{D_c}{p} t_{copy}$
A $\rightarrow$	ALL	$D_a t_{op}$
B $\rightarrow$	ALL	$D_b t_{op}$
C $\rightarrow$	ALL	$D_c t_{op}$

Table 2: Calculation of GROUP-BYs for a three attribute data cube ($D_a \times D_b \times D_c$)

at a level k from the parent at level $k+1$. A bipartite graph $G((V = X \cup Y), E)$ is defined as follows. A group of nodes X is the nodes at level k. Clearly $|X| = 2^k$. Another group of nodes $|Y|$ is the nodes at level $k+1$ and $|Y| = 2^{k+1}$. The edges connecting the nodes at level k and $k+1$ belong to E. The edge weights are the costs of calculating the particular aggregate at level k from the parent at level $k+1$. A node at level $k+1$ can possibly calculate $\binom{k+1}{k}$ aggregates at level k. Hence the nodes are replicated these many times at level $k+1$ and these are added to Y. We seek a match between nodes from level $k+1$ and level k which minimizes the total edge costs.

5 Results

We have implemented the algorithms presented in the previous section in 'C' and message passing using the Message Passing Interface (MPI) on the IBM SP-2 parallel machine. Each node of the SP-2 is a RS/6000 processor, with 128MB memory. The interconnection network is a multistage network connected through a high speed switch. The use of C and MPI makes the programs portable across a wide variety of parallel platforms with little or no effort.

5.1 Data sets

We have used the OLAP council benchmark which simulates a realistic On-Line Analytical Processing business situation. The data sets have attributes taken from the following set of attributes: Product (9000), Customer (900), Time (24), Channel (9) and Scenario (2). The values in brackets following each attribute are the number of unique values for that attribute. Product, Customer and Channel are character strings with 13 characters each. Time is an integer depicting year and month (e.g. 199704 is April 1997) and Scenario is

Algorithm 1 *Hash-based Data cube loading Algorithm*

k - number of attributes(dimensions)
Step 1. For each row hash each of the attributes into a separate hash table, one for each attribute. k hash tables are created as a result of this. (hash phase)
Step 2. Compress and sort each hash table.
Step 3. Update the index of the hash table with its order in the corresponding sorted list.
Step 4a. Pick up each row and probe the hash tables for each of its attributes to obtain the indices in each dimension. (probe phase)
Step 4b. Update the cell at the index in the md-array with the measure values of the relation.

Figure 3: *Hash-based* Algorithm for multi-dimensional Data cube Loading

a 6 character character string. The *measure* stored in each cell of the array is a **float** value depicting the sales figure for the particular combination of the attributes. This can potentially contain many more values. We have used the *Sum()* function to compute the aggregates in the results presented here.

5.2 Data cube Construction

We present results for the different phases of data cube construction for all the data sets described above.

Production Cost data contains 270,000 records and the performance of the two methods on this data set is shown in Figure 4. Again, as we increase the number of processor, each of the component in the construction algorithm scales well resulting in good overall scalability for both methods.

Figure 5 gives the performance on Budget data with nearly a million records for 8, 16, 32, 64 and 128 processors. It can be observed that for large data sets the methods perform well as we increase the number of processors.

Next, we compare the effect of density of the data sets on the data cube construction algorithm. The size of the multi-dimensional array is determined by the number of unique values in each of the dimensions. Hence the aggregation costs, which use the multi-dimensional array are not affected by the change in density as can be seen from Figure 6. The other phases of the data cube construction deal with the rows and the number of rows increase as the density of the data set increases. The costs for partition, sort, hash and the load phases increase because the number of rows on each processor increases. The scalability for both densities, and both hash-based and sort-based methods is good as observed from the figures.

6 Conclusions

On-Line Analytical Processing is fast gaining importance for business data analysis using large amounts of data. Multi-Dimensional databases provide support for complex analytical queries and are amenable to parallelization. Most data to be analyzed comes from relational data warehouses. Hence multi-dimensional databases need to interface with relational systems. In this paper, we have presented algorithms for constructing the data cube on distributed memory parallel computers. Sort-based and hash-based methods to load the multi-dimensional array from a relational database perform equally well on small data sets. We would expect the sort-based method to outperform the hash-based method for large amounts of data due to the regularity of accesses in it. Chunking of arrays can improve the performance of the sort-based method.

7 Acknowledements

This work was supported in part by NSF Young Investigator Award CCR-9357840 and NSF CCR-9509143.

References

[1] Codd E. F., *Providing OLAP to user-analysts : An IT mandate*, Technical Report, E.F. Codd and Associates, 1993.

[2] Gray J., Bosworth A., Layman A and Pirahesh H., *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*, ICDE'96.

[3] Sarawagi S., Agrawal R., and Gupta A., *On Computing the Data Cube*, Research Report 10026, IBM Almaden Research Center, San Jose, California, 1996.

[4] Harinarayan V., Rajaraman A. and Ullman J. D., *Implementing Data Cubes Efficiently*, SIGMOD'96.

[5] S. Sarawagi and M. Stonebraker, *Efficient Organization of Large Multi-Dimensional Arrays*, ICDE'94.

[6] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin Cummings, 1994.

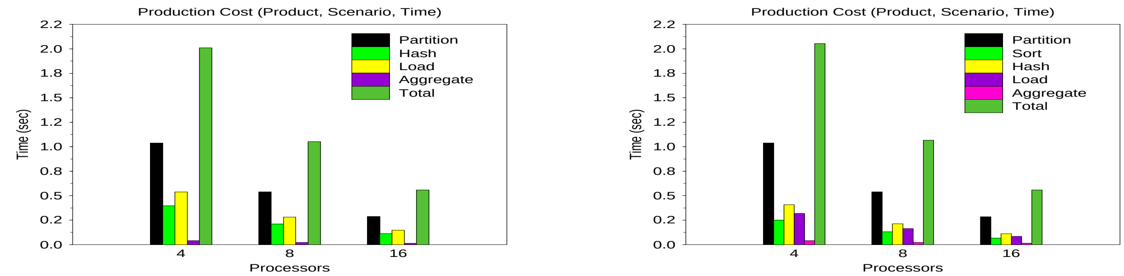

Figure 4: Various phases of cube construction using (a) hash-based (b) sort-based method for Production Cost data (N= 270,000 records and p = 4, 8 and 16 and data Size = 10MB)

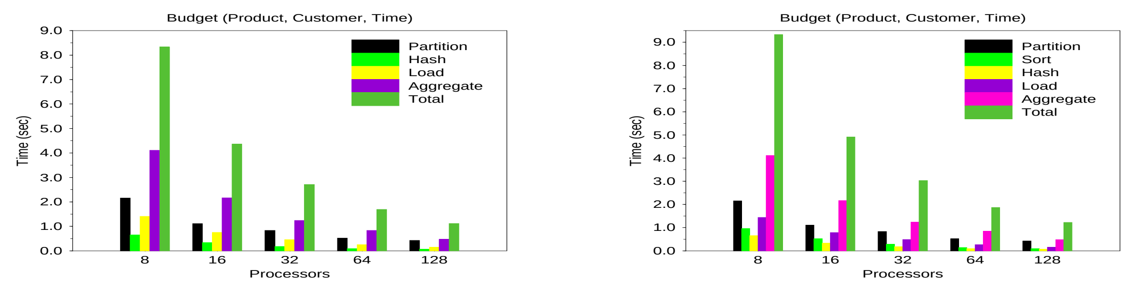

Figure 5: Various phases of cube construction using (a) hash-based (b) sort-based method for Budget data (N = 951,720 records and p = 8, 16, 32, 64 and 128 and data size = 420MB)

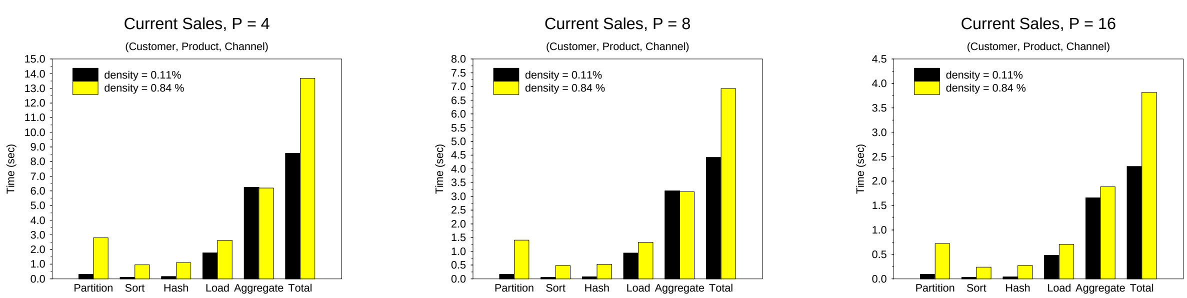

Figure 6: Comparison of cube construction components for two different densities for the sort based method