High Performance Data Mining Using Data Cubes On Parallel Computers [*]

Sanjay Goil Alok Choudhary
ECE Department and CPDC, Northwestern University
Technological Institute, 2145 Sheridan Road, Evanston IL-60208
{sgoil,choudhar}@ece.nwu.edu

Abstract

On-Line Analytical Processing techniques are used for data analysis and decision support systems. The multidimensionality of the underlying data is well represented by multidimensional databases. For data mining in knowledge discovery, OLAP calculations can be effectively used. For these, high performance parallel systems are required to provide interactive analysis.

Precomputed aggregate calculations in a Data Cube *can provide efficient query processing for OLAP applications. In this article, we present parallel data cube construction on distributed-memory parallel computers from a relational database. Data Cube is used for data mining of associations using* Attribute Focusing. *Results are presented for these on the IBM-SP2, which show that our algorithms and techniques are scalable to a large number of processors, providing a high performance platform for such applications.*

1. Introduction

On-line Analytical Processing (OLAP) [1] 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 its multidimensional nature. OLAP gives insight into data through fast, consistent, interactive access to a wide variety of possible views of information. It is used to summarize, consolidate, view, apply formulae to, and synthesize data according to multiple dimensions. Such data management and navigation requirements make multidimensional database techniques a natural candidate for OLAP, in which data is stored in a multidimensional array. A *cell* in multidimensional space represents a tuple, with the attribute values of the tuple identifying its location in the multidimensional space and the *measure* values represent the content of the cell.

[*]This work was supported in part by NSF Young Investigator Award CCR-9357840 and NSF CCR-9509143. We would like to thank Dr. Inderpal Bhandari for discussions and helpful suggestions.

Data mining can be viewed as an automated application of algorithms to detect patterns and extract knowledge from data [2]. An algorithm that enumerates patterns from, or fits models to, data is a data mining algorithm. Data mining is a step in the overall concept of knowledge discovery in databases (KDD). Large data sets are analyzed for searching patterns and discovering rules. Automated techniques of data mining can make OLAP more useful and easier to apply in the overall scheme of decision support systems. Data mining techniques like *Associations, Classification, Clustering and Trend analysis* [2] can be used together with OLAP to discover knowledge from data.

An approach to data mining called *Attribute Focusing* [5] targets the consumer by using algorithms that lead the user through the analysis of data by finding association between various attributes. Since data cubes have aggregation values on combinations of attributes already calculated, the computations of attribute focusing are greatly facilitated by data cubes. We present a parallel algorithm to calculate the *interestingness* function used in attribute focusing on the data cube.

In this article we present scalable parallel algorithms and techniques to construct the data cube and perform data mining through attribute focusing on the data cube. We have currently considered main-memory data sets. Disk-based implementation using parallel I/O is currently in progress. We show the performance of these algorithms on the OLAP council benchmark [14] which models a real OLAP environment on a IBM SP-2 parallel machine. Results show that our implementations are scalable to a large number of processors. To the best of our knowledge, this is the first effort on high performance parallel computation of data cubes and data mining using them.

The rest of the article is organized as follows. Communication operations used are given in Section 2. Section 3 gives an overview of the data cube operators. Section 4 presents parallel data cube construction. Section 5 describes data mining on data cubes with results on the IBM-SP2. Section 6 concludes the article.

2. Communication Costs

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 across the processors. Interaction between processors is through message passing. Popular interconnection topologies are 2D meshes (Paragon, Delta), 3D meshes (Cray T3D), hypercubes (nCUBE), fat tree (CM5) and multistage networks (IBM-SP2).

Parallelization of applications requires distributing some or all of the data structures among the processors. Each processor needs to access all the non-local data required for its local computation. This generates aggregate or collective communication structures. In a **Broadcast**, one processor sends a message of size m to all other processors. **Combine** is a binary associative operation applied on a vector of size m on every processor. **Gather** collects data of size m from p processors and stores the resulting vector of size mp in one of the processors. In a **All-to-All Communication**, each processor sends a distinct message of size at most m to every processor. Further details on these operations can be found in [11].

3. Data Cube

The data cube operator is the n-dimensional generalization of the group-by operator [7]. 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.

Data cube computes multiple 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 data set with k attributes this would lead to 2^k GROUP-BY calculations. An aggregate of an attribute is represented by introducing a new value *ALL* in a tuple.

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. *Distributive* functions allow partitions of the input set to be aggregated separately and later combined. *Algebraic* functions can be expressed in terms of other distributive functions, e.g. *average()* can be expressed as the ratio of *sum()* and *count()*. *Holistic* functions, such as *median()* cannot be computed in parts and combined.

3.1. Operations on the Data Cube

Data Cube 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** involves rotating the cube to change the dimensional orientation of a report or page on display. It may consist of swapping the two dimensions (row and column in a 2D-cube) or introducing another dimension instead of some dimension already present in the cube. **Slicing-dicing** involves selecting some subset of the cube. For a fixed attribute value in a given dimension, it reports all the values for all the other dimensions. It can be visualized as *slice* of the data in a 3D-cube. Some dimensions have a hierarchy defined on them. Aggregations can be done at different levels of hierarchy. Going up the hierarchy to higher levels of generalization is known as **roll-up**. 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. Drill-down displays detail information for each aggregated point. Trend analysis over sequential time periods is another OLAP operation.

As an example, consider a multidimensional database with *product*, *date*, *supplier* as *dimensions* and *sales* as a *measure*. Partitioning a data set 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 OLAP engine can be 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 multidimensional view on the underlying data. Alternatively, multidimensional OLAP (MOLAP) systems use multidimensional databases modeled as multidimensional arrays. An intuitive view of data provides a sophisticated analytical functionality and support for complex queries. Data relationships are modeled more naturally and intuitively.

3.2. Optimizations

Several optimizations can be done over the naive method of calculating each aggregate separately from the initial data [7]. Optimizations 1 and 2 are normally considered for a uniprocessor model. Optimization 3 is an added and important consideration for a parallel implementation to reduce the overheads from communication costs.

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 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.

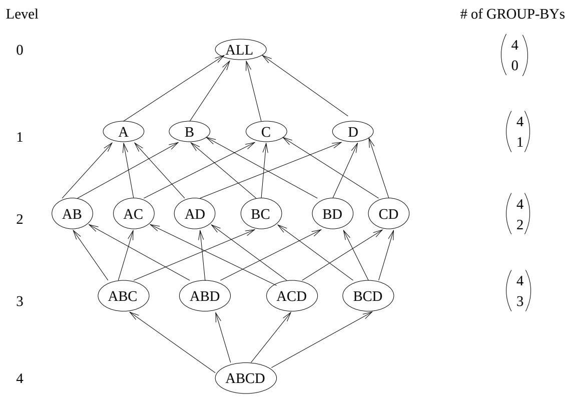


Figure 1. Lattice for cube operator

A lattice framework to represent the hierarchy of the group-bys was introduced in [10]. This is an elegant model for representing the dependencies in the calculations and also to model costs of the aggregate calculations. A scheduling algorithm can be applied to this framework substituting the appropriate costs of computation and communication. A lattice for the group-by calculations for a four-dimensional cube ($ABCD$) is shown in Figure 1. Each node represents an aggregate and an arrow represents a possible aggregate calculation which is also used to represent the cost of the calculation.

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. This is described later in the section on aggregation.

We assume that the total available memory on the processors is large enough to hold the data sets in memory. This is a reasonable assumption since most parallel machines these days have 128-256 MB main memory per node. With 16 nodes we can handle databases of size 2GB and larger data sets can be handled by increasing the number of processors. Hence, it is important to develop scalable algorithms to handle larger databases. In this article we develop in-memory algorithms to calculate the data cube. Disk-based algorithms are also being explored as part of this research.

4. Parallel Data Cube Construction

We assume that data is provided as a set of tuples coming from a relational database 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 data set with D_a, D_b, D_c and D_d as the number of their distinct values, respectively. We assume that the number of distinct values in each dimension is known. However, the values are determined from the database by the algorithm. 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 tuples.

Figure 2 shows the various steps in the data cube construction algorithm.

Algorithm 1 *Parallel data cube construction and operations*

```
1.Partition tuples between processors    (Partitioning)
2.Load tuples into multidimensional array    (Loading)
3.Generate schedule for order of group-by calculations
4.Perform aggregation calculations    (Aggregation)
5.Redistribute sub-cubes to processors for query processing
6.Define local and distributed hierarchies on all dimensions
```

Figure 2. Parallel cube construction steps

First, the tuples are partitioned on p processors in a partitioning step. The *partitioning* phase is followed by a *loading* phase in which a multidimensional array is loaded on each processor from the tuples 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 have compared their 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 tuples 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, tuples end up being sorted locally on each processor.

4.2. Loading

Once tuples are partitioned on processors, they are loaded into a multidimensional 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 tuple 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 tuples. We describe two methods to perform this task, a hash based method and a sort-based method.

4.2.1. Hash-based method

Each attribute is hashed in a separate hash table to get unique values for it. A sort on the attribute's hash table will index the dimension of the base cube corresponding to that attribute. These hash tables are then probed to fill in the measure values in the base cube. Hashing techniques are known to provide good performance on the average, though it heavily depends on the choice of a good hash function.

4.2.2. Sort-based method

This method provides regularity of access over the sorted hash tables since the attributes probing them are sorted, unlike the hash-based method, where accesses in the hash table have no order. The sorted hash-tables are scanned only in one direction for all the tuples which have the same value for all the attributes in dimensions which come earlier, i.e have more unique values, since the order in which the attributes are sorted is with respect to their number of unique values. For example, for two consecutive records (a_1, b_1, c_1, d_1) and (a_1, b_1, c_1, d_4), hash table for D is scanned from the current position to get the index. However, for (a_1, b_1, c_1, d_4) and (a_1, b_2, c_1, d_1), hash tables for both C and D need to be scanned from the beginning.

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 1. Calculation of GROUP-BYs for a three attribute data cube ($D_a \times D_b \times D_c$), on p processors

4.3. Aggregation

Each processor has $\frac{D_a}{p}$ portion of A. The costs of calculating the GROUP-BYs from the base cube for a three attribute (or 3D) cube is given in Table 1. Let t_{op} be the cost of an addition, t_{copy} the cost of copying a byte. t_{comb} is the cost of a Combine operation. These costs are then used by a scheduling algorithm, which generates a schedule for calculating the various group-bys.

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 multidimensional array is stored these costs could be different since the stride of the access can affect the cache performance. From the cost calculations shown in Table 1, 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 ABC $\rightarrow$ AB $\rightarrow$ A is a local calculation on each node.

Each cube is distributed across the processors since dimension A is distributed. The intermediate cubes, resulting from aggregation of parent cubes are also distributed among the processors. This results in good load balancing among the processors. Calculations involving multiple cubes can also be distributed across processors as a result of this. The first attribute in the aggregate cube is always distributed among processors. As a result, A is distributed in ABCD, ABC, AB and A, B is distributed in BCD, BC and B, and C is distributed in CD and C and D is distributed.

Figure 3 illustrate a global aggregate calculation of cal-

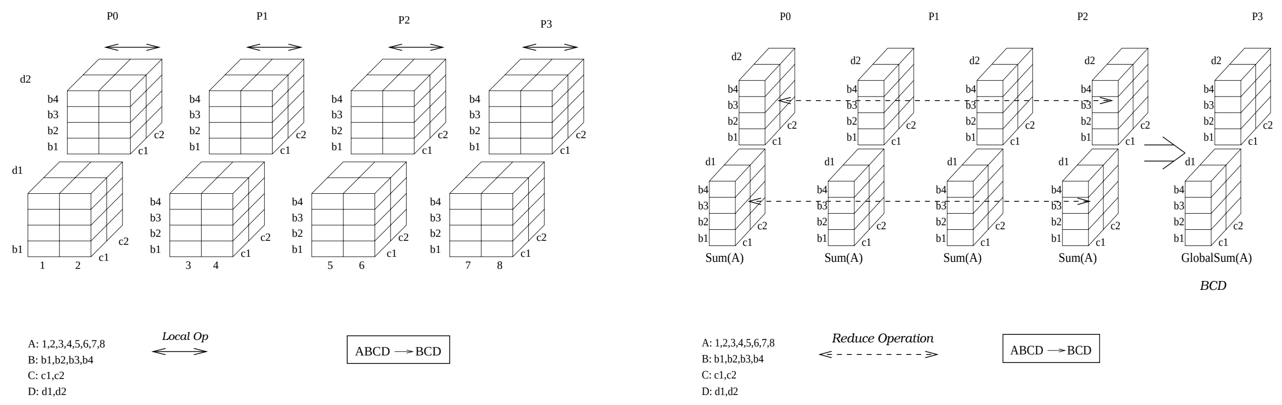


Figure 3. *Global* **aggregation calculation, ABCD → BCD, (a) local phase (b) global phase**

culating $ABCD \rightarrow BCD$. 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 [12], this is solved by posing it as a graph problem and using minimum cost matching in a bipartite graph. We have augmented it by our cost model and the optimizations needed. Again, refer to Figure 1. Suppose we want to calculate aggregates 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 $k+1$. Costs described in Table 1 are used here. A node at level $k+1$ can possibly calculate $\begin{pmatrix} k+1 \\ k \end{pmatrix}$ 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. This is done for each level and the resulting order of calculations at each level is picked up. This creates a directed acyclic graph which is then traversed, from the base cube as the root, to each child in a depth first search manner, calculating each aggregation. This results in preserving all of the three optimizations for multiple group-bys described in an earlier section.

4.4. Results

We have implemented the algorithms presented in the previous section using '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 effort.

We have used the OLAP council benchmark [14] 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). Product, Customer and Channel are character strings with 12 characters each. Time is an integer depicting year and month (e.g. 199704 is April 1997) and Scenario is a 6 character string showing if it is an "Actual" or a "Budget" value. The *measure* stored in each cell of the array is a **float** value for the sales figure. *Sum()* function is used here to compute the aggregates in the results presented here.

Figure 4 gives the performance on History Sales data containing 4 attributes (4 dimensional base cube) with around 1 million records for 8, 16, 32 and 64 processors. The density of the cube is 1%. The hash-based method performs better than the sort-based method for this case. The addition of a dimension increases the stride factor for accesses for the sort-based loading algorithm, deteriorating the cache performance. The hash-based method has no particular order of reference in the multidimensional array and benefits from a better cache performance. It can be observed that for large data each component in cube construction scales well as we increase the number of processors.

We have compared the effect of density of the data sets on the cube construction algorithm. The aggregation costs, which use the multi-dimensional array are not affected by the change in density. The other phases of the data cube construction deal with the tuples and the number of tuples increase as the density of the data set increases. The costs for partition, sort, hash and the load phases increase because the number of tuples on each processor increases.

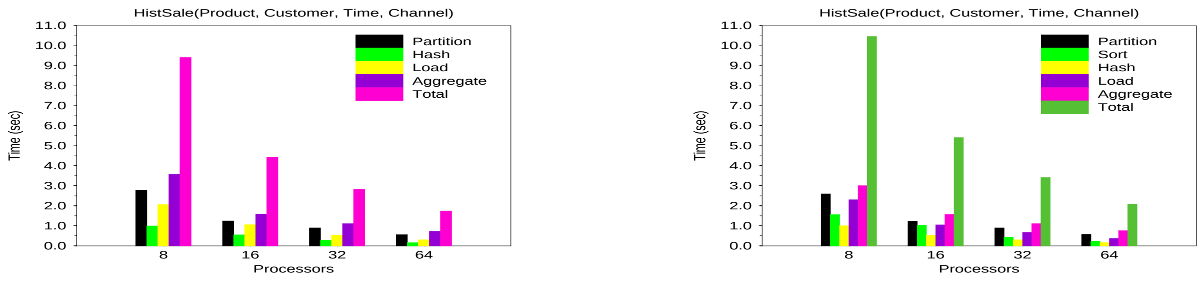


Figure 4. Various phases of cube construction using (a) hash-based (b) sort-based method for History Sales data (N = 1,010,000 records, p = 8, 16, 32 and 64 and data size = 500MB)

5. Data Mining on Data Cubes

Data mining techniques are used to discover patterns and relationships from data which can enhance our understanding of the underlying domain. A data mining tool can be understood along the following components : *data*, what is analyzed ; *statistics*, the kind of patterns that are searched for by the tool; *filtering*, how many of those patterns are to be presented to the user and the selection of those patterns; *visualization*, the visual representation used to present the patterns to the user; *interpretation*, what should the user think about when studying the pattern; *exploration*, in what order should the patterns be considered by the user; *process support*, how does one support data mining as a process as opposed to an exercise done only once. Attribute focusing (AF) is a data mining method that has most of these but in particular relies on exploration and interpretation [5].

Discovery of quantitative rules is associated with quantitative information from the database. The data cube represents quantitative summary information for a subset of the attributes. Attribute-oriented approaches [5] [8] [9] to data mining are data-driven and can use this summary information to discover association rules. Transaction data can be used to mine association rules by associating *support* and *confidence* for each rule. Support of a pattern A in a set S is the ratio of the number of transactions containing A and the total number of transactions in S. Confidence of a rule $A \rightarrow B$ is the probability that pattern B occurs in S when pattern A occurs in S and can be defined as the ratio of the support of AB and support of A. The rule is then described as $A \rightarrow B$ [*support, confidence*] and a *strong* association rule has a support greater than a pre-determined minimum support and a confidence greater than a pre-determined minimum confidence. This can also be taken as the measure of "interestingness" of the rule. Calculation of support and confidence for the rule $A \rightarrow B$ involve the aggregates from the cube AB, A and ALL. Additionally, dimension hierarchies can be utilized to provide multiple level data mining by progressive generalization (roll-up) and deepening (drill-down). This is useful for data mining at multiple concept levels and interesting information can potentially be obtained at different levels.

Attribute Focusing calculates associations between attributes by using the notion of percentages and subpopulations. The overall distribution of an attribute is compared with its distribution in various subsets of data. If a certain subset of data has a characteristically different distribution for the focus attribute, then that combination of attributes are marked as interesting.

An event, E is a string $E_n = x_1, x_2, x_3, \ldots x_n$; in which x_j is a possible value for some attribute and x_k is a value for a different attribute of the underlying data. E is interesting to that x_j's occurrence depends on the other x_i's occurrence. The "interestingness" measure is the size $I_j(E)$ of the difference between:
(a) the probability of E among all such events in the data set and,
(b) the probability that $x_1, x_2, x_3, \ldots x_{j-1}, x_{j+1}, \ldots x_n$ and x_j occurred independently. The condition of interestingness can then be defined as $I_j(E) > \delta$, where δ is some fixed threshold.

Another condition of interestingness, in attribute focusing depends on finding the optimal number of attribute values, n formally described as $I_j(E_n) > I_j(E_{n-1})$; $I_j(E_n) \geq I_j(E_{n+1})$; where $E_n = x_1, x_2, x_3, \ldots, x_n$. AF seeks to eliminate all but the most interesting events by keeping E only if the number of attribute values, n, is just right. Eliminate one or more x_i's and I_j decreases, include one ore more new x_i's to the string and I_j gets no larger. The convergence to n removes patterns like E_{n-1} and E_{n+1} which are less interesting than E_n and have information already contained by E_n. Hence, as a result of this the user does not have to drill down or roll up from a highlighted pattern, since the event descriptions returned are at their most interesting level.

5.1. Attribute Focusing on data cubes

In this section we present an algorithm to compute the first measure of interestingness of the attribute focusing method using data cubes. Figure 5 shows the algorithm.

Algorithm 2 *Calculation of interestingness using data cubes*

```
1.Replicate each single attribute sub-cubes on all processors using a **Gather** followed by a **Broadcast**.
2.Perform a **Combine** operation of ALL (0D cube) followed by a **Broadcast** to get the correct value of ALL on all processors.
3.Take the ratio of each element of the AB sub-cube and ALL to get $P(AB)$. Similarly calculate $P(A)$ and $P(B)$ using the replicated sub-cubes
  A and B.
4.For each element $i$ in AB calculate $|P(AB) - P(A)P(B)|$, and compare it with a threshold $\delta$, setting AB[i] to 1 if it is greater, else set it to 0.
```

Figure 5. Algorithm for calculating interestingness between attributes A and B with data cubes

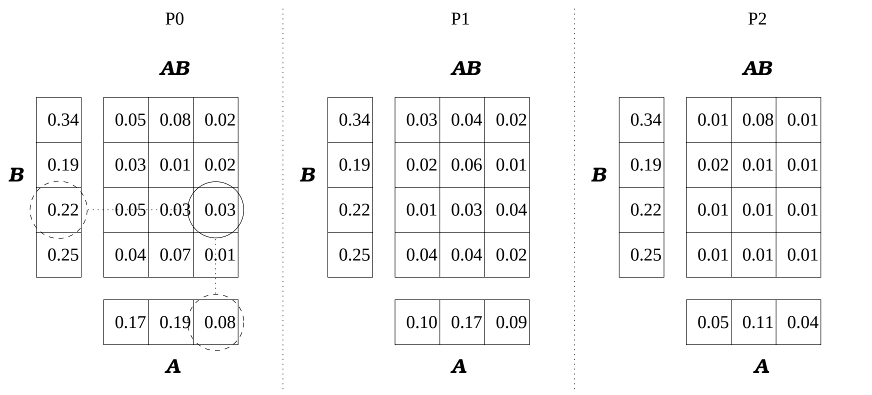


Figure 6. Use of sub-cubes AB, A and B for calculations on 3 processors

Consider a 3 attribute data cube with attributes A, B and C, defining $E_3 = ABC$. For showing 2-way associations, we will calculate the interestingness function between A and B, A and C and finally between B and C. When calculating associations between A and B, the probability of E, denoted by $P(AB)$ is the ratio of the aggregation values in the sub-cube AB and ALL. Similarly the independent probability of A, $P(A)$ is obtained from the values in the sub-cube A, dividing them by ALL. $P(B)$ is similarly calculated from B. The calculation $|P(AB) - P(A)P(B)| > \delta$, for some threshold δ, is performed in parallel. Since the cubes AB and A are distributed along the A dimension no replication of A is needed. However, since B is distributed in sub-cube B, and B is local on each processor in AB, B needs to be replicated on all processors. AB and A cubes are distributed, but B is replicated on all processors. Figure 6 shows a sample calculation of P(AB), P(A) and P(B) on three processors. A sample calculation is, highlighted in the figure, $|0.03 - 0.22 \times 0.08| = 0.0124$ which is greater than δ values from 0.001 to 0.01, and the corresponding attribute values are associated within that threshold.

5.2. Results : Data Mining

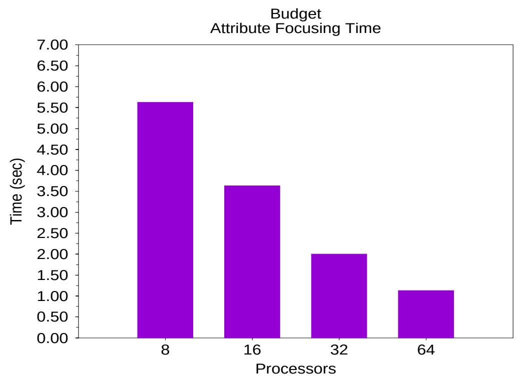
We have used a few data sets from the OLAP council benchmark. We perform 2-way association calculations by performing attribute focusing for all combinations of two attributes. For example, for a 3D cube, ABC, interestingness measure will be calculated between A and B, A and C and between B and C. Typically, a few different δ values are used for analysis of data, to vary the degree of association between the attributes. We run each calculation for 20 different δ values, ranging from 0.001 to 0.02 in steps of 0.01.

Figure 7 shows the run times we observe for the Budget data for attribute focusing calculations on 8, 16, 32 and 64 processors. The associated communication time (in milliseconds) is also given.

6. Conclusions

On-Line Analytical Processing is fast gaining importance for business data analysis using large amounts of data now available in data warehouses. Aggregations are an important function of OLAP queries and can benefit from the data cube operator introduced in [7]. Multidimensional databases model the multi-dimensionality of data intuitively, providing support for complex analytical queries, also being amenable to parallelization. Summary results in the data cube can be used to perform data mining through attribute focusing methods. We have presented a parallel algorithm to perform attribute focusing on the data cube.

In this article, we presented algorithms and techniques for constructing multidimensional data cubes on distributed memory parallel computers and perform data mining through attribute focusing on them. Our techniques are



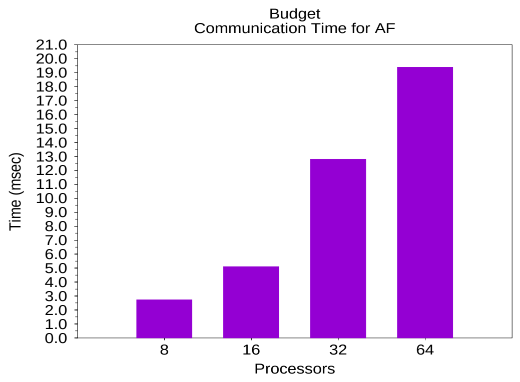


Figure 7. Time for (a) attribute focusing for 20 different δ values for Budget data, (b) associated communication cost (in milliseconds) on 8, 16, 32 and 64 processors

platform independent and are portable across a wide variety of parallel platforms without much effort.

References

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

[2] Fayyad U.M, Piatesky-Shapiro G., Smyth P. and Uthurusamy R., "From data mining to knowledge discovery: An overview", Advances in data mining and knowledge discovery, MIT Press, pp. 1-34.

[3] Bhandari I., Colet E., et.al., "Advanced Scout: Data Mining and Knowledge Discovery in NBA Data", Research Report RC 20443, IBM T.J Watson Research Center, 1996.

[4] Bhandari I., Halliday M., Tarver E., Brown D., Chaar J. and Chillarege R., "A case study of software process improvement during development", IEEE Transactions on Software Engineering, 19(12), December 1993, pp. 1157-1170.

[5] Bhandari I., "Attribute Focusing: Data mining for the layman", Research Report RC 20136, IBM T.J Watson Research Center.

[6] Goil S. and Choudhary A., "Parallel Data Cube Construction for High Performance On-Line Analytical Processing", Proc. 4th Intl. Conf. on High Performance Computing, Bangalore, India, 1997.

[7] Gray J., Bosworth A., Layman A and Pirahesh H., "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals", Proc. Intl. Conf. on Data Engineering, 1996.

[8] Han J., Cai Y. and Cercone N., Data-Driven Discovery of Quantitative Rules in Relational Databases, *IEEE Trans. on Knowledge and Data Engineering, Vol 5, No. 1, February 1993*.

[9] Han J. and Fu Y., Discovery of Multiple-Level Association Rules from Large Databases, *Proc. of the* 21^{st} *VLDB Conference, Zurich, 1995.*

[10] Harinarayan V., Rajaraman A. and Ullman J. D., "Implementing Data Cubes Efficiently", Proc. SIGMOD'96.

[11] Kumar V., Grama A., Gupta A. and Karypis G., "Introduction to Parallel Computing: Design and Analysis of Algorithms", Benjamin Cummings Publishing Company, California, 1994.

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

[13] Sarawagi S. and Stonebraker M., "Efficient Organization of Large Multidimensional Arrays", Proc. of the 11th Intl. Conf. on Data Engineering, Houston, February 1994.

[14] "OLAP Council Benchmark" available from http://www.olapcouncil.org