High Performance OLAP and Data Mining on Parallel Computers

Sanjay Goil and Alok Choudhary

Technical Report No. CPDC-TR-97-05

©1997 Center for Parallel and Distributed Computing

December 1997

Center for Parallel and Distributed Computing

Department of Electrical & Computer Engineering

Northwestern University

Technological Institute

2145 Sheridan Road, Evanston IL-60208

High Performance OLAP and Data Mining on Parallel Computers *

Sanjay Goil Alok Choudhary
Department of Electrical & Computer Engineering and
Center for Parallel and Distributed Computing,
Northwestern University,
Technological Institute, 2145 Sheridan Road, Evanston, IL-60208
{sgoil,choudhar}@ece.nwu.edu

January 26, 1998

Abstract

On-Line Analytical Processing (OLAP) techniques are increasingly being used in decision support systems to provide analysis of data. Queries posed on such systems are quite complex and require different views of data. Analytical models need to capture the multidimensionality of the underlying data, a task for which multidimensional databases are well suited. Multidimensional OLAP systems store data in multidimensional arrays on which analytical operations are performed. Knowledge discovery and data mining requires complex operations on the underlying data which can be very expensive in terms of computation time. High performance parallel systems can reduce this analysis time.

Precomputed aggregate calculations in a *Data Cube* can provide efficient query processing for OLAP applications. In this article, we present algorithms for construction of data cubes on distributed-memory parallel computers. Data is loaded from a relational database into a multidimensional array. We present two methods, *sort-based* and *hash-based* for loading the *base* cube and compare their performances. Data cubes are used to perform consolidation queries used in *roll-up* operations using dimension hierarchies. Finally, we show how data cubes are used for data mining using *Attribute Focusing* techniques. We present results for these on the IBM-SP2 parallel machine. Results show that our algorithms and techniques for OLAP and data mining on parallel systems are scalable to a large number of processors, providing a high performance platform for such applications.

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

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 the multidimensional nature of the enterprise data. 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 help in extracting knowledge from data.

OLAP is used to summarize, consolidate, view, apply formulae to, and synthesize data according to multiple dimensions. Queries posed on such systems are quite complex and require different views of data. Traditionally, a relational approach (relational OLAP) has been taken to build such systems. Relational databases are used to build and query these systems. A complex analytical query is cumbersome to express in SQL and it might not be efficient to execute. More recently, multidimensional database techniques (multidimensional OLAP) have been applied to decision-support applications. Data is stored in multidimensional arrays which is a more natural way to express the multi-dimensionality of the enterprise data and is more suited for analysis. A "cell" in multidimensional space represents a tuple, with the attributes of the tuple identifying the location of the tuple in the multidimensional space and the measure values represent the content of the cell.

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 targets the consumer by using algorithms that lead the user through the analysis of data. Attribute Focusing has been successfully applied in discovering interesting patterns in the NBA [5] and other applications. Earlier applications of this technique were to software process engineering [4]. 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.

Typically, large amounts of data are analyzed for OLAP and data mining applications. Ad-hoc analytical queries are posed by analysts who expect the system to provide real-time performance. Parallel computers can be used in such a scenario for a variety of reasons. Scalable solutions can build a highly accurate data mining model quicker. Mining large databases and constructing complex models take a large amount of computing power which can take hours of valuable time. Scalable parallel systems can reduce this wait time to minutes or even seconds, thus increasing productivity and better understanding of the knowledge discovery process. The use of many processors enables the use of more memory and a larger database can be handled in the main memory attached to the processors. We have currently considered main-memory databases. Extensions to disks using parallel I/O will be future work.

In this article we present scalable parallel algorithm and techniques for OLAP in multidimensional databases. Parallel construction and maintenance of data cubes and their use for OLAP queries is shown. Results show that our implementations are scalable to a large number of processors. Consolidation queries make use of hi-

Category	Examples
Distributive	Sum(), Count(), Minimum(), Maximum()
Algebraic	Average(), Standard_Deviation(), MaxN()(N largest values), MinN() (N smallest values), Center_of_Mass()
Holistic	Median(), MostFrequent()(i.e. Mode()), Rank()

Table 1: Categories of aggregate functions

erarchies on dimensions to make OLAP queries possible at different levels of detail. We show the performance of these algorithms on the OLAP council benchmark [15] which models a real OLAP environment on a IBM SP-2 parallel machine. The IBM SP-2 is a network of RS/6000 workstations connected together on a high speed communication switch, which is fast getting popular as a parallel computing platform. We show the use of data cubes to perform data mining by using attribute focusing to find two-way associations between attributes. These can easily be extended to n-way associations. To the best of our knowledge, this is the first effort on high performance parallel computation of data cubes for MOLAP and data mining using them.

The rest of the paper is organized as follows. Section 2 gives an overview of the data cube and its operators. The model of parallel computation we have used is given in Section 3. Section 4 presents the issues in data cube construction. Section 5 presents parallel data cube construction for MOLAP. Section 6 gives results for a OLAP council benchmark suite on the IBM-SP2 parallel machine. Section 7 presents consolidation queries that use hierarchies defined on the various dimensions. Section 8 describes data mining on data cubes with results on the IBM-SP2. Section 9 concludes the article.

2 Data Cube

The Data cube operator was introduced recently by Gray et. al [7] to support multiple aggregates. Data cube computes 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. In this article we present algorithms for calculating the data cube using multidimensional 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 as shown in Table 1. *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.

2.1 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:** This is also called *rotating* and 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 :** This operation 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.
- **Roll-up :** 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:** This operation 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

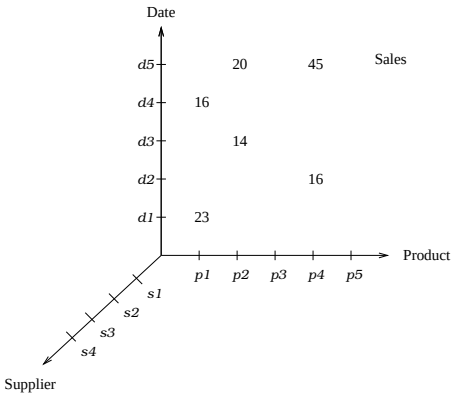


Figure 1: Example database of sales data

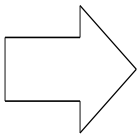
Figure 1 shows a multidimensional database with *product*, *date*, *supplier* as *dimensions* and *sales* as a *measure*. Partitioning a data set into dimensions and measures is a design choice. Dimensions usually have a hierarchy associated with them, which specify aggregation levels and the granularity of viewing data. For example $day \rightarrow month \rightarrow quarter \rightarrow year$ is a hierarchy on date.

An aggregate of an attribute is represented by introducing a new value *ALL*. The right hand side table in Figure 2 illustrates a data cube for the set of tuples given in the left hand side [7].

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 multidimensional view on the underlying data. Alternatively, multidimensional OLAP (MOLAP) systems have appeared only recently. These use multidimensional databases modeled as

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 2: Data cube Illustration using a relation

multidimensional 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.

Spatial databases [8] represent and model geometric objects (points, lines, polygons etc.) in multidimensional space. MOLAP data can be considered as points in the multidimensional space of attributes and benefit from some spatial database techniques. The operations in OLAP although different from the operations, such as *overlap*, *containment* etc., used in spatial databases can benefit from the spatial indexing structures developed for spatial databases.

3 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).

Most of these machines have cut-through routed networks which will be used for modeling the communication cost of our algorithms. For a lightly loaded network, a message of size m traversing d hops of a cut-through (CT) routed network incurs a communication delay given by $T_{comm} = t_s + t_h d + t_w m$, where t_s represents the handshaking costs, t_h represents the signal propagation and switching delays and t_w represents the inverse bandwidth of the communication network. The startup time t_s is often large, and can be several

Primitive	Running time on p processors
Broadcast	$O((t_s + t_w m) \log p)$
Reduce	$O((t_s + t_w m) \log p)$
Parallel Prefix	$O((t_s + t_w) \log p)$
Gather	$O(t_s \log p + t_w mp)$
All-to-All Communication	$O((t_s + t_w m)p + t_h p \log p)$

Table 2: Running times of various parallel primitives on a multi-staged network

hundred machine cycles or more. The per-word transfer time t_w is determined by the link bandwidth. t_w is often higher (an order to two orders of magnitude is typical) than t_c, the time to do a unit computation on data available in the cache.

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.

Several algorithms have been described in the literature for these primitives and are part of standard textbooks [12]. The use of collective communication provides a level of architecture independence in the algorithm design. It also allows for precise analysis of an algorithm by replacing the cost of the primitive for the targeted architecture. Table 2 provides a complexity analysis of these operations on a multi-staged network. These costs are used in the analysis of the algorithms presented in the next section.

1. **Broadcast.** In a Broadcast operation, one processor has a message of size m to be broadcast to all other processors.

2. **Reduce.** Given a vector of size m on each processor and a binary associative operation, the Reduce operation computes a resultant vector of size m and stores it on every processor. The i^{th} element of the resultant vector is the result of combining the i^{th} element of the vectors stored on all the processors using the binary associative operation.

3. **Parallel Prefix.** Suppose that $x_0, x_1, \ldots, x_{p-1}$ are p data elements with processor P_i containing x_i. Let $\otimes$ be a binary associative operation. The Parallel Prefix operation stores the value of $x_0 \otimes x_1 \otimes \ldots \otimes x_i$ on processor P_i.

4. **Gather.** Given a vector of size m on each processor, the Gather operation collects all the data and stores the resulting vector of size mp in one of the processors.

5. **All-to-All Communication.** In this operation each processor sends a distinct message of size m to every processor.

4 Data Cube Construction

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, as shown in Figure 2.

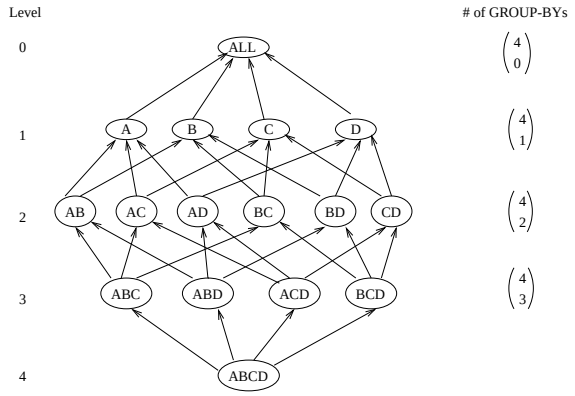


Figure 3: Lattice for cube operator

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 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 [11]. This is an ele-

gant 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 3. 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. External algorithms are also being explored as part of this research.

5 Parallel Data Cube Construction for MOLAP

We assume that data is provided as a set of tuples 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 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 4 shows the various steps in the data cube construction algorithm. Each step is explained in the next few subsections.

Algorithm 1 *Parallel data cube construction and operations*

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

---

Figure 4: Steps in parallel data cube construction

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.

5.1 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. The algorithm is illustrated in Figure 5 with communication steps shown in bold.

Algorithm 2 *Sample based partitioning*

p - Number of processors.
Step 1. Sort tuples on attribute A locally on each processor
Step 2. Pick up a sample of size p of attribute A from each processor. **Gather** the sample on processor 0 and sort it locally.
Step 3. Pick up $p-1$ splitters from this list on processor 0 and **Broadcast** them.
Step 4. Using the splitters each processor divides it's tuples into p sorted partitions with all values of attribute A in a lower partition being lower than a higher numbered partition.
Step 5. Using **All-to-All communication** each partition is sent to its destination processor.
Step 6. Each processor merges the sorted partitions obtained from other processors resulting in a set of tuples sorted globally on attribute A.

Figure 5: Sample based partitioning

5.2 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.

5.2.1 Hash-based method

Figure 6 describes the hash-based cube loading algorithm. Each attribute is hashed to get a hash table of 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.

5.2.2 Sort-based method

Figure 7 describes the sort-based cube loading algorithm. This method provides regularity of access over the sorted hash tables since the attributes probing them are sorted, unlike the hash-based method, where

Algorithm 3 *Hash-based data cube loading Algorithm*

k - number of attributes(dimensions)
Step 1. For each tuple, 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 tuple 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 tuple.

Figure 6: *Hash-based* Algorithm for multidimensional data cube Loading

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.

Algorithm 4 *Sort-based data cube loading Algorithm*

k - Number of attributes(dimensions)
Step 1. Sort tuples locally on each processor in such a way that B is sorted for each unique value of A, C is sorted for each unique value of B and so on. Note that A is already sorted on each processor.
Step 2. For each tuple 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 3. Compress and sort each hash table.
Step 4. Pick up each tuple and calculate the indices of each attribute in the md-array in the following way. (Assume that the sorting order is A, B and C)
Step 4a. Pick up the attribute at the innermost level of the sort and linearly scan the sorted hashed list for it to pick up the index.
Step 4b. As long as the values at higher levels are the same, scan past the point where the scan stopped last, to find the index. If a attribute value changes, scan the sorted hash lists from the start for all inner attributes.
Step 4c. Update the cell at the index in the md-array with the measure values of the tuple.

Figure 7: *Sort-based* Algorithm for multidimensional data cube Loading

5.3 Aggregation

An example of a base cube is shown in Figure 8 with $p = 4, D_a = 8, D_b = 4, D_c = 2, D_d = 2$. Hence each processor has $\frac{D_a}{p}$ portion of A. We illustrate the costs of calculating the various GROUP-BYs from the base cube for a three attribute (or 3D) cube in Table 3. 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 the previous section. 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. An example of a local aggregate calculation, ABCD $\rightarrow$ ABD, is shown in Figure 9. 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 3, 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 8 shows A as distributed in ABCD.

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	$(\frac{D_a}{p} D_b D_c)t_{op} + (D_b D_c)t_{reduce} + (\frac{D_b}{p} D_c)t_{copy}$
AB $\rightarrow$	A	$(\frac{D_a}{p} D_b)t_{op}$
	B	$(\frac{D_a}{p} D_b)t_{op} + D_b t_{reduce} + \frac{D_b}{p} t_{copy}$
AC $\rightarrow$	A	$(\frac{D_a}{p} D_c)t_{op}$
	C	$(\frac{D_a}{p} D_c)t_{op} + D_c t_{reduce} + \frac{D_c}{p} t_{copy}$
BC $\rightarrow$	B	$(\frac{D_b}{p} D_c)t_{op}$
	C	$(\frac{D_b}{p} D_c)t_{op} + D_c t_{reduce} + \frac{D_c}{p} t_{copy}$
A $\rightarrow$	ALL	$D_a t_{op}$
B $\rightarrow$	ALL	$D_b t_{op}$
C $\rightarrow$	ALL	$D_c t_{op}$

Table 3: Calculation of GROUP-BYs for a three attribute data cube ($D_a \times D_b \times D_c$), on p processors

Figures 10 and 11 illustrate a global aggregate calculation of calculating $ABCD \rightarrow BCD$. First a local aggregation is done along dimension A (Figure 10). This is followed by a **Reduce** operation on BCD (Figure 11). 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 [13], 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 3. 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 3 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

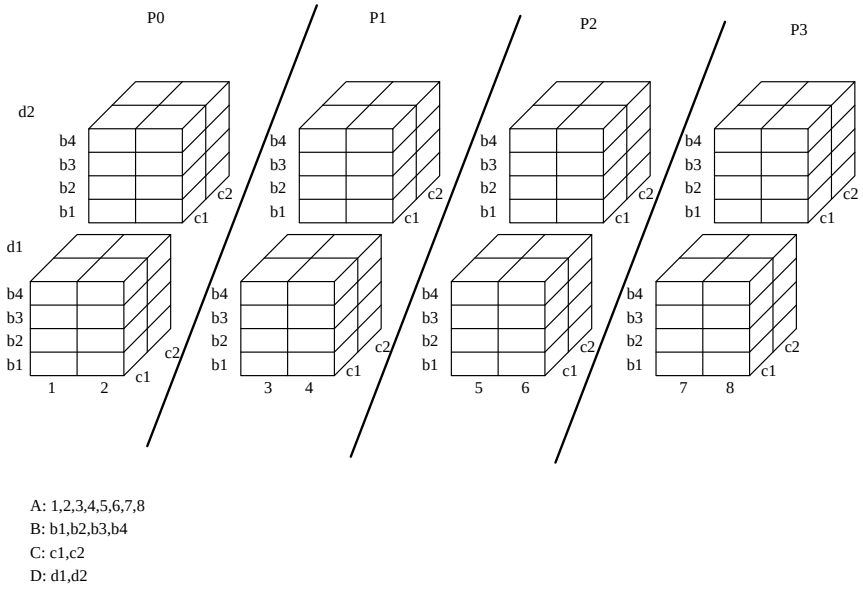


Figure 8: Basecube for 4 attributes on 4 processors

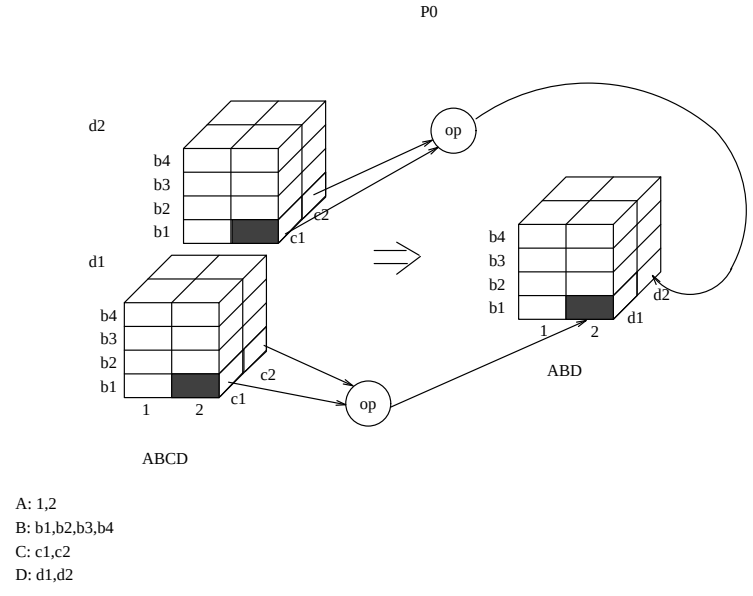


Figure 9: *Local* aggregation calculation, ABCD $\rightarrow$ ABD on processor P0

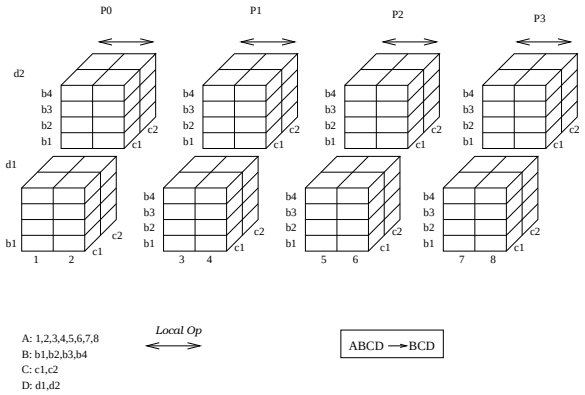


Figure 10: *Global* aggregation calculation, local phase, ABCD → BCD

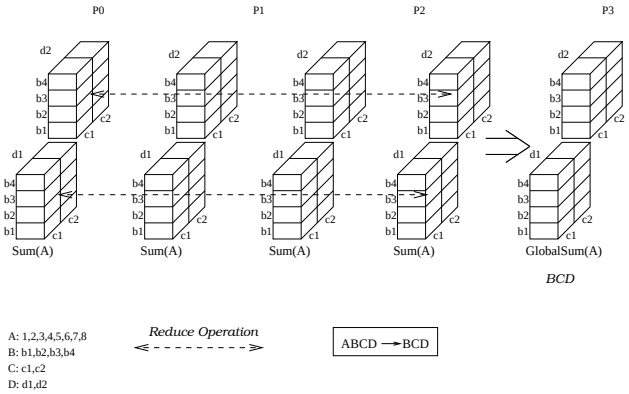


Figure 11: *Global* aggregation calculation, global phase, ABCD → BCD

between nodes from level $k+1$ and level k which minimizes the total edge costs. Figure 12 shows an example of calculating level 2 from level 3 of the lattice of Figure 3. 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.

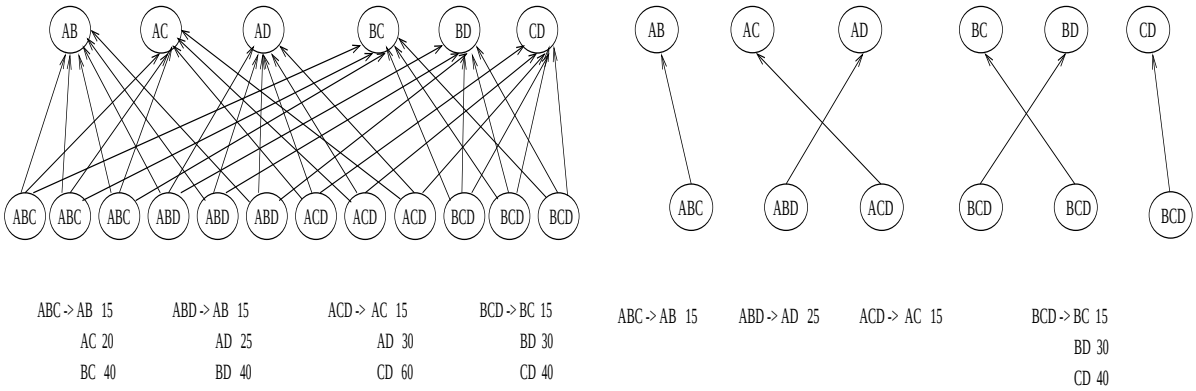


Figure 12: A minimum cost matching for calculation of Level 2 aggregates from Level 3

6 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.

6.1 Data sets

We have used the OLAP council benchmark [15] 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. These are taken to be the size of the dimensions in the resulting multi-dimensional arrays in the sub-cubes of the data cube. For the History Sales data, we have picked out 1000, 101, 100 and 10 distinct values for its 4 attributes from the benchmark data and generated the records by picking out each attribute randomly from the set of distinct values for that attribute. 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 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. The characteristics of the data sets are summarized in Table 4.

Data Set	Dimensions	Number of records	Base cube dimensions	Data size
Shipping Cost	Customer, Time, Scenario	27000	$900 \times 24 \times 2$	1MB
Production Cost	Product, Time, Scenario	270,000	$9000 \times 24 \times 2$	10MB
Current Sales	Product, Customer, Channel	81000 and 608866	$9000 \times 900 \times 9$	320MB
Budget	Product, Customer, Time	951,720	$9000 \times 900 \times 12$	420MB
History Sales	Product, Customer, Time, Channel	1,010,000	$1000 \times 101 \times 100 \times 10$	500 MB

Table 4: OLAP benchmark data sets used for data cube construction

6.2 Data cube Construction

We present results for the different phases of data cube construction for all the data sets described above. Figure 13 shows the performance of hash-based and sort-based methods for the Shipping Cost data with 27000 records. We observe that the algorithm scales well for this small data size. Each individual component scales well also. The two methods take almost the same time for the data cube construction.

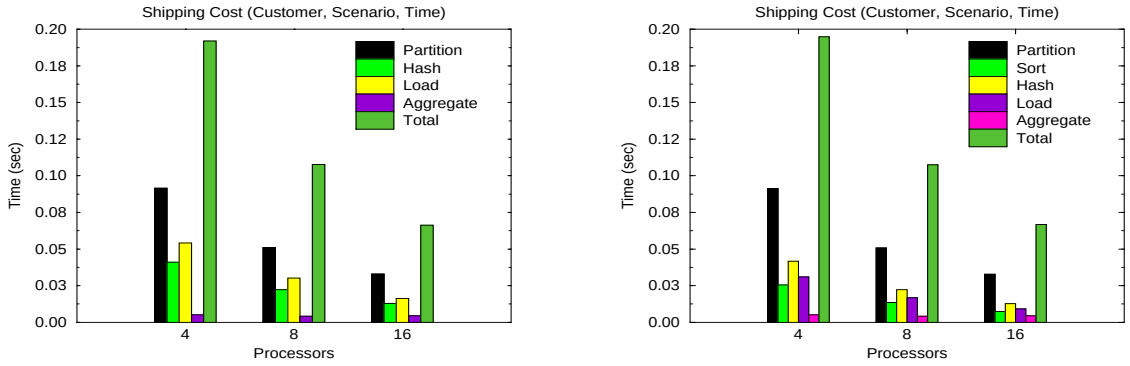


Figure 13: Various phases of cube construction using (a) hash-based (b) sort-based method for Shipping Cost data (N = 27000 records, p = 4, 8 and 16 and data size = 1MB)

Production Cost data contains 270,000 records and the performance of the two methods on this data set is shown in Figure 14. 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 15 shows the performance of hash-based and sort-based methods for the Current Sales data with 81000 records. This data has a density of 0.11%.

The aggregation time is the main component in the total time because of the large cube size for this data set. Figure 16 shows the performance of hash-based and sort-based methods for the Current Sales data with 608,866 records. This data has a density of 0.84%. All the individual components have increased except the aggregation component. The number of tuples has increased in this data set which affects the components till the loading phase. Aggregation times are dependent on the cube size, which has not changed.

Figure 17 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.

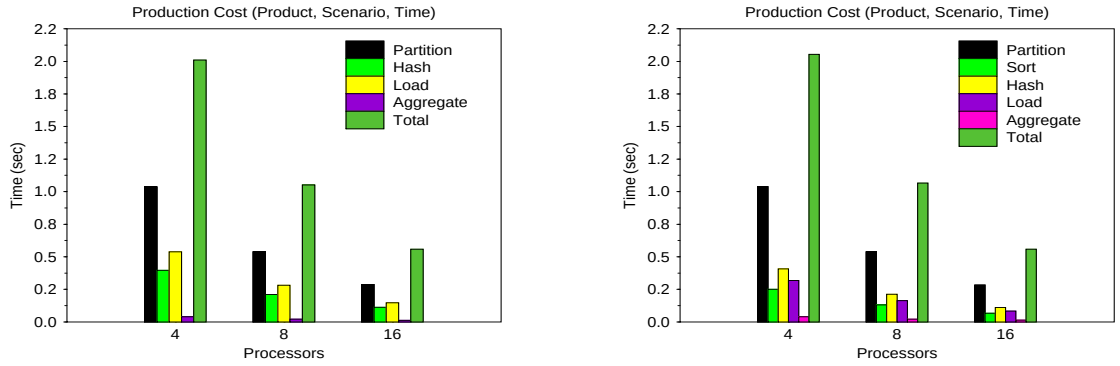


Figure 14: Various phases of cube construction using (a) hash-based (b) sort-based method for Production Cost data (N= 270,000 records, p = 4, 8 and 16 and data size = 10MB)

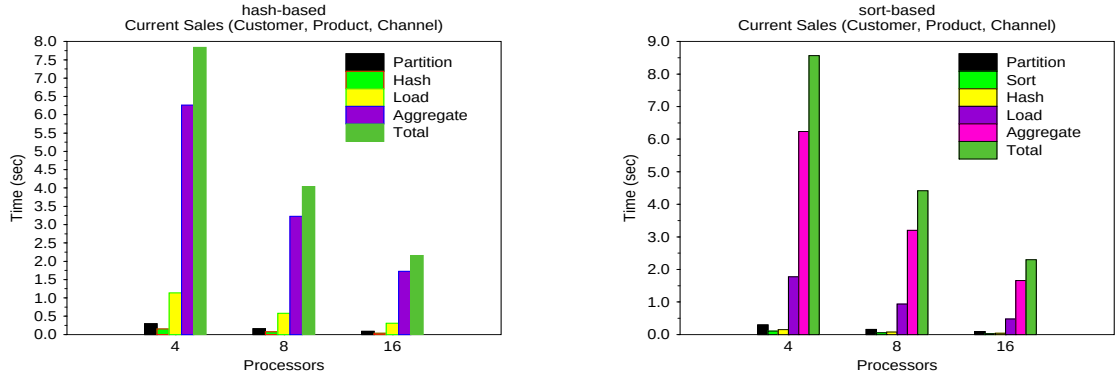


Figure 15: Various phases of cube construction using (a) hash-based (b) sort-based method for Current Sale data (N = 81,000 records, p = 4, 8 and 16 and data size = 320MB)

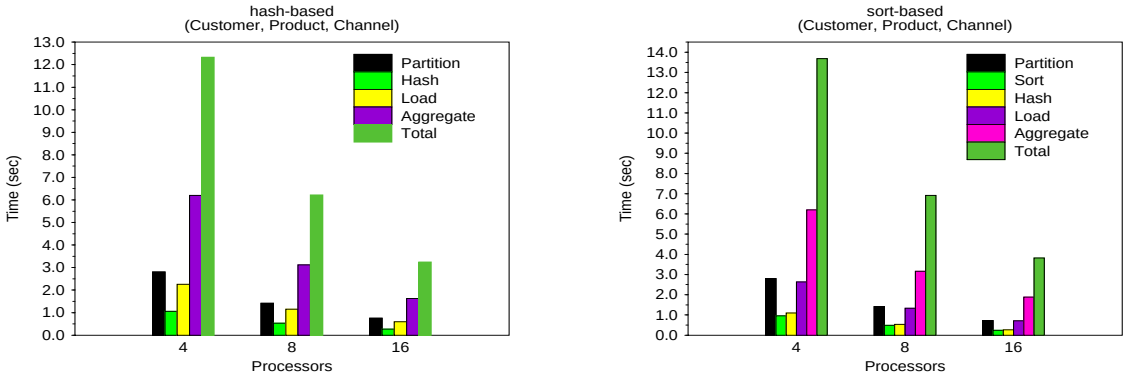


Figure 16: Various phases of cube construction using (a) hash-based (b) sort-based method for Current Sale data (N = 608,866 records, p = 4, 8 and 16 and datasize = 320MB)

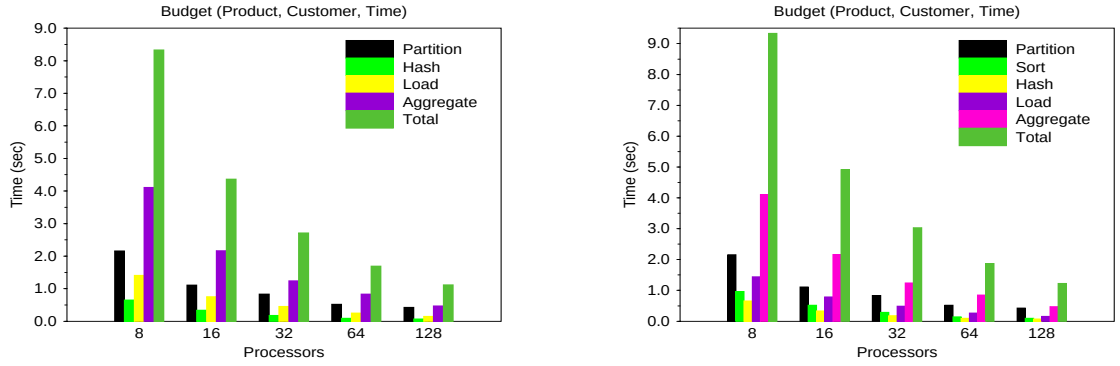


Figure 17: Various phases of cube construction using (a) hash-based (b) sort-based method for Budget data (N = 951,720 records, p = 8, 16, 32, 64 and 128 and data size = 420MB)

Figure 18 gives the performance on History Sales data containing 4 attributes (4 dimensional base cube) with more than a 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.

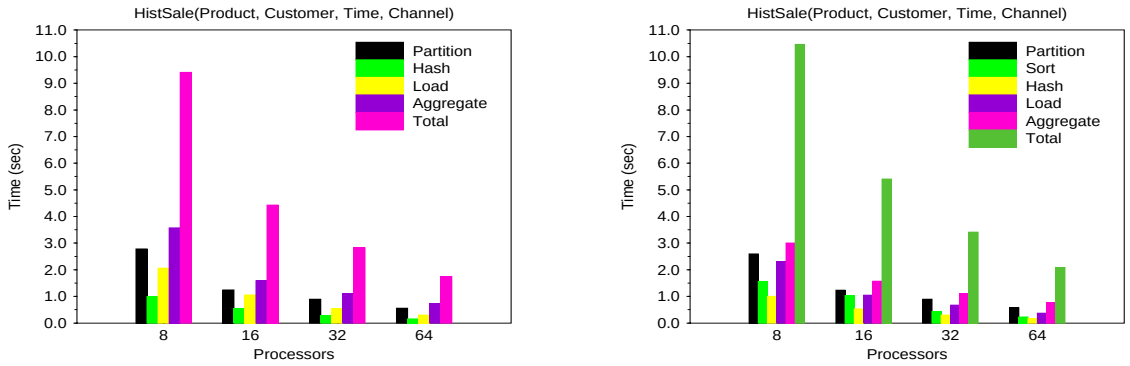


Figure 18: Various phases of cube construction using (a) hash-based (b) sort-based method for History Sales data (N = 1,010,000 records, p = 8, 16, 32 and 64 and data size = 500MB)

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 19 and Figure 20. 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. The scalability for both densities, and both hash-based and sort-based methods is good as observed from the figures.

Comparing the sort-based methods and the hash-based methods, we observe that the hash-based method performs slightly better for 3D cubes but a lot better for the 4D case. We have used a multi-dimensional

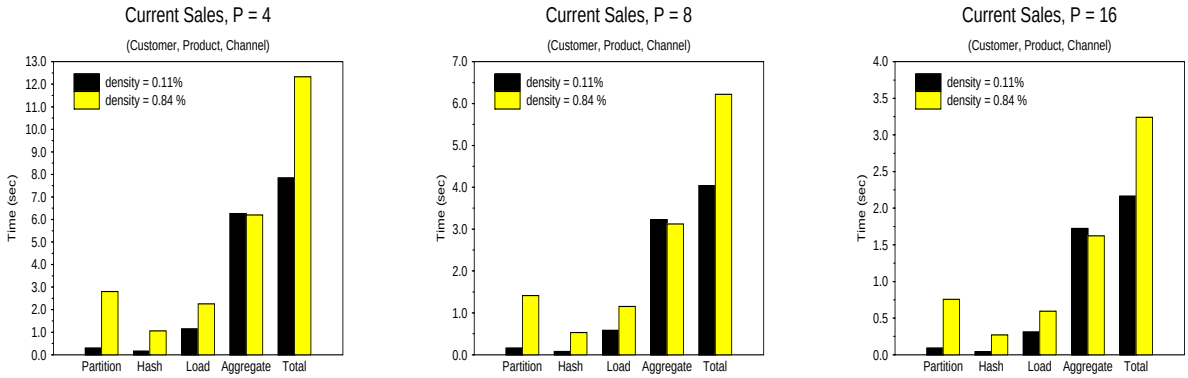


Figure 19: Comparison of cube construction components for two different densities for the hash based method

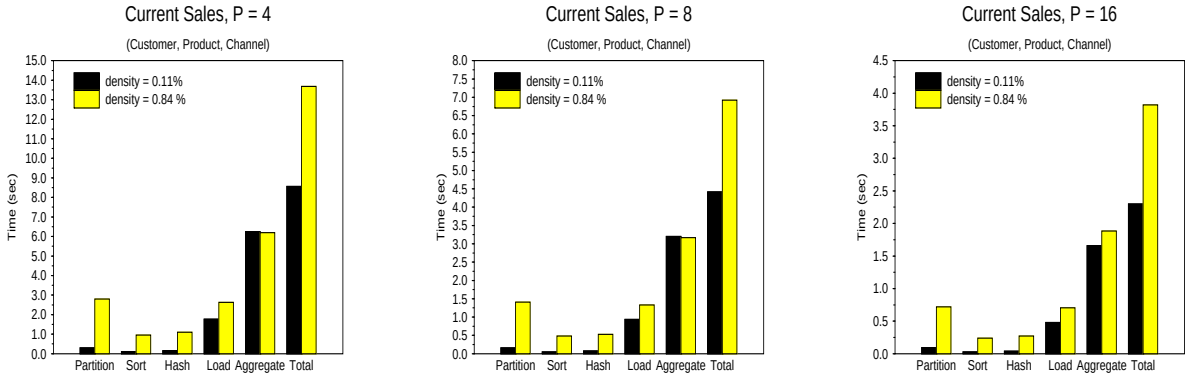


Figure 20: Comparison of cube construction components for two different densities for the sort based method

array and the outermost dimension of the array (the one changing the slowest) does not match the inner-most sorted-dimension. The inner-most dimension is used for the attribute with the largest number of unique values to make the aggregate calculations efficient. The sort-based loading algorithm starts with the inner-most attribute in the sorted order which is also the smallest dimension. This also happens to be the outermost dimension of the array having the maximum stride. This is done to make the accesses during the aggregate calculation regular in the largest dimension, which is the innermost dimension. Chunking of arrays [14] can make the memory accesses for the sort-based method more regular since it does not favor any particular dimension for contiguous storage.

7 Aggregate Query Processing

Most data dimensions have hierarchies defined on them. Typical OLAP queries probe summaries of data at different levels of the hierarchy. Consolidation is a widely used method to provide roll-up and drill-down functions in OLAP systems. Hierarchies can be used for data mining at multiple levels in attribute oriented approaches as discussed in a later section. Attribute-oriented induction [9] is performed for association rule discovery by ascending a predefined concept hierarchy. Each dimension in a cube can potentially have a hierarchy defined on it. This hierarchy can be applied to all the sub-cubes of the data cube. We discuss each of these in the following subsections.

7.1 Hierarchies

Figure 21 shows an example of a hierarchy for a 3D base cube with (Location, Time, Person) as the attributes (dimensions). Assume that hierarchies (City, Quarter, Group) are defined on these dimensions. Each dimension represents an attribute and each value in that dimension is a distinct value for that attribute. A hierarchy describing the grouping of some of the attribute values is defined in a *dimension hierarchy*. A hierarchy can have several levels. Figure 21 defines a first level hierarchy on each dimension where $(a_1, a_2, a_3) \in C1$, $(b_1, b_2) \in Q1$, $(c_1, c_2) \in G1$ and so on. The distributed dimension can have its hierarchy distributed over a processor boundary. A global index of the hierarchy is maintained at each processor. For example, cities in **C3** are distributed across P_0 and P_1 in the figure, containing the indices for (a_7, a_8, a_9). A consolidation operation would need a inter-processor communication if the consolidated value of $C3$ is required. Hierarchy at higher levels can be defined similarly. Support for roll up operations using consolidation can then be provided along each dimension. Drill downs can use them to index into the appropriate dimension index to provide a higher level of detail.

A hierarchy is defined over a dimension in the base cube. Hence the sub-cubes of the data cube can use the same hierarchy, since the dimensions in the sub-cubes are a subset of the dimensions in the base cube. Additionally, more dimensions can get distributed over the processors, and a distributed hierarchy needs to be maintained for these dimensions as well. In our example, in the cube (Time, Person) a 2D cube calculated from (Location, Time, Person), Time is a distributed dimension on which a distributed hierarchy needs to be maintained. As a result, each dimension has two hierarchies for it, one *local* and one *distributed*. Queries will choose the appropriate hierarchies when performing operations on these cubes.

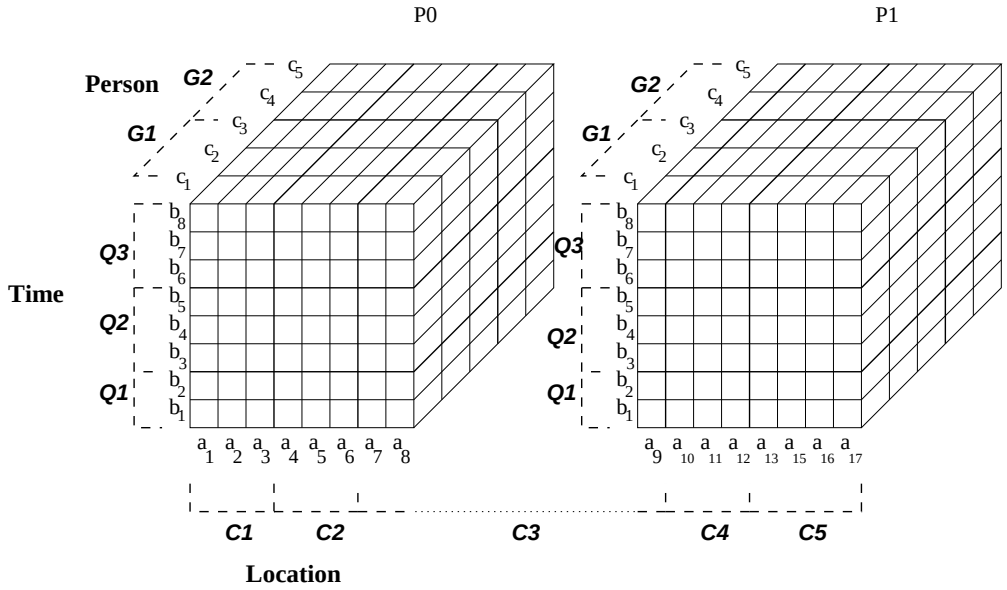


Figure 21: An example hierarchy defined on the dimensions for a two processor case

7.2 Consolidation

Consolidation provides an aggregate for the hierarchies defined on a dimension. For example, if it is desired to report the value for group **G1** for a city **C1** in the quarter **Q1**, then the corresponding values are aggregated by using the *dimension hierarchy* defined in the previous section. Some consolidation operations are *local*, as in they can be performed at a single processor, like **C4** in the figure. However, some operations are *non-local*, like the values for city **C3** in any quarter will need the aggregates values from both the processors as shown in Figure 22. A inter-processor communication is needed to further aggregate the data to get the correct result.

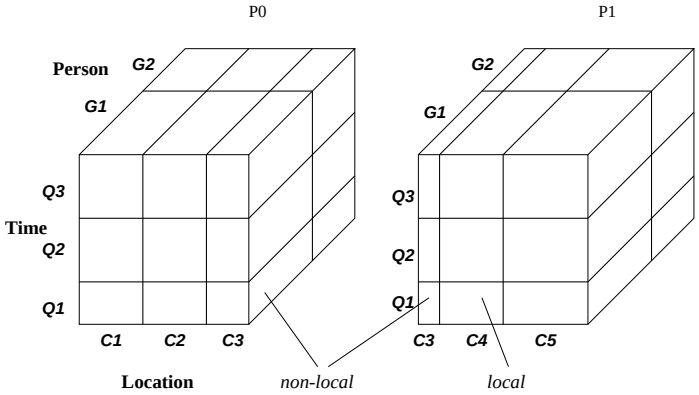


Figure 22: Consolidation for a level 1 hierarchy defined on the cube in Figure 21

7.3 Results

We again use the OLAP council benchmarks described earlier and define some sample hierarchies on each dimension. Table 5 defines a sample hierarchy for the first two levels. For our experiments we have divided the

Dimension	Hierarchy	
	Level 1	Level 2
Customer	9	4
Product	8	3
Channel	4	2
Time	8	2

Table 5: Sample level 1 and level 2 number of groups for hierarchies defined on the dimensions

size of a dimension equally among the number of groups defined for the hierarchy at that level. The allocation of actual attribute values to the groups at a higher level is contiguous for these experiments. However, these can be arbitrarily assigned, if desired.

Figure 23 shows the times for performing consolidation operation using the hierarchies on the dimensions. For Current Sales we have used the level 1 hierarchy on Product and Customer. For History Sales data, the hierarchy is used for all the dimensions, i.e. Product, Customer, Time and Channel. The figure for Budget also includes hierarchies along all dimensions, Product, Customer and Time. We see good scaling properties for each data set as the number of processors is increased.

Comparing the consolidation time for Current Sales and Budget, we observe that the time for Budget data is lower even though their base cubes are similar in size. The presence of a hierarchy along the Time dimension, which is the outermost dimension in the multi-dimensional array, improves the cache performance. Chunks of size $\lceil \frac{12}{8} \rceil \times \frac{900}{9} \times \frac{9000}{8}$, from outermost to innermost dimension, are consolidated for Budget, whereas chunks of size $9 \times \frac{900}{9} \times \frac{9000}{8}$ are consolidated for Current Sales. However, both scale well as the number of processors is increased.

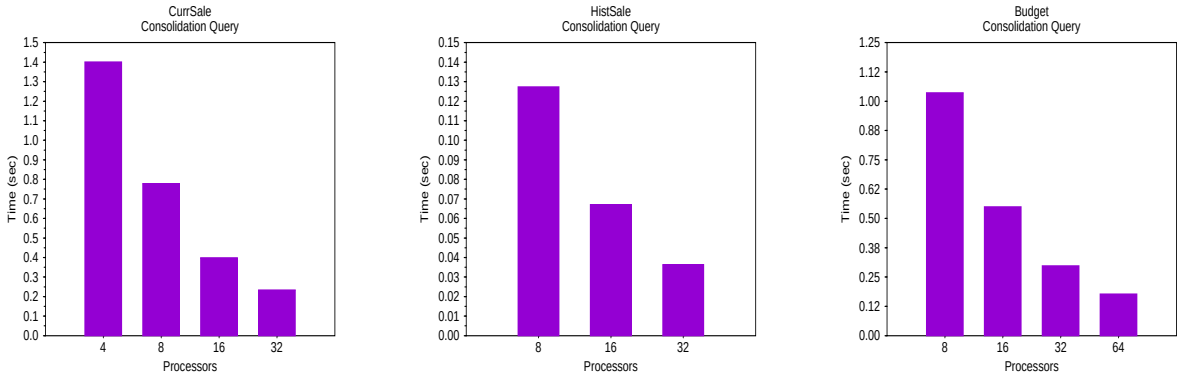


Figure 23: Time for consolidation query using the hierarchies defined in Table 5 for Current Sales, History Sales and Budget data sets

8 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] [9] [10] 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 sub-populations. 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.

8.1 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 24 shows the algorithm.

Algorithm 5 *Calculation of interestingness using data cubes*

Step 1. Replicate each single attribute sub-cubes on all processors using a **Gather** followed by a **Broadcast**.

Step 2. Perform a **Reduce** operation of ALL (0D cube) followed by a **Broadcast** to get the correct value of ALL on all processors.

Step 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.

Step 4. For each element i in AB calculate $|P(AB) - P(A)P(B)|$, and compare it with a threshold δ, setting AB[i] to 1 if it is greater, else set it to 0.

Figure 24: Algorithm for calculating interestingness between attributes A and B with data cubes

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 25 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.

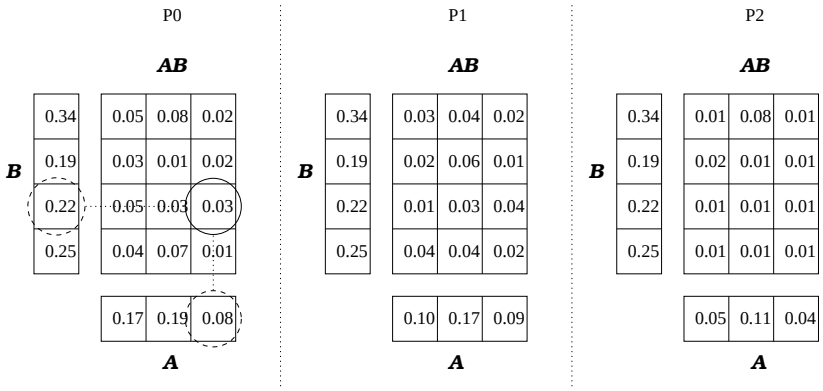


Figure 25: Use of aggregate sub-cubes AB, A and B for calculating "interestingness" on three processors

8.2 Results

We have used a few data sets from the OLAP council benchmark. In particular, Current Sales (3D), Budget(3D) and History Sales (4D)(as described in previous sections).

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 26 shows the time for these calculations on Current Sales data on 4, 8, 16 and 32 processors. We observe good speedups as the number of processors is increased. Also, communication time increases as the number of processors is increased. Replication of the 1D cubes involve communication, and with larger number of processors, the terms involving p in the complexity of the collective communication operations (refer to Table 2) increase. However, since communication time is marginal compared to computation times, the increase does not affect the overall scalability.

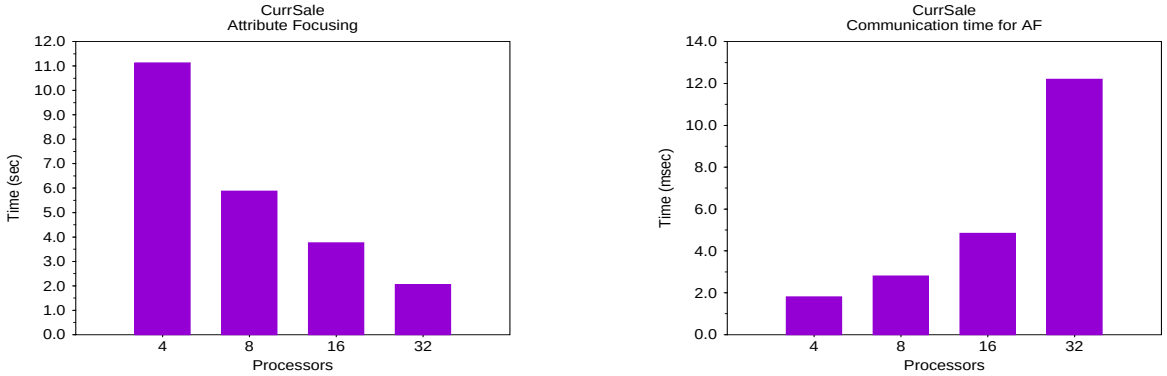


Figure 26: Time for (a) attribute focusing time for 20 different δ values for Current Sales data, (b) associated communication cost (in milliseconds) on 4, 8, 16 and 32 processors

Figure 27 shows the attribute focusing calculation time for the History Sales data. The associated communication time is slightly higher in this case since 4 1D arrays are replicated, instead of 3 in the 3D case. The communication time (which is shown in milliseconds) is still at least an order of magnitude smaller than the computation time. The size of the cubes is smaller than the Current Sales data and hence the magnitudes of time are much smaller. Hence, the speedups are more modest as we increase the number of processors.

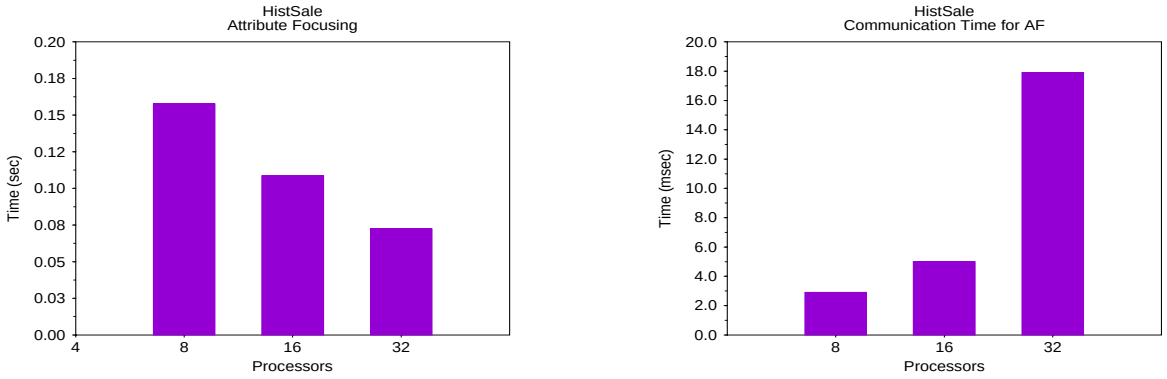


Figure 27: Time for (a) attribute focusing for 20 different δ values for History Sales data, (b) associated communication cost (in milliseconds) on 8, 16 and 32 processors

Figure 28 shows the speedups we observe for the Budget data for attribute focusing calculations. The

communication costs are similar to the ones observed in the Current Sales data since they involve similar sized arrays.

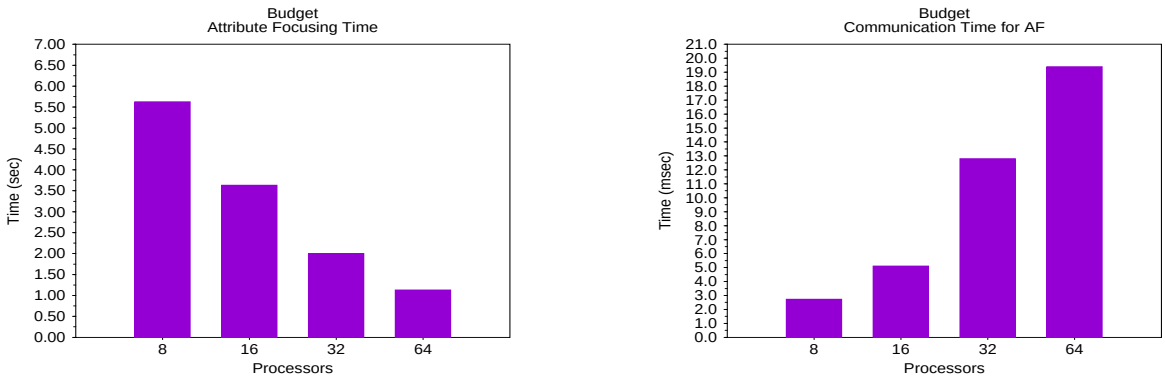


Figure 28: Time for (a) attribute focusing for 20 different δ values for Budget data, (b) associated communication cost (in milliseconds) on 8, 16, 32 and 64 processors

9 Conclusions

On-Line Analytical Processing is fast gaining importance for business data analysis using large amounts of data. High performance computing is required to provide efficient analytical query processing in OLAP systems. Aggregations are an important function of OLAP queries and can benefit from the data cube operator introduced in [7]. Multidimensional databases have recently gained importance since they model the multi-dimensionality of data intuitively and are easy to visualize. They provide support for complex analytical queries and are amenable to parallelization. Most data to be analyzed comes from relational data warehouses. Hence multidimensional databases need to interface with relational systems. In this article, we presented algorithms and techniques for constructing multidimensional data cubes on distributed memory parallel computers and perform OLAP and data mining operations on them. The two methods for base cube loading, sort-based and hash-based, perform equally well on small data sets but the hash-based method is better for larger data sets and with increase in number of dimensions. We would expect the sort-based method to outperform the hash-based method for large amounts of data due to the regularity of accesses that cube loading from sorted data provides. The mismatch in the order of the dimensions when using a multidimensional array does not make the memory usage of the sort-method very efficient. Sort-based method is expected to be better for external memory algorithms because it will result in reduced disk I/O over the hash-based method. Since OLAP operations frequently require information at varying degree of detail, we have developed methods to perform consolidation queries by defining levels of hierarchies on each dimension. Our results show that these techniques are scalable for a large number of processors. Finally, we have shown a implementation of scalable data mining algorithm using attribute focusing. Our techniques are platform independent and are portable across a wide variety of parallel platforms without much effort.

We are currently investigating efficient sparse methods for multidimensional sparse arrays, disk-based algorithms and techniques that apply parallel I/O optimizations to perform OLAP and data mining.

10 Acknowledgments

We would like to thank Inderpal Bhandari of IBM T.J Watson Research Center for discussions and helpful comments.

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", To appear in the 4th International Conference on High Performance Computing, Bangalore, India.

[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. International Conference on Data Engineering, 1996.

[8] Guting A., "An Introduction to Spatial Databases", VLDB Journal, 3, 1994, pp. 357-399.

[9] 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.*

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

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

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

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

[14] Sarawagi S. and Stonebraker M., "Efficient Organization of Large Multidimensional Arrays", Proc. of the Eleventh International Conference on Data Engineering, Houston, February 1994.

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

[16] Zhao Y., Tufte K. and Naughton J., "On the Performance of an Array-Based ADT for OLAP Workloads", Technical Report, University of Wisconsin, Madison, May 1996.