Partial Data Cubes for Parallel Data Mining *

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

Abstract

Summary information from data in large databases is used to answer queries in On-Line Analytical Processing (OLAP) systems and to build decision support systems over them. Further, intermediate computations stored in such systems can be used for data mining tasks like discovery of association rules. Queries posed on such systems are quite complex and require different views of data. Analytical models need to capture the multi-dimensionality of the underlying data, a task for which multi-dimensional databases are well suited. Also, they are amenable to parallelism, which is necessary to deal with large (and still growing) data sets. Multi-dimensional databases store data in multi-dimensional arrays on which analytical operations are performed. While this is good to store dense data, most data sets in practice are sparse. Various schemes have been used for sparse data storage and access in the literature, there being a trade-off between storage and access of data. Complex operations, such as required for knowledge discovery and data mining can be very expensive in terms of data access time if efficient data structures are not used.

In this paper we present *(1)* a parallel OLAP infrastructure for multidimensional databases. *(2)* A multidimensional data cube used to store summary data, computed partially for larger number of dimensions. *(3)* Methods for parallel OLAP queries and data mining of association rules performed using the cube. Attribute-oriented mining of 2-way association rules needs calculations that are already available in the lower levels of the data cube. *(4)* Efficient parallel algorithms for the scheduling and construction of the upper levels of the partial data cube and its use in mining 2-way association rules within the OLAP framework. *(5)* We propose *left* and *right* schedules built from the cube lattice as optimizations for minimizing intermediate *non-essential* cube calculations and inter-processor communication. Our implementation can handle large data sets and a large number of dimensions by using disk I/O in our algorithms. We have proposed a bit-encoded sparse structure(BESS) which we use for storing multi-dimensional sparse data in chunks. Query operations are performed using bit operators on the data structure. *(6)* Results of our experiments on large data sets with upto 20 dimensions are presented on a shared-nothing parallel computer (IBM SP2).

Keywords: High Performance, Multidimensional, Attribute Focusing, BESS

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

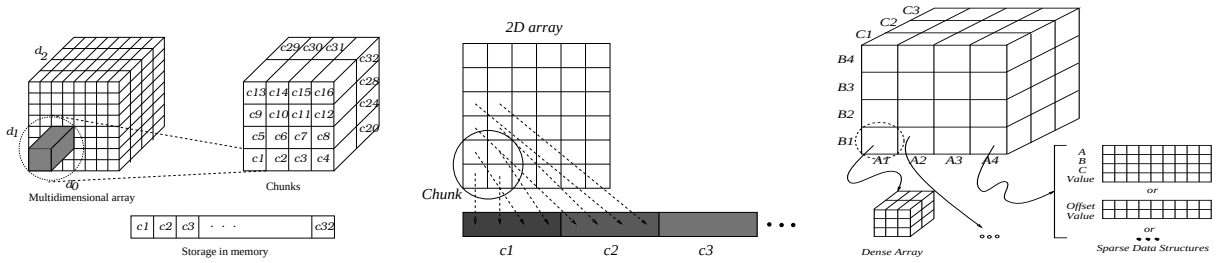
1 Introduction

Multi-Dimensional databases are suitable for On-Line Analytical processing (OLAP) and data mining since these applications require dimension oriented operations on data. In traditional OLTP applications, the need is to support a large number of small transactions, mainly of the type of storing and retrieving information. In contrast OLAP and data mining analyze large amounts of data from data repositories, called *data warehouses*. Analysis of data and discovery of patterns and trends from it can greatly help in making better business decisions.

Decision support systems use OLAP and data mining to find interesting information from large databases. Queries posed on such systems are quite complex and require different views of data. Analytical models need to capture the multi-dimensionality of the underlying data, a task for which multi-dimensional databases are well suited. Traditional multi-dimensional databases store data in multi-dimensional arrays on which analytical operations are performed. Multi-Dimensional arrays are good to store dense data. However, most datasets are sparse in practice for which efficient storage schemes are required. For large databases, reduced storage requirements will reduce the I/O time needed for data accesses during operations. However, there is a trade-off between storage and efficient access of data. Each cell in a multi-dimensional array can be accessed by an index in each of its dimension. Hence, finding an element in it is a trivial operation. If data is sparse most of the cells will remain unused resulting in wasted memory space. On the other hand, sparse data structures can provide efficient storage mechanisms, but locating an element in the sparse data structure requires some additional work. Multi-dimensional databases can benefit from sparse structures to efficiently store this data provided they are efficient in data retrieval. It is important to weigh the trade-offs involved in reduction in storage space offered versus the increase in access time for each sparse data structure when compared with multi-dimensional arrays. These trade-offs are dependent on many parameters some of which are (1) number of dimensions, (2) sizes of dimensions and (3) degree of sparsity of the data. Complex operations such as required for data-driven data mining can be very expensive in terms of data access time if efficient data structures are not used. We compare the storage and operational efficiency in OLAP and data mining of various sparse data storage schemes in [GC97b]. A novel data structure using bit encodings for dimension indices called Bit-Encoded Sparse Structure (BESS) is also introduced there.

In this paper we introduce a parallel multidimensional OLAP framework for large data sets integrated with attribute-oriented data mining of association rules. Techniques for calculating a partial data cube in parallel and optimizations required for parallel processing are presented. Sparsity is handled by using compressed *chunks* using a bit-encoded sparse structure (BESS). Optimizations for scheduling and construction of a partial datacube for 2-way association rules are presented which minimizing the intermediate, *non-essential* computations. We present the *left* and *right* schedules and contrast their performance. Finally, some results of our disk-based implementations are presented on the IBM SP2 for upto 20 dimensions. Due to space limitations only a few results are presented here.

The rest of the paper is organized as follows. Section 2 describes multi-dimensional databases and arrays and introduces the bit-encoded sparse structure. Section 3 describes data mining on the data cube. Section 4 presents the scheduling and construction algorithm of the upper levels of a data cube. Section 5 presents results of our implementation on the IBM SP2. Section 6 concludes the paper.



(a) Chunking for a 3D array (b) Storage for a 2D array (c) Chunked storage for the cube

Figure 1: Storage of data in chunks

2 Multi-dimensional Data Model

Multi-dimensional database technology facilitates flexible, high performance access and analysis of large volumes of complex and interrelated data [Sof97]. It is more natural and intuitive for humans to model a multi-dimensional structure. A cell position in a k-dimensional array is given by an index containing an offset in each of the k dimensions. Access to an element is done by using these index values which makes the calculation of the position of the data element very efficient. A multi-dimensional array can be stored in memory in row major, column major or chunked order.

A *chunk* is defined as a block of data from the multi-dimensional array which contains data in all dimensions. A collection of chunks would then define the entire array. Figure 1(a) shows chunking of a three dimensional array. A chunk is stored contiguously in memory and data in each dimension is strided with the dimension sizes of the chunk. Chunk storage for a two dimensional array is shown in Figure 1(b).

Multi-Dimensional arrays are good structures for dense data, in which most cells are occupied by valid data elements. To store sparse data, space is wasted since many cells will remain unused. Most sparse data may not be uniformly sparse. Dense clusters of data can be found and stored as multi-dimensional arrays. Sparse data structures are needed to store the sparse portions of data. These chunks can then either be stored as dense arrays or stored using an appropriate sparse data structure as illustrated in Figure 1(c). Chunks also act as an index structure which helps in extracting data for queries and OLAP operations.

Typically, sparse structures have been used for advantages it provides in terms of storage, but operations on data have been performed on a multi-dimensional array which is constructed from it. However, this is not always possible when either the dimension sizes are large or the number of dimensions is large. Since we are dealing with multi-dimensional structures for a large number of dimensions, we will be interested in performing operations on the sparse structure itself. This is desirable to reduce I/O costs by having more data in memory to work on. This is one of the primary motivations for BESS. Details can be found in [GC97b].

3 Data Mining on the OLAP Data Cube

OLAP is used to summarize, consolidate, view, apply formulae to, and synthesize data according to multiple dimensions. 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, multi-dimensional database techniques (multi-dimensional OLAP) have been applied to decision-support applications. Data is stored in multi-dimensional 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 multi-dimensional space represents a tuple, with the attributes of the tuple identifying the location of the tuple in the multi-dimensional space and the measure values represent the content of the cell. Various sparse storage techniques have been applied to deal with sparse data in earlier methods. We have used compressed chunks with BESS and constructed a parallel data cube on a shared nothing parallel architecture.

3.1 Data Cubes

Data can be organized into a data cube by calculating all possible combinations of GROUP-BYs [GBLP96]. This operation is useful for answering OLAP queries which use aggregation on different combinations of attributes. For a data set with k attributes this leads to 2^k GROUP-BY calculations. A data 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.

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.

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.

3.2 Attribute-Oriented 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. Discovery of quantitative rules is associated with quantitative information from the database. The data cube represents quantitative summary

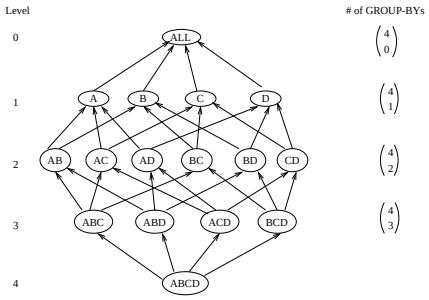


Figure 2: Lattice for cube operator

information for a subset of the attributes. Data mining techniques like *Associations, Classification, Clustering and Trend analysis* [FPSSU94] can be used together with OLAP to discover knowledge from data. Attribute-oriented approaches [Bha95] [HCC93] [HF95] 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. $(P(B|A))$. 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.

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.

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 3 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.

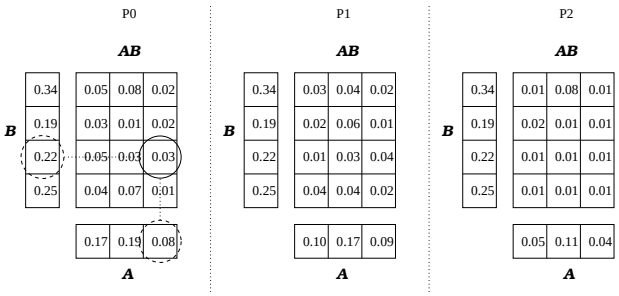


Figure 3: Use of aggregate sub-cubes AB, A and B for calculating "interestingness" on three processors

4 Partial Data Cube Construction

It follows from the previous section that attribute-oriented mining calculations can e performed on the lower m levels of the data cube to find rules with m predicates. We now consider scheduling the construction of a partial datacube for these levels. Since data is distributed over the available processors, we describe data partitioning techniques first.

4.1 Data Partitioning

Data is distributed on processors to distribute work equitably. In addition, a partitioning scheme for multidimensional has to be *dimension-aware* and for dimension-oriented operations have some regularity in the distribution. A dimension, or a combination of dimensions can be distributed. In order to achieve sufficient parallelism, it would be required that the product of cardinalities of the distributed dimensions be much larger than the number of processors. For example, for 5 dimensional data ($ABCDE$), a 1D distribution will partition A and a 2D distribution will partition AB. We assume, that dimensions are available that have cardinalities much greater than the number of processors in both cases. That is, either $|A_i| \gg p$ for some i, or $|A_i||A_j| \gg p$ for some i, j, $1 \leq i, j \leq n$, and n is the number of dimensions. We will use these partitioning in order to study the communication requirements for data movement in the intermediate aggregate calculations in the data cube. Cost parameters for communication in schedule generation will include the data movement involved in aggregate calculation. Figure 4 illustrates 1D and 2D partitions on a 3-dimensional data set on 4 processors.

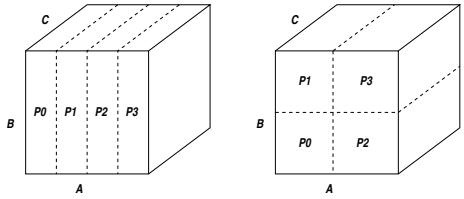


Figure 4: 1D and 2D partition for 3 dimensions on 4 processors

4.2 Schedule Generation for Data Cubes

Several optimizations can be done over the naive method of calculating each aggregate separately from the initial data [GBLP96]. **Smallest Parent**, computes a group-by by selecting 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 next optimization is to compute the group-bys in an order in which thee 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. An important multi-processor optimization is to **minimize inter-processor communication**. The order of computation should minimize the communication among the processors because inter-processor communication costs are typically higher than computation costs. For example, for a 1D partition, $BC \rightarrow C$ 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 [HRU96]. 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 2. 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. Details of these techniques for a parallel implementation can be found in [GC97a].

Optimizations mentioned above have been used on uniprocessors in a sort based algorithm *PipeSort* and a hash based algorithm *PipeHash* [AAD+96]. Also, the *Overlap* algorithm presented here overlaps the calculation of cubes from their parents. An array based algorithm which uses a schedule to optimize memory use and disk scans for I/O is proposed in [ZDN97]. A recursive sparse data cube computation algorithm called *Partitioned-Cube* is presented in [RS97].

4.3 Schedule for Aggregates needed in Attribute-oriented Data Mining

Attribute focusing techniques for m-way association rules and meta-rule guided mining for association rules with m predicates in the rule require that all aggregates with m dimensions be materialized. Let us call these the *essential* aggregates. Since data mining calculations are performed on the cubes with at most m dimensions, the complete data cube is not needed. Only cubes up to level m in the data cube lattice are needed. To calculate this partial cube efficiently, a schedule is required to minimize the intermediate calculations. Since the number of aggregates is a fraction of the total number (2^n), a different schedule, with additional optimizations to the ones discussed earlier, is needed to minimize and calculate the intermediate, *non-essential* aggregate calculations, that will help materialize the essential aggregates.

Meta-rule guided mining on a complete datacube and a dynamically materialized datacube has been reported in [KHC97]. In their approach, either a n-dimensional datacube is constructed and used or all m-dimensional datacubes are constructed. The latter approach is shown to be beneficial for

mining of association rules. However, no optimizations are mentioned for calculating these datacubes.

In the following sections we present optimizations for constructing the upper m levels of a data cube for attribute oriented mining of association rules. In addition to the optimizations discussed in previous sections, we identify **minimizing the number of intermediate cubes**, at a level higher than m. Since we would like to handle a large number of dimensions in our multidimensional framework, this is an important area to reduce the number of *non-essential* cube calculations. Consider a n dimensional data set and the cube lattice discussed earlier and a level by level construction of the partial datacube. The set of intermediate cubes between levels n and m are *essential* if they are capable of calculating the next level of essential cubes by aggregating a single dimension. This provides the minimum set at a level $k+1$ which can cover the calculations at level k. This is determined in a top-down approach on the lattice, resulting in the *left* and the *right* schedules presented next.

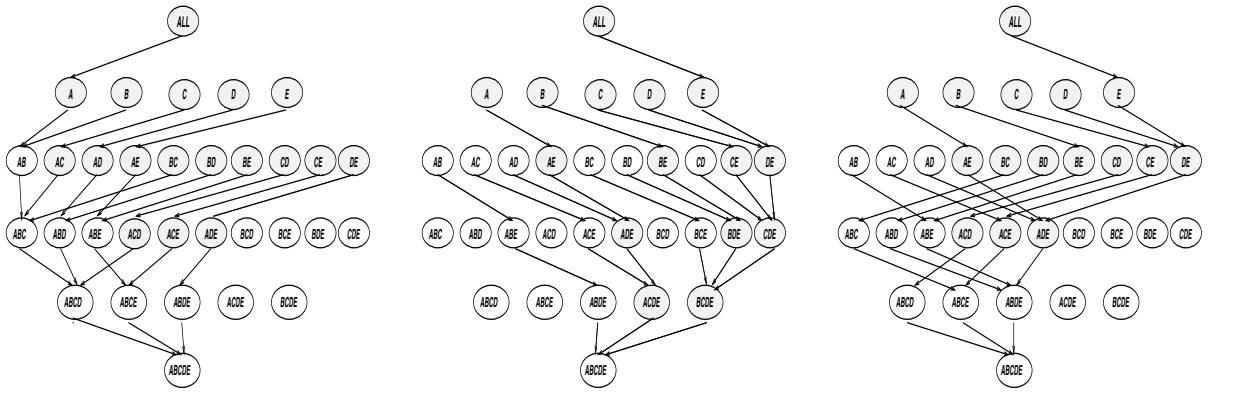
4.3.1 Left and Right Schedules

To obtain the essential aggregates, we traverse the lattice (Figure 2), level by level starting from level 0. For a level k, each entry has to be matched with the an entry from level $k+1$, such that the cost of calculation is the lowest and the minimum number of non-essential aggregates are calculated. We have to determine the lowest number of non-essential intermediate aggregates to be materialized. This can be done in one of two ways. For each entry in level k we choose an arc to the leftmost entry in level $k+1$, which it can be calculated from. This is referred to as the *left schedule*. For example, ABC has arcs to $ABCD$ and $ABCE$, and the left schedule will choose $ABCD$ to calculate ABC. Similarly, a *right schedule* can be calculated by considering the arcs from the right side of the lattice. Of course, there are different trade-offs involved in choosing one over the other, in terms of sizes of the cubes, partitioning of data on processors and the inter-processor communication involved.

Figures 5(a) and (b) show the two schedules for a 2D partitioning (AB is partitioned) for 5 dimensions. Shaded entries are the ones that generate communication. The two schedules have different communication requirements. For a 2D partitioning, $ABCD \rightarrow ABC$ will be a *local* calculation, since D is to be aggregated and each processor can do it on the portion of $ABCD$ it owns, resulting in ABC also being partitioned on AB. However, $ABCD \rightarrow ACD$ requires data movement since one of the distributed dimension is involved in the aggregation, namely B. The resulting aggregate, ACD will be distributed on AC. We maintain 2D distribution for all the aggregates with 2 or more dimensions. 1D partitioning is used for one dimensional aggregates.

Communication in the left schedule can further be reduced by using conforming partitioning between the source (parent) and target (child) aggregates. For example, AC is calculated from ABC in the left schedule which generates communication. Since ACD is also present, it can be used to calculate AC without any communication. The left schedule reduces communication at the higher levels (levels with larger number of dimensions) by preserving conforming partitioning for calculations. The right schedule results in data movement earlier on (at higher levels) which leads to reduced communication at lower levels, i.e with aggregates with lower number of dimensions. Figure 5(c) shows the schedule with reduced communication, by using conformal partitioning in the left schedule. It can be observed that a right schedule applied to the output of the left schedule will result in the needed communication optimizations.

To obtain sufficient parallelism for 2 dimensional partitioning, in a general case, the partitioning dimensions should be chosen such that they provide sufficient parallelism for a given number of processors, p. The rest of the dimensions can then be ordered in a non-increasing order of their



(a) Left (b) Right (c) Left (communication optimized)

Figure 5: Left and Right schedules for 5 dimensions, 2D partitioning, dimensions are ordered in a non-increasing order of their cardinality. Shaded entries represent inter-processor communication

Table 1: Aggregates materialized for the complete data cube and for the partial case where m=2, for n=20

Level	Data Cube	m=2 (Partial)	Level	Data Cube	m=2 (Partial)
1	20	20	11	127960	55
2	190	190	12	105970	45
3	1140	171	13	77520	36
4	4845	153	14	38760	28
5	15504	136	15	15504	21
6	38760	120	16	4845	15
7	77520	105	17	1140	10
8	105970	91	18	190	6
9	127960	78	19	20	3
10	184756	66			

cardinalities.

4.4 Number of aggregates to materialize

In a full data cube, the number of aggregates is exponential. In the cube lattice there are n levels and for each level i there are $\left(\begin{array}{c} n \\ i \end{array} \right)$ aggregates at each level containing i dimensions. The total number of aggregates, thus is 2^n. For a large number of dimensions, this leads to a prohibitively large number of cubes to materialize.

Next, we need to determine the number of intermediate aggregates needed to calculate all the aggregates for all levels below a specified $m, m < n$, for m-way associations and for meta-rules with m predicates.

Table 1 gives the number of aggregates at each level for a data cube with $n = 20$ and for a partial cube which has all 2-dimensional aggregates materialized. The number of aggregates at level 2 is $\sum_{j=1}^{n-1} \left(\begin{array}{c} j \\ 1 \end{array} \right)$. For level 3 it is $\sum_{j=1}^{n-2} \left(\begin{array}{c} j \\ 1 \end{array} \right)$. At each level k, the first $k-1$ literals in

the representation are fixed and the others are chosen from the remaining. The total number of aggregates is then $\sum_{i=1}^{n-1}\sum_{j=1}^{i}\begin{pmatrix} j \\ 1 \end{pmatrix}$. After simplification this becomes $\frac{n(n-1)(2n+2)}{12}$. For example, when $n = 10$, this number is 165, a great reduction from $2^{10} = 1024$. The savings are even more significant for $n = 20$, where this number is 1330, much smaller than $2^{20} = 1048576$.

The base cube is n-dimensional. As intermediate aggregates are calculated by traversing the lattice (converted into a DAG by the left schedule) up from level n, the following happens: 1) Dimensionality decreases at each level, 2) number of aggregates increase and 3) sizes of each aggregate decrease.

Also, for a large number of dimensions it is impractical to calculate the full data cube. A subset of the aggregates can be materialized, and queries can be answered from the existing materializations, or some others can be calculated on the fly. Data cube reductions for OLAP have been topics of considerable research. [HRU96] has presented a greedy strategy to materialize the cubes depending on their benefit.

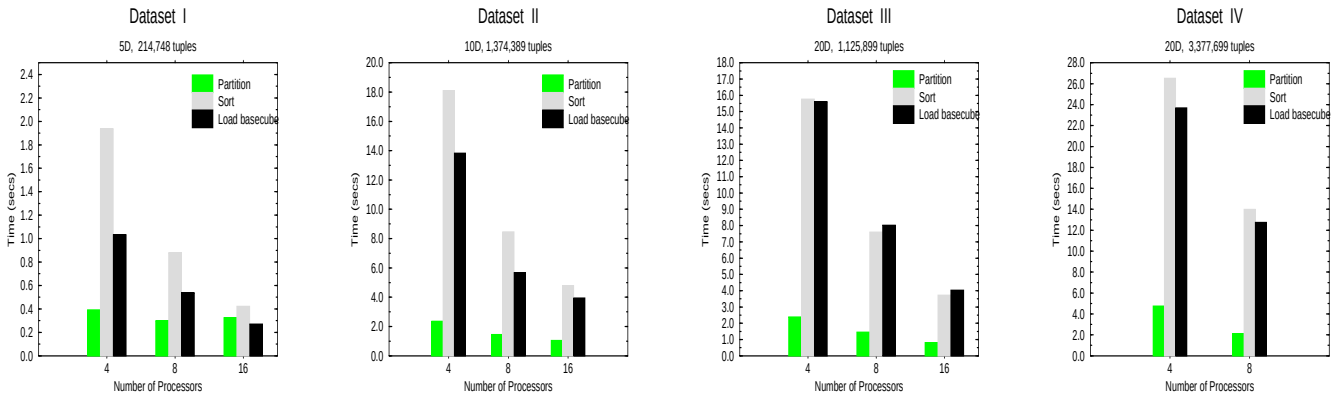
5 Implementation and Results

We have implemented the cube construction and scheduling algorithms on a shared nothing parallel computer, the IBM-SP2 with 16 nodes. Each node is a 120MHz PowerPC, has 128MB memory and a 1GB of available diskspace. Communication between processors is through message-passing and is done over a high performance switch. 'C' along with Message Passing Interface (MPI) for communication is used.

A scalable parallel multidimensional database engine for OLAP and data mining is constructed over this platform. To support large data sets with a large number of dimensions we store data in chunks. BESS is used to store sparse data. Chunks can be dense if the density is above a certain threshold. Disk I/O is used to write and read chunks from disk as and when needed during execution. For each chunk referenced, a small portion of the chunk (called *minichunks*) is allocated memory initially. When more memory is required, minichunks are written to disk and their memory is utilized.

Initial data is assumed to be from a relational database. The following summarizes the steps in the overall process. Assume N tuples and P processors and a 2D partitioning.

1. Each processor reads $\frac{N}{p}$ tuples.
2. Use a **Partitioning** algorithm to obtain a 2D partition of tuples using the largest two dimensions as partitioners.
3. On each processor use a **Sort** algorithm to produce a multi-attribute sort.
4. Use a disk-based **Chunk Loading** algorithm to load sorted tuples in chunks to construct the base cube.
5. Calculate the schedule for partial cube construction using a **Scheduling** algorithm.
6. Calculate the aggregates of the intermediate cubes by an **Cube Aggregation** algorithm.
7. Perform attribute-oriented mining of association rules using a **Attribute Focusing** algorithm.



(a) (b) (c) (d)

Figure 6: (a) Dataset I (5D, 214,748 tuples) (b) Dataset II (10D, 1,374,389 tuples) (c) Dataset III (20D, 1,125,899 tuples) (d) Dataset IV (20D, 3,377,699 tuples)

Details of each step are omitted in this version due to lack of space and present results for only the first few steps of the overall process. These show good performance of our techniques and show that they provide good performance on large data sets and are scalable to larger data sets (larger N) and larger number of processors (larger P).

Three data sets, one each of 5D, 10D and 20D are considered. Table 2 describes the three datasets. The products of their dimensions are $2^{31}, 2^{37}$ and 2^{51} respectively. Different densities are used to generate uniformly random data for each data set to illustrate the performance. A maximum of nearly 3.5 million tuples on 8 and 16 processors is used for the 20D case. Table 2 summarizes the characteristics of the datasets.

Table 2: Description of datasets, (N) Numeric (S) String attribute

Data set	No. of dimensions	Cardinalities (d_i)	Product ($\prod_i d_i$)	Tuples
Dataset I	5	1024(S),16(N),32(N),16(N),256(S)	2^{31}	214,748
Dataset II	10	1024(S),16(S),4(S),16,4,4,16 4,4,32 (N)	2^{37}	1,374,389
Dataset III	20	16(S),16(S),8(S),2,2,2,2,4,4, 4,4,4,8,2,8,8,8,2,4,1024 (N)	2^{51}	1,125,899 and 3,377,699

Performance of partitioning tuples in a 2 dimensional pattern on processors, multi-attribute sorting of tuples for chunk loading and the loading process for 4, 8 and 16 processors is given in Figure 6. We observe good speedups for every data set. Even with around 3.5 million tuples for the 20D set, the density of data is very small. We are using minichunk sizes of 25 BESS-value pairs. With this choice, the data sets used here run in memory and the chunks do not have to be written and read from disk. However, with greater densities, the minichunk size needs to be selected keeping in mind the trade-offs of number of minichunks in memory and the disk I/O activity needed for writing minichunks to disk when they get full. We are currently investigating disk I/O optimizations in terms of the tunable parameters of chunk size, number of chunks and data layout on disk.

6 Conclusions

In this paper we have presented a parallel OLAP infrastructure for multidimensional databases integrated with data mining for association rules. A multidimensional data cube is used to store summary data, computed partially for larger number of dimensions. Parallel OLAP queries and data mining of association rules are performed using the cube. A schedule is presented for the calculation of the upper levels of the partial data cube and its use in mining 2-way association rules within the OLAP framework. Optimizations for minimizing intermediate *non-essential* cube calculations and inter-processor communication are captured in the *left* and *right* schedules built from the cube lattice. Our implementation can handle large data sets and a large number of dimensions by using disk I/O in our algorithms. We have proposed a bit-encoded sparse structure(BESS) which we use for storing multi-dimensional sparse data in chunks. Query operations are performed using bit operators on the data structure. Sample results of our experiments on large data sets with up to 20 dimensions are presented on a shared-nothing parallel computer.

References

[AAD+96] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the Computation of Multidimensional Aggregates. In *Proc 22th Int'l Conference on Very Large Databases, Mumbai, India*, September 1996.

[Bha95] I. Bhandari. Attribute Focusing: Data mining for the layman. Technical Report RC 20136, IBM T.J. Watson Research Center, 1995.

[FPSSU94] U.M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in data mining and knowledge discovery.* MIT Press, 1994.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *Proc. 12th International Conference on Data Engineering*, 1996.

[GC97a] S. Goil and A. Choudhary. High performance olap and data mining on parallel computers. *Journal of Data Mining and Knowledge Discovery*, 1(4), 1997.

[GC97b] S. Goil and A. Choudhary. Sparse data storage schemes for multi-dimensional data for OLAP and data mining. Technical Report CPDC-9801-005, Northwestern University, December 1997.

[HCC93] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(1), February 1993.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the* 21^{st} *VLDB Conference, Zurich*, 1995.

[HRU96] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing data cubes efficiently. In *Proc. SIGMOD International Conference on Management of Data*, 1996.

[KHC97] M. Kamber, J. Han, and J. Y. Chiang. Metarule-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes. In *Proceeding of the 3rd International Conference on Knowledge Discovery and Data Mining, Newport Beach*, August 1997.

[RS97] K. Ross and D. Srivastava. Fast Computation of Sparse Datacubes. In *23rd VLDB Conference*, September 1997.

[Sof97] Kenan Software. An introduction to multi-dimensional database technology. In *http://www.kenan.com/acumate/mddb.htm*, 1997.

[ZDN97] Y. Zhao, P. Deshpande, and J. Naughton. An array-based algorithm for simultaneous multi-dimensional aggregates. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 159–170, 1997.