High Performance Multidimensional Analysis of Large Datasets *

Sanjay Goil Alok Choudhary
Center for Parallel and Distributed Computing,
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. The *Data Cube* is used to calculate and store summary information on a variety of dimensions, which is computed only partially if the number of dimensions is large. Queries posed on such systems are quite complex and require different views of data. These may either be answered from a materialized cube in the data cube or calculated on the fly. Further, data mining for associations can be performed on the data cube. Analytical models need to capture the multidimensionality of the underlying data, a task for which multidimensional databases are well suited. Multidimensional databases store data in multidimensional structure on which analytical operations are performed. A challenge for these systems is how to handle large data sets in a large number of dimensions.

This paper presents q parallel OLAP infrastructure for multidimensional databases integrated with association rule mining. Scheduling optimizations for parallel computation of *complete* data cubes are presented. We propose *left* and *right* schedules for *partial* data cubes for m-way mining of association rules. Our implementation on the IBM SP-2, a shared-nothing parallel machine, can handle large data sets and a large number of dimensions by using disk I/O in our algorithms.

1 Introduction

Decision support systems use On-Line Analytical processing (OLAP) and data mining to find interesting information from large databases. Multidimensional databases are suitable for OLAP and data mining since these applications require dimension oriented operations on data. Traditional multidimensional databases store data in multidimensional arrays on which analytical operations are performed. Multidimensional arrays are good to store dense data, but most datasets are sparse in practice for which other efficient storage schemes are required. Each cell in a multidimensional array can be accessed by an index in each of its dimension. Hence, finding an element in it is a trivial operation. Sparse data structures can provide efficient storage mechanisms, but locating an element in the sparse data structure requires some additional work. It is important to weigh the trade-offs involved in reducing the storage space versus the increase in access time for each sparse data structure, in comparison to multidimensional 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 used to store data in chunks, which supports fast OLAP query operations on sparse data using bit operations without the need for exploding the sparse data into a multidimensional array. This allows for high dimensionality and large dimension sizes. Chunking has been used to store sparse data in compressed format by storing an offset in the chunk for a location [ZDN97]. A sparse chunk is exploded into a multidimensional array to perform OLAP operations which might not be feasible in large dimensions.

Our techniques for multidimensional analysis can be applied to scientific and statistical databases [LS98] but we do not discuss that in this paper.

In this paper we introduce a parallel multidimensional OLAP framework for large data sets which has been integrated with data mining of association rules. Parallel data cube construction for large data sets and a large number of dimensions using sparse storage structures is presented. Sparsity is handled by using compressed *chunks* using BESS. We present techniques for data partitioning, and *sort-based* loading of chunks in the *base* cube which is a n-dimensional structure at level n of the data cube, from which data cubes can be computed. Schedules for constructing complete and partial data cubes which use the base cube are described briefly. Finally, some results of our disk-based implementations are presented on the IBM SP2, for large datasets and up to 20 dimensions.

The rest of the paper is organized as follows. Section 2 describes chunk storage using BESS. Section 3 describes mining of association rules on the data cube. Section 4 presents steps in the computation of the data cube on a parallel machine. Section 5 presents results of our implementation on the IBM SP2. Section 6 concludes the paper.

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

DOLAP '98 Washington DC USA

2 Multidimensional Data Storage and BESS

Multidimensional 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 multidimensional structure. A *chunk* is defined as a block of data from the multidimensional array which contains data in all dimensions. A collection of chunks defines 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. Most sparse data may not be uniformly sparse. Dense clusters of data can be found and stored as multidimensional 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(b). 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 are performed on a multidimensional array which is populated from the sparse data. 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 multidimensional structures for a large number of dimensions, we are 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. For each cell present in a chunk a dimension index is encoded in $\lceil \log |d_i| \rceil$ bits for each dimension d_i, where $|d_i|$ is the cardinality of the dimension d_i. A 8-byte encoding is used to store the BESS index along with the value at that location.

A chunk index structure stores the chunk offsets for each chunk. Chunks are stored in memory in some order of dimensions. To reference an element in the chunk, first the chunk offset is dereferenced to get the dimension index values. Since the dimension extents are known for the chunk in each dimension, this can be done easily. The second step is to add the index value in the chunk to this by retrieving it from the bit encoding. This can be done by appropriate bit shifts and bit masking operations from the BESS encoding. Further details can be found in [GC97b].

3 Data Mining on 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.

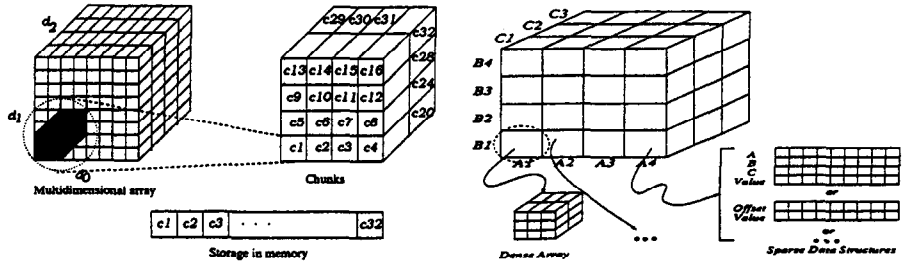
Consider the following query which uses the cube operator.

```
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 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] [KHC97] 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 min-



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

Figure 1: Storage of data in chunks

imum 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.

4 Computing the Data Cube in Parallel

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, $1 \leq i, j \leq n$, n is the number of dimensions. Partitioning determines the communication requirements for data movement in the intermediate aggregate calculations in the data cube. Figure 2 illustrates 1D and 2D partitions on a 3-dimensional data set on 4 processors. We use a 2D partition in our implementations. p processors are divided into two groups $(k, \frac{p}{k})$, where k is the number of groups to be created by the first distributed dimension, chosen to be a divisor of p. For each tuple, A_i is used to choose among the k partitions, and then A_j is further used to find the correct processor among the $\frac{p}{k}$ processors in that group.

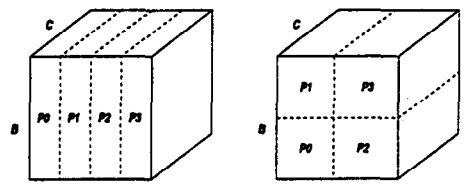


Figure 2: 1D and 2D partition for 3 dimensions on 4 processors

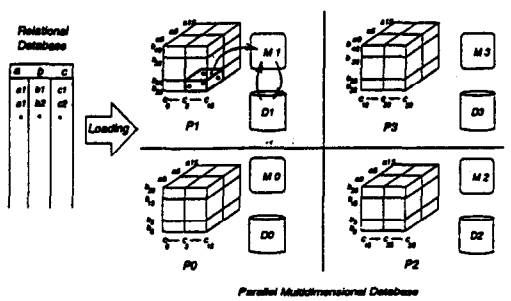


Figure 3: Base cube loading for a 3 dimensional cube on 4 processors

4.2 Base Cube Loading

Figure 3 illustrates the multidimensional base cube loading process from a relational database for 4 processors. Consider N tuples and p processors. $\frac{N}{p}$ tuples are read by each processor. The dimensions with the two largest cardinalities are picked as the partitioning dimensions. Each processor picks a small sample in each of the two dimensions and sorts the two lists globally. $k-1$ and $\frac{p}{k}-1$ splitters are picked respectively from the sorted samples. The splitters for each dimension are then **broadcast** to each processor which use the splitters to distribute their tuples into appropriate partitions for each processor. An **All-to-all Communication** phase does an exchange of the tuples. This results in a 2D partitioning of the tuples.

Each processor then proceeds to construct a local base-cube from the its tuples. The tuples are to be loaded in a parallel multidimensional data structure which stores *sparse chunks* using the bit-encoded sparse structure BESS. A *sort-based* chunk loading algorithm is used to exploit locality in the access of chunks. Sorted runs of the tuples are created by using a *multi-attribute* sort routine. Consider four attributes A, B, C and D, with $|A| > |B| > |C| > |D|$. Tuples are sorted in such a manner that attribute A is sorted, and for each duplicate value of A, B is sorted, within which for each non-unique B, C is sorted, and so on. Chunks are arranged in memory in the order $ABCD$, D being the innermost dimension (contiguous dimension) and A being the outermost (maximum strided dimension). This allows regularity and locality in access of chunks in memory and provides better performance than loading unsorted tuples which would access the the chunks in random order.

4.3 Schedule Generation for Complete and Partial 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 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. 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 five-dimensional cube ($ABCDE$) is shown in Figure 4(a). 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. Some calculations do not involve communication and are *local*, others involving communication are labeled as *non-local*. Details of these techniques for a parallel implementation can be found in [GC97a].

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. These are called *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.

In addition to the optimizations discussed for a complete data cube, 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 multi-dimensional 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 data cube. 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. To obtain the essential aggregates, we traverse the lattice (Figure 4(a)), 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* shown in Figure 4(b). 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 (Figure 4(c)). Further optimizations on the left schedule can be performed by replacing $ABC \rightarrow AC$ by $ACE \rightarrow AC$, since ACE is also being materialized and the latter involves a local calculation. 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.

4.3.1 Number of aggregates to materialize for mining 2-way associations

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 $\binom{n}{i}$ 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.

The number of aggregates at level 2 is $\sum_{j=1}^{n-1} \binom{j}{1}$. For

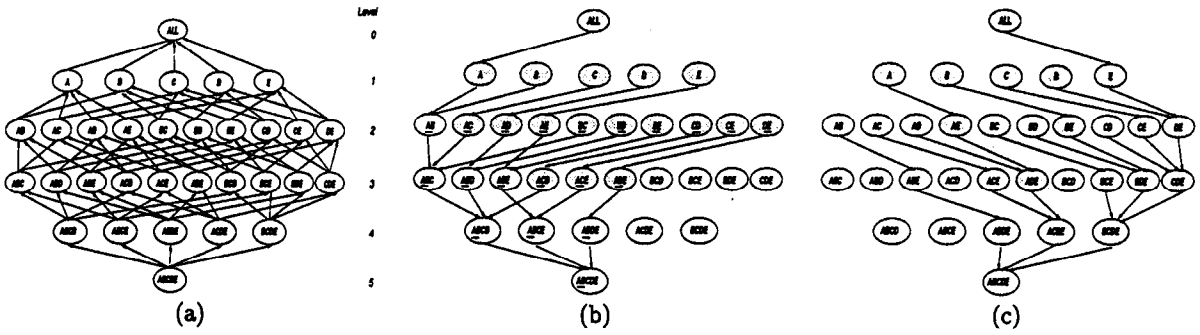


Figure 4: (a) Lattice for cube operator (b) DAG for partial data cube (left schedule) (c) right schedule

level 3 it is $\sum_{j=1}^{n-2} \binom{j}{1}$. 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 $\sum_{i=1}^{n-1} \sum_{j=1}^{i} \binom{j}{1}$. 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 RS6000 processor, 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. Depending on the density of the dataset considered and the data distribution, an appropriate minichunk size should be chosen. Too large a minichunk size will accommodate sparser and fewer minichunks in memory. Too small a size will invoke disk I/O activity, to write the full chunks on to disk. We have experimented with several minichunk sizes and have used a size of 25 (BESS, value) pairs in the performance figures.

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 using the DAG schedule.*
7. *Perform attribute oriented data mining using a* **Attribute Mining** *algorithm.*

Results are presented for partitioning, sorting and chunk loading steps. 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 1 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.

Tuple partitioning using a 2D pattern on processors, multi-attribute sorting of tuples for chunk loading and the sort-based loading process for 4, 8 and 16 processors is given in Figure 5. We observe good speedups for each 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. Multiple cubes of varying dimensions are maintained during the partial data cube construction. Optimizations for chunk management are necessary to maximize overlap and reduce the I/O overhead. 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 with our parallel framework.

Table 1: Description of datasets and attributes, (N) Numeric (S) String

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,		1,125,899
Dataset IV		4,4,4,8,2,8,8,8,2,4,1024 (N)	2^{51}	3,377,699

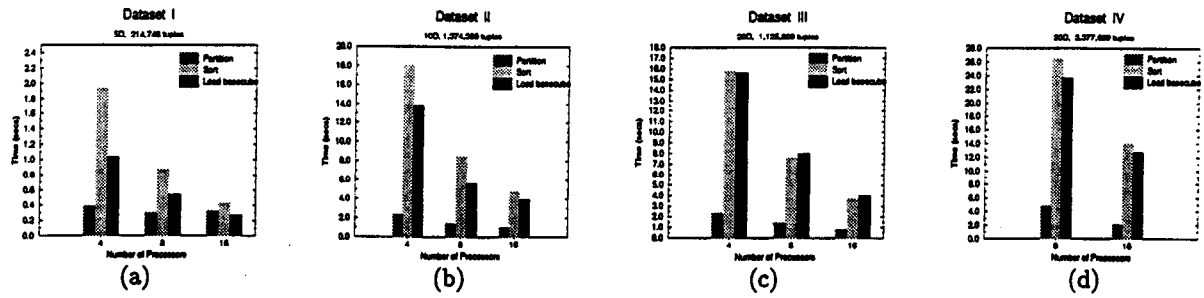


Figure 5: (a) Dataset I (5D, 214,748 tuples) (b) Dataset II (10D, 1,374,389 tuples) (c) Dataset III (20D, 1,125,899 tuples) (d) Dataset IV (20D, 3,377,699 tuples)

6 Conclusions

In this paper we have presented a parallel multidimensional database infrastructure for OLAP and data mining of association rules which can handle a large number of dimensions and large data sets. Parallel techniques are described to partition and load data into a base cube from which the data cube is calculated. Optimizations performed on the cube lattice for construction of the complete and partial data cubes are presented. Our implementation can handle large data sets and a large number of dimensions by using sparse chunked storage using a novel data structure called BESS. Experimental results on data sets with around 3.5 million tuples and having 20 dimensions on a 16 node shared-nothing parallel computer (IBM SP2) show that our techniques provide high performance and are scalable.

References

[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 21st 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.

[LS98] W. Lehner and W. Sporer. On the design and implementation of the multidimensional cube-store storage manager. In *15th IEEE Symposium on Mass Storage Systems (MSS'98)*, March 1998.

[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.