High Performance Multidimensional Analysis and Data Mining *

Sanjay Goil and 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. Also, they are amenable to parallelism, which is necessary to deal with large (and still growing) data sets. 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. These techniques are also applicable to scientific and statistical databases (SSDB) which employ large multidimensional databases and dimensional operations over them.

In this paper we present *(1)* A parallel infrastructure for OLAP multidimensional databases integrated with association rule mining. *(2)* Introduce Bit-Encoded Sparse Structure (BESS) for sparse data storage in *chunks*. *(3)* Scheduling optimizations for parallel computation of *complete* and *partial* data cubes. *(4)* Implementation of a large scale multidimensional database engine suitable for dimensional analysis used in OLAP and SSDB for (a) large number of dimensions (20-30) (b) large data sets (10s of Gigabyte)

Our implementation on the IBM SP-2 can handle large data sets and a large number of dimensions, using disk I/O. Results are presented showing its performance and scalability.

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

1 Introduction

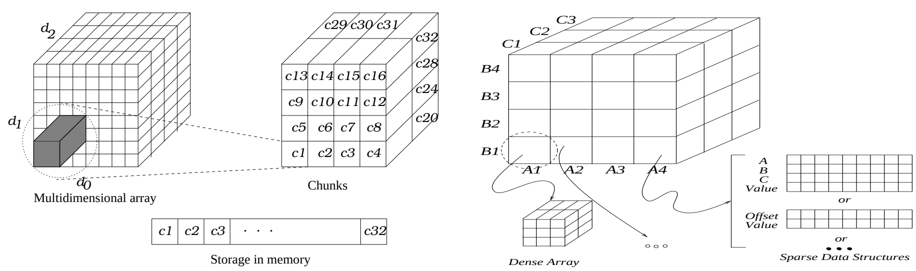
Decision support systems use On-Line Analytical processing (OLAP) and data mining to find interesting information from large databases. Multidimensional databases are suitable for OLAP and data mining since these applications require dimension oriented operations on data. Traditional multidimensional 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. For large databases, reduced storage requirements can reduce the I/O time needed for data accesses during operations when appropriate data structures are used. 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 dimension oriented operations can be more expensive when compared to multidimensional arrays, because of the additional processing required to de-reference indices. 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 those 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. Our techniques for multidimensional analysis can be applied to scientific and statistical databases [LS98] which rely heavily on dimensional analysis.

In this paper we present a parallel multidimensional framework for large data sets in OLAP and SSDB 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. Three schedules for constructing complete and partial data cubes from the cube lattice are developed, using cost estimates between parent and child aggregates: (1) Pipelined method for cubes (2) Pipelined method for chunks and (3) Level by level method. These incorporate various optimizations to reduce computation, communication and I/O overheads. Chunk management strategies are described to store the various intermediate cubes and handle large data sets.

The rest of the paper is organized as follows. Section 2 describes chunk storage using BESS and the multidimensional infrastructure. Section 3 describes mining of association rules on the data cube. Section 4 describes strategies and optimizations for parallel aggregate computation. Section 5 presents results of our implementation on the IBM SP2. Section 6 concludes the paper.

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 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.



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

Figure 1: Storage of data in chunks

Typically, sparse structures have been used for advantages they provide 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 our Bit-encoded sparse storage (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 of size $|d_i|$. A 8-byte encoding is used to store the BESS index along with the value at that location. A larger encoding can be used if the number of dimensions are larger than 20. A dimension index can then be extracted by a bit mask operation. Aggregation along a dimension d_i can be done by masking its dimension encoding in BESS and using a sort operation to get the duplicate resultant BESS values together. This is followed by a scan of the BESS index, aggregating values for each duplicate BESS index. For dimensional analysis, aggregation needs to be done for appropriate chunks along a dimensional plane.

3 OLAP and Data Mining

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. Alternatively, multi-dimensional database techniques (multi-dimensional OLAP) have been applied to decision-support applications. Data is stored in multi-dimensional structures 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 use bit-encoded sparse structure (BESS) is used for compressing chunks since it is suited for fast dimensional operations.

We have evaluated the nature of query operations in OLAP and SSDB in [GC97b], which are dimension oriented classified as:

- **Retrieval of a random cell element**
- **Retrieval along a dimension or a combination of dimensions**
- **Retrieval for values of dimensions within a range (Range Queries)**
- **Aggregation operations on dimensions**
- **Multi-dimensional Aggregation (Generalization/Consolidation: lower to higher level in hierarchy)**

In [GC97b], an analysis is provided for each of these query operations using various sparse data structures and it is shown that BESS is superior to others.

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.

3.1 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] 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 for some $j, k \in [1, n]$, x_j is a possible value for some attribute and x_k is a value for a different attribute of the underlying data. Whether or not E is interesting to that x_j's occurrence depends on x_k'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, interestingness function between A and B, A and C and finally between B and C are calculated. 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. Similarly, an appropriate placement of values is required for a two dimensional data distribution.

4 Implementation and Optimizations

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.

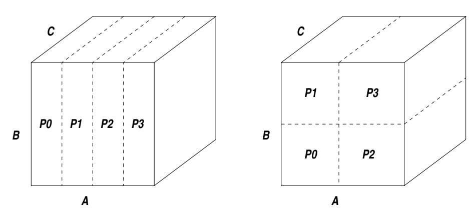


Figure 2: 1D and 2D partition for 3 dimensions on 4 processors

4.2 Partial cubes

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) in a complete cube, 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.

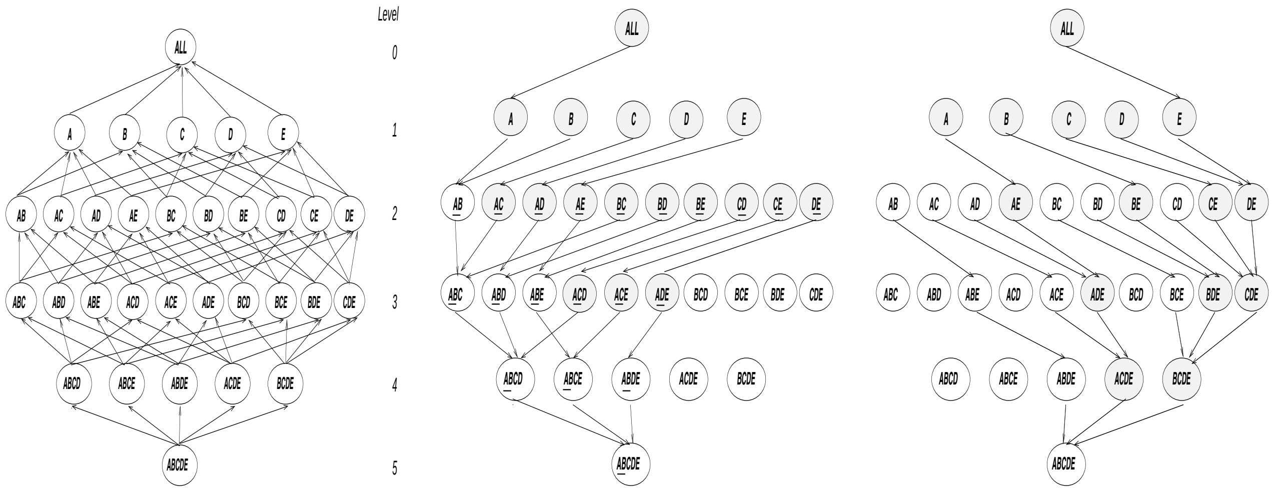
Also, for a large number of dimensions it is impractical to calculate the complete data cube for OLAP. A subset of the aggregates can be materialized in our framework, and queries can be answered from the existing materializations, or some others can be calculated on the fly. Data cube reductions for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

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 3(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 3(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 3(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, if 2D partitioning is considered. 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 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 $\begin{pmatrix} n \\ i \end{pmatrix}$ 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.



(a) (b) (c)

Figure 3: (a) Lattice for cube operator (b) DAG for partial data cube (left schedule) (c) right schedule

For $m = 2$, the number of aggregates at level 2 is $\sum_{j=1}^{n-1} \begin{pmatrix} j \\ 1 \end{pmatrix}$. For level 3 it is $\sum_{j=1}^{n-2} \begin{pmatrix} j \\ 1 \end{pmatrix}$. 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.

4.4 Cube Optimizations

Several optimizations can be done over the naive method of calculating each aggregate separately from the initial data [GBLP96].

1. **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.

2. **Cache Results**: 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 scans and I/O**. For example, after computing ABC from $ABCD$ we compute AB followed by A.

3. 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.

4. For partial cubes, as in mining m-way association rules, **minimizing the number of intermediate cubes**, at a level higher than m.

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 3(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. Applying the optimizations to this lattice will result in a cube ordering DAG.

4.5 Scheduling Aggregation Operations

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 using multidimensional arrays can be found in [GC97a]. For higher dimensions and sparse data sets, a chunk based implementations using BESS needs to determine the order in which chunks are accessed. To reduce disk I/O, chunk reuse has to be maximized. During aggregation, both source and destination chunks can be reused. The aggregation order can either be a *depth-first* traversal of the DAG or a *breadth-first* traversal. Each of these can be either done at the individual cube reuse level or at the chunk reuse level, thus resulting in 4 different schedules. At the cube reuse level, a child cube is calculated from the parent by aggregating each chunk of the parent along the aggregation dimension, in a pipelined manner. For a depth-first traversal this would result in the following order of calculation ABCD $\rightarrow$ ABC $\rightarrow$ AB $\rightarrow$ A, and then other calculations from ABCD and so on. Similarly, for a breadth-first traversal, ABCD $\rightarrow$ ABC, ABCD $\rightarrow$ ABD, ABCD $\rightarrow$ ACD, ABCD $\rightarrow$ BCD are materialized and then other calculations from ABC and so on.

In a chunk-reuse mechanism, the order of calculations are the same but this order is followed on a chunk by chunk basis. We are evaluating these for the effect of various optimizations on performance and results of these will be presented at the conference.

4.6 Chunk management

Chunks for each cube are arranged on disk in a given order of dimensions. Aggregation operations can either access them in linear order and optimize for I/O [ZDN97] or access chunks in a dimension order of the dimension being aggregated. In this work we explore both alternatives.

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 is 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. A file is kept to store the minichunks of the chunks in each cube.

Also, the chunk sizes have to be carefully chosen to balance the size of the overall chunk structure and the representation of the dimension indices for BESS. This also has an effect on the sizes of the files created for each cube, which on a conventional UNIX system are limited to 2 GB. For larger sizes either support for long pointers is required or a parallel file system can be used. Further, the chunk structure for each cube, which stores the meta-data information of chunks, their topology, distribution on processors, etc. has to fit in the available memory. If the chunk structure is larger than available memory, which might be the case for a large number of dimensions and large dimension sizes, the chunk structure has to be paged by keeping the chunk information of the current chunks in memory and the rest on disk. This will also help in the chunk-reuse based calculations by memory being allocated to only the parent and child chunks being calculated.

5 Results

We have implemented the cube construction and scheduling algorithms on a shared nothing parallel computer, the IBM-SP2. 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, using Message Passing Interface (MPI).

Our first prototype implementation was using multidimensional arrays for up to 10 dimensions, using 1D partitioning for complete datacube construction showing good parallel performance for the various stages involved. Figure 4 shows a sample result on a OLAP council benchmark. We are currently implementing a scalable multidimensional engine capable of handling large datasets, high number of dimensions, using chunks and sparse data structure (BESS) suitable for dimensional analysis. This supports both 1D and 2D partitioning, dense and sparse chunks, the various scheduling alternatives and optimizations.

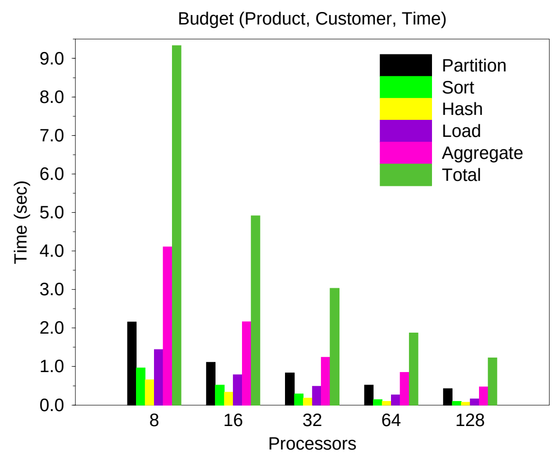


Figure 4: Various phases of cube construction using sort-based method for Budget data (from OLAP council benchmark) (N = 951,720 records, p = 8, 16, 32, 64 and 128 and data size = 420MB)

Assuming initial data from a relational database, the following summarizes the steps in the overall process. Assume N tuples and p processors.

1. *Each processor reads* $\frac{N}{p}$ *tuples.*
2. *Use a* **Partitioning** *algorithm to obtain a 1D (2D) partition of tuples using the largest one (two) dimension(s) as partitioner.*
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.*
8. *Perform dimension oriented OLAP and SSDB* **Query Operations** *on complete or partial data cubes.*

Four data sets, one each of 5D, 10D and two of 20D are considered. Table 1 describes the four 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.

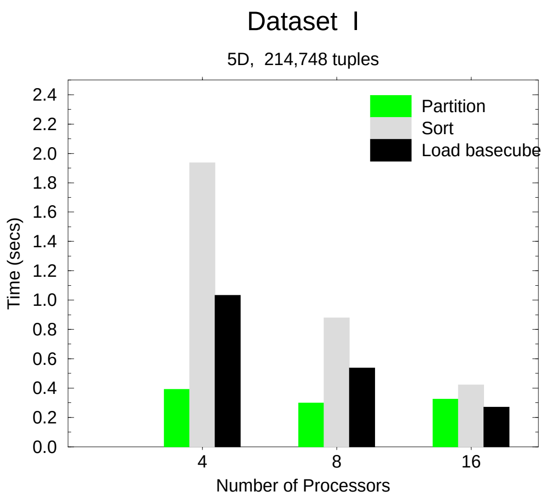
Table 1: Description of datasets and attributes, (N) Numeric (S) String

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

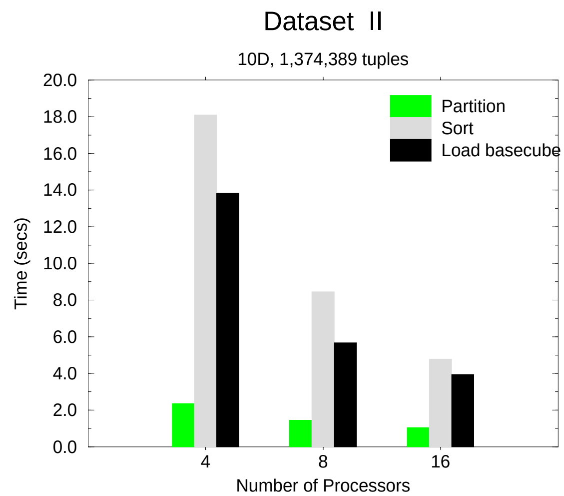
Results are presented for partitioning, sorting and chunk loading steps in Figure 5 and Figure 6. These show good performance of our techniques and show that they provide good performance on large data sets and are scalable to larger data sets (larger N) and larger number of processors (larger p). Even 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.

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.

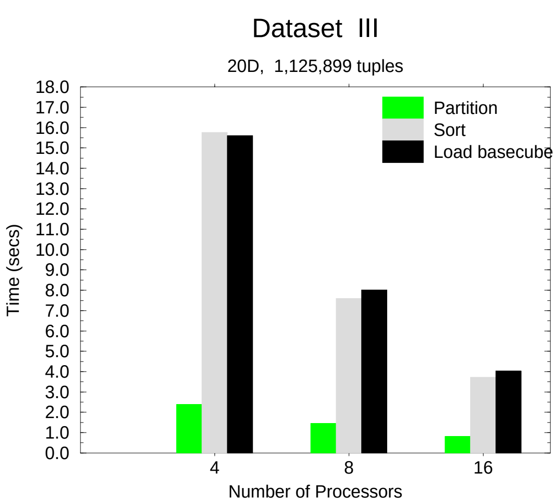


(a)

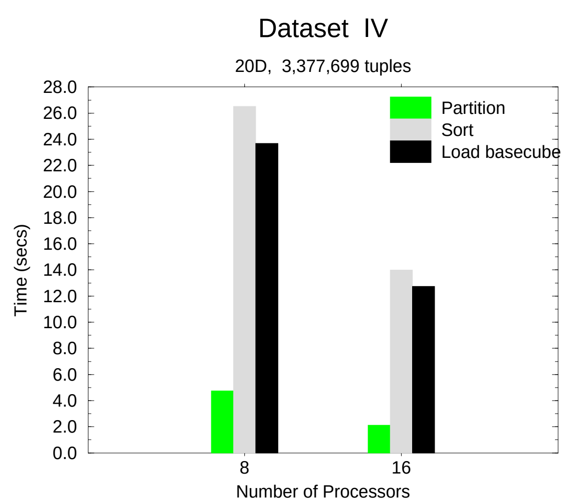


(b)

Figure 5: (a) Dataset I (5D, 214,748 tuples) (b) Dataset II (10D, 1,374,389 tuples)



(a)



(b)

Figure 6: (a) Dataset III (20D, 1,125,899 tuples) (b) Dataset IV (20D, 3,377,699 tuples)

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* 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.

[LS98] W. Lehner and W. Sporer. On the design and implementation of the multidimensional cubestore 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.