Intra-Query Parallelism for Multidimensional Array Data

Karl Hahn, Bernd Reiner

FORWISS – Bavarian Research Center for Knowledge Based Systems
Orleansstrasse 34, 81667 Munich, Germany
{hahnk, reiner}@forwiss.tu-muenchen.de

Abstract

Intra-query parallelism is a well-established mechanism for achieving high performance in (object-) relational database systems. However, the methods have yet not been applied to the upcoming field of multidimensional array databases. Specific properties of multidimensional array data require the adaptation of established methods but also new parallel algorithms. This paper presents a discussion of adapted and new techniques for parallelizing queries in multidimensional array database management systems. It shows their implementation in the RasDaMan DBMS, the first DBMS for generic multidimensional array data. The efficiency of the techniques presented is briefly discussed.

1. Introduction

Arrays of arbitrary size and dimensionality appear in a large variety of database application fields, e.g., medical imaging, geographic information systems, scientific simulations, etc. Recently, the integration of an application domain-independent, generic type constructor for such *Multidimensional Discrete Data (MDD)* into *Database Management Systems (DBMS)* has received growing attention. Current scientific contributions in this area mainly focus on MDD algebra and specialized storage architectures [1] [4].

Since MDD objects usually have a size of several MB and more and, operations on these values can be very complex, compared to scalar values, their efficient evaluation becomes a critical factor for the overall query response time. Beyond query optimization, parallel query processing is the most promising technique to speed up complex operations on large data volumes.

This paper discusses the suitability of intra-query parallelism concepts developed for relational DBMS in array DBMS. Special properties of array data, e.g., the size of a data object combined with expensive cell operations require adapted algorithms for parallel processing. Suitable concepts found in relational DBMS are already implemented and evaluated in the RasDaMan Array DBMS.

2. Parallel Query Processing in Relational DBMS vs. Array DBMS

2.1 Similarities and Differences

The data model of multidimensional array data can be compared to the relational data model: relational *tuples* consist of values for defined attributes; a relational *table* holds data of the same structure, i.e. tuples. The logical objects of multidimensional array data are *MDD* having a defined dimensionality, cell type, and spatial domain and *collections* which hold a set of MDD with the same structure. Hence, the multidimensional query language and query tree can be based on its relational counterpart. The Array DBMS RasDaMan supports a declarative multidimensional query language which is an extension of standard SQL. Like in SQL, conditions can be evaluated on data sets (where clause) and operations can be applied to the resulting data (select clause). The main difference is the complexity of the operations which in most cases leads to CPU-bound queries. As the most complex relational operations are sorting and join; typical operations on multidimensional data are arithmetic operations (like computing average or maximum and minimum values), 2-D image conversion, edge detection, Fourier transformation, etc. Hence, unlike its relational counterpart, the query tree is distributed in a relational-like part,

and several arithmetic-like parts. Therefore, parallel query execution has to consider both sections of the query tree. For the relational-like part (following the well-known iterator concept) established methods of parallel RDBMS can be adapted. The arithmetic-like section of the query tree requires new parallel algorithms which have to be developed and evaluated.

Evaluating parallel algorithms for array data we have to consider that the data being processed is quite different. A relational data object is typically very small and has a simple structure. MDD typically have a size of up to several GB, arbitrary dimensionality and arbitrary (often recursively structured) cell types. Therefore, the transmission of intermediate results has proven to be a special challenge in the implementation.

2.2 Relational Parallel Algorithms for Array Data

Parallel query execution in RDBMS was a major research issue of the last years [5] [7]. Parallel architectures (shared-nothing, -disc, -everything) [3], various algorithms (data- and pipeline-, inter- and intra-operator parallelism), and different execution strategies (load balancing, data distribution, etc) [2] [6] have been investigated. Some of these methods suit very well to array data while others do not.

Using the *Message Passing Interface (MPI)* standard for the inter-process communication MPI, we are independent of the parallel architecture.

Considering the special properties of array data and typical queries on this data we conclude that data parallelism is very well suited while pipeline parallelism shows inherent problems. As a data element typically is very large and the operations on it are typically complex the data elements can be distributed to the processes dynamically at query execution time. Unlike the relational data parallelism where a data distribution strategy is chosen in advance, the next data element can here be requested by each process on demand. Hence, data skew is avoided and load balancing is assured.

Pipeline parallelism on the other hand will not lead to an adequate speed-up. Transferring intermediate results between processes is much more expensive for array data because of the data's complexity.

2.3 New Methods for Very Large Data

The methods presented above were adapted from relational parallelism, taking an MDD as a logical, atomic data unit which was not split. Taking into account that large multidimensional data is often stored as a set of non-overlapping tiles (multidimensional object of the same dimensionality, covering a sub-area of the spatial domain) to assure good access times when retrieving multidimensional sub-areas we conclude that the operations on an MDD can also be done in parallel by distributing this internal storage unit of tiles.

Nevertheless, parallelizing an operation on one single MDD requires the analysis of each operation. Especially promising are two classes of operations, namely *induced operations* which apply an operation defined on the cell base type to a complete MDD (e.g., summation of 2 integers induces summation of 2 MDD with an integer cell base type) and *aggregation operations* which aggregate an MDD to a single value (e.g., average or summation of a complete MDD). Furthermore, the set of tiles required by processes are not always disjoint. An operation on two (or more) MDD with different tiling (tiles of different size and spatial domain) means that we have to give one tile to more than one process because this tile overlaps more than one tile of the other MDD.

Summarizing, this tile-based parallelism requires more complex algorithms but it can cover a typical class of multidimensional queries.

3. Results Achieved

3.1 Inter-MDD Parallelism

The parallel query execution described in section 2.2 was implemented and evaluated in the commercial Array DBMS RasDaMan.

Fig. 1 illustrates the query tree which was adapted to allow for parallel execution. The arithmetic-like parts of the query tree are only sketched in the figure as 'cond' for the condition tree and 'op' for multidimensional operations to be applied on the resulting MDD. Two pairs of *send and receive nodes* have been inserted to encapsulate the inter-process communication. Hence, the query tree is split in three parts which will be executed by different processes: one process holds the client-server connection and distributed the workload to the internal processes. The computational work (middle part of the query tree) will be done by several processes, typically as many as we have CPUs (or computers in a workstation cluster). As both, condition and operation of an MDD are done in one process, internal cached multidimensional objects can be used. One process called 'tuple server' manages the distribu-

tion of MDD to the processes. The MDD will be allocated on demand.

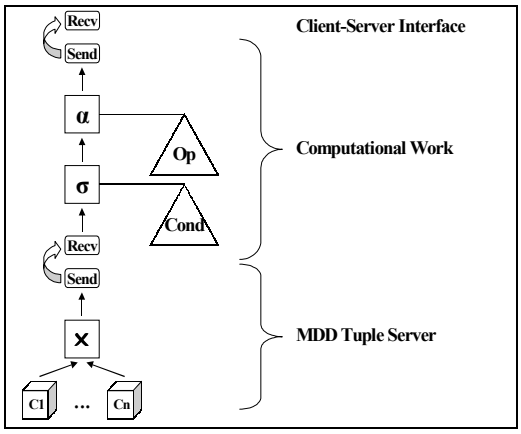

Fig. 1: Parallel query tree

This strategy ensures good speed-ups because it minimizes the inter-process communication and ensures excellent load balancing.

3.2 Evaluation

As we have expected this straightforward algorithm shows a very good performance (compared to the original RasDaMan implementation supporting no intra-query parallelism). So far we measured the speed-up on different multiprocessor computer (2 and 4 CPUs). CPU-bound queries on collections holding many MDD showed almost linear speed-ups. Tests will be extended to workstation cluster soon. At the moment we can not judge how the transmission of MDD over a network will affect query performance.

4. Future Work

4.1 Intra-MDD Parallelism

Research work of the coming months will concentrate on parallel execution of operations on a single MDD (as discussed in section 2.3). This requires the development of a tile distribution algorithm which ensures that minimal data is transferred between processes. Furthermore, each operation defined on multidimensional data (in the RasDaMan query language there are about 40 operations) has to be analyzed regarding the potential for parallel execution.

We do not expect intra-MDD parallelism to have a similar good performance as inter-MDD parallelism as data distribution and inter-process communication is much more complex. Nevertheless, this parallelism covers an important class of queries, i.e. a very complex operation on a single very large multidimensional object.

4.2 Parallel Dynamic Query Execution Control

The final goal for multidimensional intra-query parallelism is an execution control which chooses the best parallelization strategy at run-time. As far as possible the well-performing inter-object parallelism should be used unless one single data object has to be processed (this can also occur at the end of a query when only one single MDD is left).

Additionally, if more processes are available than MDD have to be processed, an intelligent combination of both parallelization strategies should be chosen by the parallel query execution control.

5. References

1. Baumann, P., Furtado, P., Ritsch, R., Widmann, N.: Geo/Environmental and Medical Data Management in the RasDaMan System. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), 1997
2. Bouganim, L., Florescu, D., Valduriez, P.: Dynamic Load Balancing in Hierarchical Parallel Database Systems. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), 1996
3. DeWitt, D.J., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems, Communication of the ACM, Volume 35, 1992
4. Furtado, P.A., Baumann, P.: Storage of Multidimensional Arrays Based on Arbitrary Tiling, Proc. of the ICDE, p. 480-489, 1999
5. Nippl, C., Mitschang, B.: TOPAZ: a Cost-Based, Rule-Driven, Multi-Phase Parallelizer. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), 1998
6. Rahm, E.: Dynamic Load Balancing in Parallel Database Systems. In Proc. of EURO-PAR, 1996
7. Tamer Özsu, M., Valduriez, P.: Principles of Distributed Database Systems, Second Edition. Prentice-Hall, 1999

6. Authors' Refereed Publications

1. Hahn, K., Sapia C., Blaschka M.: Automatically Generating OLAP Schemata from Conceptual Graphical Models. In Proc. of the 3rd Int. Workshop on Data Warehousing and OLAP (DOLAP), 2000
2. Hahn, K., Reiner, B, Höfling, G., Baumann, P.: Parallel Query Support for Multidimensional Data: Inter-object Parallelism. To appear in the Proc. of the 13th Int. Conf. on Database and Expert Systems Applications (DEXA), 2002
3. Reiner, B., Hahn, K., Höfling, G., Baumann, P.: Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. To appear in the Proc. of the 13th Int. Conf. on Database and Expert Systems Applications (DEXA), 2002