In: .IEEE Data Engineering, Vol. 1, No. 1, March 1989, pp. 23-29

Parallel Query Evaluation: A New Approach to Complex Object Processing

T. Härder H. Schöning A. Sikeler

University Kaiserslautern, Department of Computer Science, P.O. Box 3049, D-6750 Kaiserslautern, West Germany

Abstract

Complex objects to support non-standard database applications require the use of substantial computing resources because their powerful operations must be performed and maintained in an interactive environment. Since the exploitation of parallelism within such operations seems to be promising, we investigate the principal approaches for processing a query on complex objects (molecules) in parallel. A number of arguments favor methods based on inter-molecule parallelism as against intra-molecule parallelism. Retrieval of molecules may be optimized by multiple storage structures and access paths. Hence, maintenance of such storage redundancy seems to be another good application area to explore the use of parallelism.

1. Introduction

Non-standard database applications such as 3D-modeling for workpieces or VLSI chip design [1] require adequate modeling facilities for their application objects for various reasons. Enhanced data models provide many of such desired features; above all they support forms of data abstraction and encapsulation (e.g. ADTs) which relieve the application from the burden of maintaining intricate object representations and checking complex integrity constraints. On the other hand, the more powerful the data model the longer the DBMS's execution paths, since all aspects of complex object handling have to be performed inside the DBMS. Hence, appropriate means to concurrently execute "independent" parts of a user operation are highly desirable [2].

The use of intra-transaction parallelism for higher-level operations was investigated in a number of database machine projects [3]. These approaches focus on the exploitation of parallelism in the framework of the relational model. Complex relational queries are transformed into an operator tree of relational operations in which subtrees are executed concurrently (evaluation of subqueries on different relations) [4]. Other approaches utilize special storage allocation schemes by distributing relations across multiple disks. Parallelism is achieved by evaluating the same subquery on the various partitions of a relation [5, 6].

We investigate possible strategies to exploit parallelism when processing complex objects. In order to be specific, we have to identify our concepts and solutions in the framework of a particular data model and a system design facilitating the use of parallelism. Therefore, we refer to the molecule-atom data model (MAD model [7]) which is implemented by an NDBS kernel system called PRIMA [8]. We use the term NDBS to describe a database system tailored to the support of non-standard applications.

2. A Model of NDBS Operations

The overall architecture consists of a so-called NDBS kernel and a number of different application layers, which map particular applications to the data model interface of the kernel. Our application-independent kernel is divided into three layers:

- The storage system provides a powerful interface between main memory and disk. It maintains a database buffer and enables access to sets of pages organized in segments [8].
- The access system manages storage structures for basic objects called atoms and their related access paths. For performance reasons, multiple access paths and redundant storage structures may be defined for atoms.
- The data system dynamically builds the objects available at the data model interface. In our case, the kernel interface is characterized by the MAD model. Hence, the data system performs composition and decomposition of complex (structured) objects called molecules.

The application layer uses the complex objects and tailors them to (even more complex) objects according to the application model of a given application. This mapping is specific for each application area (e.g. 3D-CAD). Hence, different application layers exist which provide tailored interfaces (e.g. in form of a set of ADT operations) for the corresponding applications.

The NDBS architecture lends itself to a workstation-server environment in a smooth and natural way. The application and the corresponding application layer are dedicated to a workstation, whereas the NDBS kernel is assigned either to a single server processor or to a server complex consisting of multiple processors. This architectural subdivision is strongly facilitated by the properties of the MAD model: Sets of molecules consisting of sets of heterogeneous atoms may be specified as processing units.

Before we start to evaluate our concepts for achieving parallelism to perform data system and access system functions, we briefly sketch our process (run-time) environment. In order to provide suitable computing resources, PRIMA is mapped to a multi-processor system, i.e. the kernel code is allocated to each processor of our server complex (multiple DBMSs). The DB operations to be considered are typically executed on shared (or overlapping) data which requires synchronization of concurrent accesses. Due to the frequency of references (issued from concurrent tasks) accessibility of data and synchronization of access must be solved efficiently.

For this reason, we have designed a closely coupled multiprocessor system as a server complex. Each instance of PRIMA (running on a particular processor with private memory) uses an instruction-addressable common memory [9] for buffer management, synchronization, and logging/recovery. Furthermore, each instance of PRIMA is subdivided into a number of processes which may initiate an arbitrary number of tasks serving as run-units for the execution of single requests. Cooperation among processes is performed by establishing some kind of client-server relationship; the calling task in the client process issues a request to the server process where a task acts upon the request and returns an answer to the caller. In our model, a client invokes a server asynchronously, i.e. it can proceed after the invocation, and hence, can run concurrently with this server. To facilitate such complex and interleaved execution sequences we have designed a nested transaction concept [10] which serves as a flexible dynamic control structure and supports fine grained concurrency control as well as failure confinement within a nested subtransaction hierarchy. Due to space limitations we can not refine our arguments on these system issues [11].

2.1 The Data System Interface

In order to describe the concepts for achieving parallelism in sufficient detail, we have to refine our view of the kernel architecture and the interfaces involved. It is obvious that the data model plays the major role and determines many essential factors which enable reasonable parallelism: sufficiently large data granules, set orientation of request, dynamic construction of objects (result sets), flexible selection of processing sequences, etc.

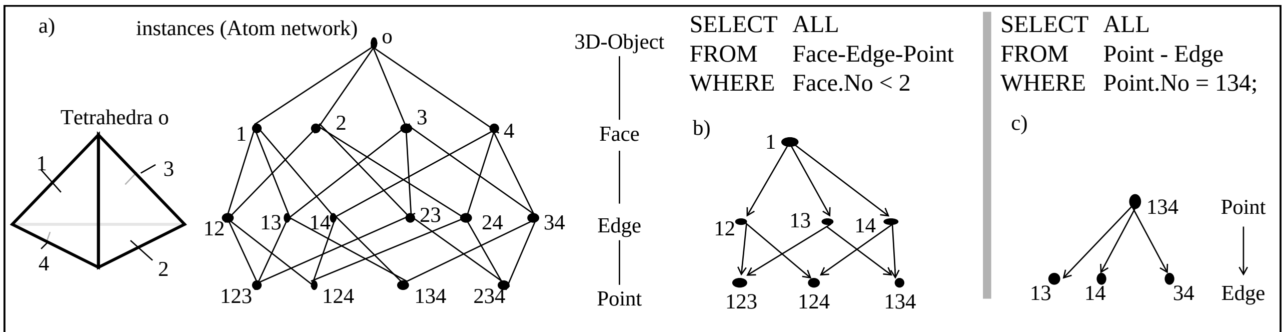
In our system, the data model interface is embodied by the MAD model and its language MQL which is similar to the well-known SQL language. Here, we cannot introduce this model with all its complex details, but only illustrate the most important concepts necessary for our discussion. In the MAD model, atoms are used as a kind of basic element (or building block) in order to represent entities of the real world. In a similar way to tuples in the relational model, they consist of an arbitrary number of attributes. The attributes' data types can, however, be chosen from a richer selection than in the relational model, i.e. apart from the conventional types the type concept includes

- the structured types RECORD and ARRAY,
- the repeating group types SET and LIST, both yielding a powerful structuring capability at the attribute level as well as
- the special types IDENTIFIER (serving as surrogates) for identification purposes and REF_TO for the connection of atoms.

Atoms are grouped to atom types. Relationships between atoms are expressed by so-called connections and are defined as connection types between atom types. Connection types are treated in a symmetric way, i.e. connections may be used in either direction in the same manner. Such connection types directly map all types of relationships (1:1, 1:n, n:m). The flexibility of the data model is greatly increased by this direct and symmetric mapping. Connection types are represented by a pair of REF_TO attributes (reference and "back-reference") one in either involved atom type, e.g.:

- FIDs: SET_OF (REF_TO(Face.EIDs)) in an atom type Edge
- EIDs: SET_OF (REF_TO(Edge.FIDs)) in an atom type Face.

In the database, all atoms connected by connections form meshed structures (atom networks) as illustrated in Fig. 1a.



```
SELECT  ALL
FROM    Face-Edge-Point
WHERE   Face.No < 2
```

```
SELECT  ALL
FROM    Point - Edge
WHERE   Point.No = 134;
```

Figure 1: Dynamic construction of molecules from a sample geometric object

Molecules are defined dynamically by using MQL statements and have to be derived at run-time. Each molecule belongs to a molecule type. The type description establishes a connected, directed and acyclic type graph (cycles occur when recursive types are specified), in that a starting point (i.e. root atom type) and all participating atom and connection types (for short "-") are specified. A particular example of a molecule type is Face-Edge-Point. Such a molecule type determines both the molecule structure as well as the molecule set which groups all molecules with the same structure. At the conceptual level, the dynamic construction of molecules proceeds in a straight-forward manner using the molecule type description as a template: For each atom belonging to the root atom type all children, grandchildren and so on are connected to the molecule structure, terminating after all leaves of the molecule structure are reached. Connecting children to the molecule structure means performing the hierarchical join which is supported by the connection concept. Hence, for each root atom a single molecule is constructed. Fig. 1b shows the result of a molecule construction for Face-Edge-Point molecules, where the set of molecules was restricted. Furthermore, it illustrates the most important properties of the MAD interface:

- An MQL request handles a set of molecules.
- The molecules as complex objects consist of sets of atoms of different type, i.e. they are embodied by sets of interrelated heterogeneous record structures.
- Molecule construction is dynamic and allows symmetric use of the atom networks (e.g. Point-Edge (Fig. 1c)).

2.2 The Access System Interface

The access system provides an atom-oriented interface which allows for direct and navigational retrieval as well as for the manipulation of atoms. To satisfy the retrieval requirements of the data system, it supports direct access to a single atom as well as atom-by-atom access to either homogeneous or heterogeneous atom sets. Manipulation operations (insert, modify, and delete) and direct access operate on single atoms identified by their logical address (or surrogate) which is used to implement the IDENTIFIER attribute as well as the REF_TO attributes. Performing manipulation operations, the access system is responsible for the automatic maintenance of the referential integrity defined by the REF_TO attributes. Thus, a manipulation operation on such an attribute requires implicit manipulation operations on other atoms in order to adjust the appropriate back references.

Different kinds of scan operations are introduced as a concept to manage a dynamically defined set of atoms, to hold a current position in such a set, and to successively deliver single atoms. Some scan operations are added in order to optimize retrieval access. Therefore, they may depend on the existence of a certain storage structure (defined by the database administrator):

- The atom-type scan delivers all atoms in a system-defined order utilizing the basic storage structure existing for each atom type.
- The access-path scan provides fast value-dependent access based on different access path structures such as B-trees, R-trees, and grid files.
- The sort scan processes all atoms following a specified sort criterion also utilizing the basic storage structure of an atom type. However, sorting an entire atom type is expensive and time consuming. Therefore, a sort scan may be supported by an additional storage structure, namely the sort order.
- The atom-cluster scan speeds up the construction of frequently used molecules by allocating all atoms of a corresponding molecule in physical contiguity using a tailored storage structure as a so-called atom cluster. For example, in Fig. 1 each Face atom and all its associated Edge and Point atoms may be organized to form an atom cluster [12]. On a logical level, an atom cluster corresponds to a molecule. It is described by a special so-called characteristic atom which consists of references to all atoms belonging to the molecule. This characteristic atom together with all the referenced atoms is mapped onto a single physical record which in turn is stored in a set of pages.

The underlying concept is to make storage redundancy available outside the access system by offering appropriate retrieval operations (i.e. the choice of several different scans for a particular access decision by the optimizer of the data system), whereas in the case of manipulation operations storage redundancy has to be hidden by the access system. As a consequence, maintaining storage redundancy in an efficient way is a major task of the access system. However, sequential update of all storage structures existing for a corresponding atom results in a lack of efficiency which is not acceptable. Therefore, exploiting parallelism seems to be a natural way to speed up a single manipulation operation.

3. Using Parallelism in Query Processing

Query processing operates on sets of molecules which are either extracted from the database (retrieval) or updated, inserted, or deleted (manipulation). In either case, the complex operation must be evaluated using operations on a single atom at a time. In this chapter we discuss some techniques to exploit parallelism in executing these atom-oriented operations as well as higher-order operations on intermediate results.

Three phases of query processing can be isolated [13]. The compilation phase checks for the syntactic and semantic correctness of a query and performs some obviously simplifying query transformations. Finally, it derives an operator tree consisting of several operator types, most of which accept and produce sets of molecules. The leaves of this operator tree transform heterogeneous atom sets to molecules (construction of simple molecules for retrieval) and vice versa (molecule modification by atom manipulation). This operator tree is the input of the optimization phase which is expected to reorganize it somehow into an “optimal” form. This includes reordering, combination, and splitting of operator tree nodes. Furthermore, methods have to be chosen for each operator, i.e. evaluation strategies (e.g. nested-loop join), storage structure usage (e.g. B-tree), and amount of parallelism (as described below). Although there is much to be said about these two phases, we concentrate on the discussion of the third phase, i.e. query evaluation. The input of this phase is the operator tree introduced above. For each node, we need an active unit to compute its result. Since there are several operator types, we assume a process for each of them. As a consequence, more than one node of an operator tree may be assigned to the same process.

We discuss the operator tree interpretation separately for retrieval and manipulation. In either case, we have to handle a set of molecules, each of which consists of a set of atoms. Thus, we investigate parallel handling of molecules (inter-molecule parallelism) as well as parallel handling of a molecule’s components (intra-molecule parallelism).

3.1 Parallelism in Retrieval Evaluation

Evaluation starts at the root operator, which needs results from all of its children in the operator tree to compute its own result. Depending on the operator type and the method chosen, children can be evaluated in parallel, e.g. the children of a binary merge-join operator can be accessed concurrently, while those of a nested-loop join cannot. The evaluation of leaf operators requires a lot of access system calls to build up simple molecules (hierarchical, non-recursive molecules). Of course, we assume that the access system is able to handle an arbitrary number of asynchronous calls in parallel. Therefore, construction of simple molecules seems to be a good candidate for using parallelism within the handling of each molecule.

Parallelism Within Construction of Simple Molecules

An obvious strategy to construct simple molecules is to call the root atom of each molecule via access system scan. Following this all child atoms of the root atom (which are identifiable by reference attributes) are called, then their children, and so on. These accesses to the children of an atom may be done in parallel (example 1a).

```
SELECT   ALL
FROM     Face-Edge-Point
WHERE    Face.No = 123;
```

Strategy:
Call Face;
Call all edges and
all points in parallel;

Query a

```
SELECT   ALL
FROM     Face-Edge-Point
WHERE    FOR_ALL Edge:
         Length > 10;
```

Strategy:
Call Face;
Call all edges sequentially;
If all edges fulfil the restriction:
call all points in parallel;

Query b

```
SELECT   ALL
FROM     Face-Edge-Point
WHERE    (FOR_ALL Point: x-coordinate = 5)
         OR (EXISTS Edge: Length > 10);
```

Strategy:
Call Face;
• call edges sequentially; then call points sequentially;
or • call one edge, call its points sequentially;
or • call n edges in parallel, call their points sequentially;

Query c

Example 1: Three sample queries and parallelism strategy choices

However, in many cases the user is not interested in all molecules of a certain type, but strongly restricts the molecules he wants to see. In this situation, it would be inefficient to fetch all atoms of all molecules and then throw away most of them by a separated restriction operator. Instead, we want to integrate the restriction facility into the operator "construction of simple molecules" leading to a more efficient evaluation strategy. Restrictions on the root atom are passed on to the access system which allows scan restrictions. All other restrictions on dependent atoms are evaluated as early as possible. As soon as it becomes evident during molecule construction that a molecule will be disqualified, none of its atoms has to be fetched any more, thus saving many access system calls (example 1b).

Of course, this approach is contradictory to the parallel molecule construction proposed above, because we want to fetch as few atoms as possible, if a molecule is disqualified. Therefore, we combine both techniques: Atom types that do not contribute to molecule qualification should be treated last. Their atoms can be called in parallel. Atom types restricted by an ALL-quantifier should be called sequentially, since each atom of this type can indicate molecule disqualification. While good strategies for these extreme cases are easy to find, much more complicated situations can be thought of (example 1c). They raise the question whether in some situation a compromise on the amount of parallelism and unnecessary atom accesses should be made, e.g. limitation of parallel atom calls to a constant n, thereby limiting unnecessary atom calls to n-1 (third choice in example 1c). We are still investigating this case for generally applicable rules to decide the optimal amount of parallelism for each atom type as well as the best sequence of atom accesses.

The top-down approaches suggested above are sometimes not the most efficient strategies. When highly selective restrictions are defined on some child types, a bottom-up approach may be more promising. In this case, the first step evaluates the qualifying child atoms. Since some of these atoms may be orphans, it is necessary to explicitly check the existence of a related root atom. Finally, the whole molecule is constructed for each of the identified root atoms following the same guidelines as sketched above (example 2b).

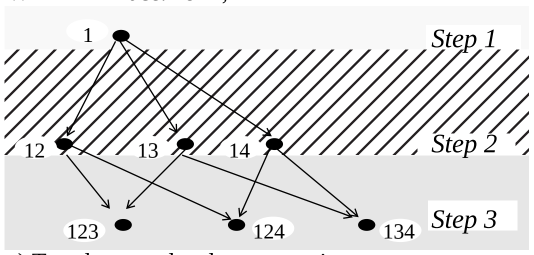
So far, we have discussed parallelism within the construction of one molecule. Since queries deal with sets of molecules, we should consider inter-molecule parallelism, too.

Inter-Molecule Parallelism

The most simple model for the computation of a set of molecules is to build up the first molecule completely, then the second and so on, thereby preserving the order of molecules induced by construction of simple molecules. Following this control scheme, there cannot be any parallelism among an process and its descendants or ancestors. To enable this kind of parallelism, we propose a pipeline mechanism. In particular, whenever the process for construction of simple molecules finds a root atom for a molecule, it builds up this molecule in a separate task, while another concurrent task calls the next root atom.

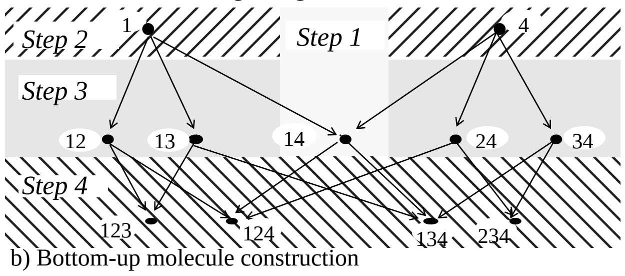
The pipeline structure defined this way (which at this point of the discussion is introduced as a model of computation and not as schedule for hardware-assignment), is very dynamic and complex, since the number of pipeline stages to run through is data dependent for many operator types and may vary for each molecule. Since this results in varying construction times, order-preservation cannot be guaranteed. As a consequence, there must not be any operator with a varying number of pipeline stages in the operator tree between a sort operator and the corresponding operator that relies on the sort order.

```
SELECT   ALL
FROM     Face-Edge-Point
WHERE    Face.No=1;
```



a) Top-down molecule construction

```
SELECT   ALL
FROM     Face-Edge-Point
WHERE    EXISTS Edge: Edge.No=14;
```



b) Bottom-up molecule construction

Example 2: Two ways to construct molecules

3.2 Parallelism in Manipulation Evaluation

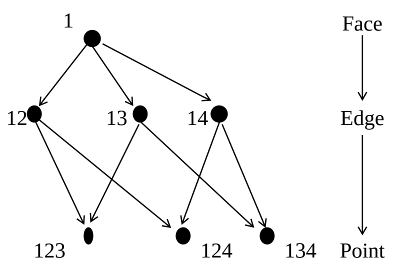
As for retrieval evaluation, we consider intra- and inter-molecule parallelism. Parallelism among several molecules by creation of a separate task for each of them is possible for manipulation, too. When existing molecules are to be manipulated, tasks emerging from retrieving them can be continued for manipulation. Within one molecule, either a top-down or a bottom-up strategy can be applied, both of them allowing parallelism among most of the atoms of a molecule (example 3).

sample manipulation statement

```
DELETE     ALL
FROM       Face-Edge-Point
WHERE      Face.No=1;
```

This is just a query to show evaluation strategies; it is semantically wrong with respect to figure 1.

affected molecule



top-down deletion

delete (1)
delete (13), delete (12), delete (14)
delete (123), delete (124), delete (134)

bottom_up deletion

delete (123), delete (124), delete (134)
delete (13), delete (12), delete (14)
delete (1)

all access system calls in the same line can be done in parallel

Example 3: Manipulation of a molecule with top-down and bottom-up strategy

4. Maintaining Redundancy by Parallel Operations

To speed up data system operations we have introduced some algorithms for the parallel construction/maintenance of complex objects represented as sets of heterogeneous atoms. In the following, we discuss the implementation of concurrent maintenance operations on redundant storage structures used for such atoms. As in the data system, two kinds of parallelism may be distinguished within the access system.

- The inter-operation parallelism allows for the parallel execution of an arbitrary number of independent access system calls. This kind of parallelism is a prerequisite for the parallelism introduced in the data system.
- The intra-operation parallelism however, exploits parallelism in executing a single access system call.

In this chapter, we will concentrate on intra-operation parallelism, since inter-operation parallelism is easily achieved by the underlying processing and transaction concept. For this purpose, however, the mapping process performed by the access system has to be outlined in some more detail in order to reveal purposeful suboperations to be executed in parallel.

In order to conceal the storage redundancy resulting from the different storage structures we have introduced the concept of a logical record (i.e. atom) made available at the access system interface and physical records stored in the “containers” offered by the storage system, i.e. each physical record represents an atom in either storage structure. As a consequence, an arbitrary number of physical records may be associated with each atom. For example, the creation of an atom cluster for each Face-Edge-Point molecule in Fig. 1 would imply that all Edge atoms belong to two atom clusters and all Point atoms to three (due to the properties of a tetrahedra). Furthermore, they always belong to the basic storage structure.

The relationship between a single atom and all its associated physical records is maintained by a sophisticated address structure related to each atom type. This address structure maps the logical address identifying an atom onto a list of physical addresses each indicating the location of a corresponding physical record within the “containers” (page address).

In contrast to the data system, however, the exploitation of parallelism within the access system is limited to the manipulation operations. Although most of the retrieval operations are also decomposed into further suboperations (e.g. in the case of an access-path scan on a tree structure: read the next entry in order to obtain the next logical address, access the address structure for the associated physical addresses, access either physical record), these suboperations cannot be executed in parallel due to the underlying precedence structure. Furthermore, each retrieval operation is tailored to a certain storage structure, thus operating not only on a single atom, but also on a single physical record.

On the other hand, each manipulation operation on an atom may be decomposed in quite a natural way into corresponding manipulation operations on the associated physical records. These lower-level manipulation operations, however, should be executed in parallel due to performance reasons. There exist (at least) two alternatives to perform such a parallel update:

Deferred Update

Deferred update means that during a manipulation operation on an atom initially only one of the associated physical records (e.g. in the basic storage structure of the atom type) is altered. All other physical records as well as the access paths are marked as invalid. Finally, a number of “processes” is initialized which alter the invalid structures in a deferred manner, whereas the manipulation operation itself is finished. Thus, low-level manipulation operations on additional storage structures may still run, although the manipulation operation on the corresponding atom or even each higher-level operation initializing the modification is already finished. This, however, strongly depends on the embedding of deferred update into the underlying transaction concept.

In order to mark a physical record as invalid the address structure may be used to indicate whether or not the corresponding physical record is valid. Therefore, all operations which utilize the address structure in order to locate a physical record may determine the valid records, whereas all operations which do not utilize the address structure will access invalid records unless the appropriate storage structure was already altered by the corresponding “process”. Hence, the corresponding storage structures themselves (access paths, sort orders, and atom clusters) have to be marked as invalid and when performing a scan operation on such an invalid structure each physical record has to be checked as to whether or not it is valid. This, however, requires an additional access to the address structure in order to locate a valid record. Consequently, the speed of a scan operation degrades, since each access to the address structure may result in an external disk access. In order to avoid this undesired behaviour, all invalid atoms (or their logical addresses) may be collected in a number of special pages assigned to each storage structure. These pages may be kept in the database buffer throughout the whole scan operation thus avoiding extra disk accesses. Nevertheless, each physical record has to be compared with the atoms collected in these pages. However, this is not sufficient, since each manipulation operation may require a modification of the whole storage structure, e.g. modifying an attribute which establishes a sort criterion requires the rearrangement of the corresponding physical record within the sort order. This fact also has to be considered during a scan operation. As a consequence, some of the scan operations may become rather complex and thus inefficient. For all these reasons, deferred update seems to be a bad idea.

Concurrent Update

The problem of maintaining invalid storage structures, however, is avoided by concurrent update. Concurrent update means that each manipulation operation on an atom invokes a number of “processes” which alter the associated physical records and access paths in parallel. The manipulation operation is finished when all “processes” are completed. When sufficient computing resources are available, concurrent update may not be more expensive, in terms of response time, than update of a single physical record if we neglect the cost of organizing parallelism.

Depending on the software structure of the access system, there are different ways to perform a concurrent update:

Autonomous Components

Each manipulation operation on an atom is directly passed to all components maintaining only a single storage structure type. Each component checks which storage structures of the appropriate type are affected by the manipulation operation. The corresponding storage structures are then modified either sequentially or again in parallel.

As a consequence, all components have to provide a uniform interface including all manipulation operations offered by the access system (i.e. insert, modify, and delete of a single atom identified by its logical address) as well as all retrieval operations. A quite simple distribution component directs each request to all components maintaining a storage structure type and collects their responses. This means, each component initially performs an evaluation phase during which it checks

- whether or not it has to perform the desired operation and if so,
- which storage structures of the appropriate type are really affected.

For this purpose, the addressing component (maintaining the common address structure) and the meta-data system (maintaining all required description information) are utilized. After the evaluation phase the proper operation is performed on each affected storage structure either sequentially or in parallel, thereby again utilizing two common components: the addressing component in order to notify the modification of a physical address and the mapping component in order to transform a logical record (i.e. atom) into a physical record and vice versa (thus achieving a uniform representation of physical records which is mandatory for some retrieval operations which use one of the physical records when accessing an atom (e.g. direct access)).

Thus, it is quite easy to add a new storage structure type (e.g. a dynamic hash structure as an additional access path structure) by simply integrating a corresponding component into the overall access system. However, there may be some drawbacks regarding performance aspects. During each operation, all components have to perform the evaluation phase although in many cases only a few or even only one component are affected. Moreover, the addressing component may become a bottleneck, since access to the address structure has to be synchronized in order to keep it consistent.

General Evaluation Component

These problems, however, may be avoided by a general evaluation component which replaces the simple distribution component as well as the evaluation phases in each of the components maintaining a storage structure type. Additionally, it solely maintains the address structure. As a consequence, this general evaluation component becomes much more complex. It requires dedicated information about each component in order to decide whether or not a component is affected by an operation, and it has to know the operations offered by either component in order to invoke the corresponding component in the right way. Although these operations may be tailored to the corresponding storage structure type (e.g. insert (key, logical address) in the case of an access path structure), it seems to be useful again to provide a uniform interface to all components in order to allow for a certain degree of extensibility. Such an interface has to consider the different characteristics of each storage structure type and the corresponding component (e.g. maintaining logical addresses instead of physical records) in an appropriate way.

In our initial design, we have decided to implement concurrent update based on a general evaluation component. In our opinion, concurrent update seems to be the better solution due to the invalidation problem. Although both software structures, autonomous components and general evaluation components, have their pros and cons with respect to performance and extensibility aspects [14], we prefer the general evaluation component, since it promises better performance. However, more detailed investigations are still necessary in order to determine the best way which may be a mixture of all. In particular, the influence of the underlying hardware architecture has to be investigated in more detail.

5. Conclusion

We have presented a discussion of the essential aspects of parallel query processing on complex objects. The focus of the paper has primarily been on the investigation of a multi-layered NDBS to achieve reasonable degrees of parallelism for a single user query. We have derived several design proposals embodying different concepts for the use of parallelism. In the data system, intra- and inter-molecule parallelism were explored. To exploit the former kind of parallelism seems to be more difficult because it turns out that it is very sensitive to the optimal degree of parallelism which may vary dynamically depending on the complex object characteristics. The latter concept is considered more promising because it allows simpler solutions. In the access system two approaches were investigated. Deferred update seems to provoke more problems than the solutions it might provide whereas concurrent update on redundant storage structures seems to incorporate a large potential for successful application.

Currently, we have finished the PRIMA implementation (single user version) and are integrating the proposed concepts for achieving parallelism in order to have a testbed for practical experiments. Performance analysis will reveal their strength and weaknesses at a more detailed level.

In the future, we wish to investigate further concepts for exploitation of parallelism. Another possibility of parallel execution on behalf of a single user would be the simultaneous activation of multiple requests within the application; for example, by means of a window system a user could issue several concurrent calls inherently related to the same task in a construction environment. Other possibilities to specify concurrent actions exist in the application layer where a complex ADT operation could be decomposed into compatible (non-conflicting) kernel requests. Usually multiple kernel requests are necessary for the data supply of an ADT operation; hence, these data requests can be expressed by MQL statements and issued concurrently to kernel servers when they do not conflict with each other or do not require a certain precedence structure.

References

[1] Dittrich, K.R., Dayal, U. (eds.): Proc. Int. Workshop on Object-Oriented Database Systems, Pacific Grove, 1986.

[2] Duppel, N., Peinl, P., Reuter, A., Schiele, G., Zeller, H.: Progress Report #2 of PROSPECT, Research Report, University Stuttgart, 1987.

[3] Special Issue on Database Machines, IEEE Transactions On Computers, Vol. C-28, No. 6, 1979.

[4] DeWitt, D., Gerber, R., Graefe, G., Heytens, M., Kumar, K., Muralikrishna, M.: GAMMA - A High Performance Dataflow Database Machine, in: Proc. VLDB 86, pp. 228-237.

[5] Neches, P.: The Anatomy of a Database Computer System, in: Proc. IEEE Spring Compcon, San Francisco, Feb. 1985.

[6] Lorie, R., Daudenarde, J., Hallmark, G., Stamos, J., Young, H.: Adding Intra-Transaction Parallelism to an Existing DBMS: Early Experience, IBM Research Report, RJ 6165, San Jose, CA, 1988.

[7] Mitschang, B.: Towards a Unified View of Design Data and Knowledge Representation, in: Proc. Second Int. Conf. on Expert Database Systems, Tysons Corner, Virginia, 1988, pp. 33-49.

[8] Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering Applications, in: Proc. VLDB 87, pp. 433-442.

[9] SEQUENT Solutions: Improving Database Performance, Sequent Computer Systems, Inc., 1987.

[10] Moss, J.E.B.: Nested Transactions: An Approach to Reliable Computing, M.I.T. Report MIT-LCS-TR-260, M.I.T., Laboratory of Computer Science, 1981.

[11] Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, in: Proc. Int. Symposium on Databases in Parallel and Distributed Systems, Austin, Texas, 1988, pp. 131-143.

[12] Schöning, H., Sikeler, A.: Cluster Mechanisms Supporting the Dynamic Construction of Complex Objects, to appear in: Proc. 3rd Int. Conf. on Foundations of Data Organization and Algorithms (FODO'89), June 21-23, 1989.

[13] Freytag, J.C.: A Rule-Based View of Query Optimization, in: ACM SIGMOD Annual Conference, 1987, pp. 173-186.

[14] Carey, M. (ed.): Special Issue on Extensible Database Systems, Database Engineering, Vol. 10, No. 2, 1987.