in : Proc. Euro-Par'96 - Parallel Processing, Workshop: Parallel and Distributed Datbase Systems, Lyon, Aug. 1996, LNCS 1124, Springer-Verlag, pp. 852-861.

Mapping a Parallel Complex-Object DBMS to Operating System Processes

Michael Gesmann
Department of Computer Science - University of Kaiserslautern
E-Mail: gesmann@informatik.uni-kl.de

Abstract. *So far, parallelism in complex-object and object-oriented DBMS has not been investigated in depth because descriptive and set-oriented query languages did not exist for these systems. However, with standardization of OQL by ODMG or SQL3 by ANSI, systems implementing these languages are ready to exploit parallel query processing strategies. In this paper, we explain differences between parallel query processing in relational and complex-object DBMS. Furthermore, we present a client/server-based system architecture that allows for fine-grained parallelism within query processing in complex-object DBMS. We investigate various strategies for mapping this architecture to processes of the underlying operating system. Finally, some measurements show the impact of these mapping strategies on query response times.*

1 Introduction

One of the most important challenges in relational DBMS query processing is the exploitation of parallelism which promises efficient handling of huge and ever growing amounts of data. Simultaneously, the demand of new applications for enhanced modeling capabilities and more expressive query languages led to object-oriented DBMS (OODBMS). However, until now, the exploitation of parallelism within OODBMS has been almost completely ignored. This is primarily due to the fact that set-oriented descriptive query languages were missing, which allow database systems to choose and optimize among various query processing strategies. Contrarily, application programs were responsible for the selection and construction of complex objects.

But, nowadays, standardization committees develop descriptive and set-oriented query languages equipped with high expressive power (OQL [1], SQL3 [2, 3]). Therefore, it seems to be a promising idea to adapt the well-known and very effective parallel processing strategies from relational DBMS to OODBMS in order to reduce query response times. However, query processing in complex-object DBMS and OODBMS shows the following entirely different characteristics:

- In OODBMS, direct representation of relationships between objects via references allows for (parallel) traversal of complex structures. The corresponding traversal algorithms are sometimes much more efficient than relational join operations embodying the only method to materialize relationships between tupels in relations [4, 5].
- Complex objects often incorporate highly meshed network structures; they may share components (atoms) among separate complex objects (molecules) [6] and may be even dynamically definable.
- In relational DBMS, data distribution across multiple sites (declustering) is one of the most important prerequisites to achieve data parallelism in query processing. Apparently, it is much more difficult in complex-object DBMS because every distribution strategy can only support some dedicated complex-object structures. Even worse, dynamically defined structures may not be preplanned at all.

• Finally, due to the enhanced semantics in OO data models and query languages, OO query processing requires comparatively more cpu-resources than relational query processing [6, 7]. These resources are mainly required for retrieval or manipulation of composite data types; obviously, they are also needed by more sophisticated query operators like recursion. Furthermore, due to structural overlap, when molecules are assembled from atoms, these atoms may appear in multiple roles in a single molecule or in different molecules. Therefore, query processing facilities have to separate atoms' data and molecules' structure information (composition relationships). This means, attribute values of a single atom, appearing in various roles in (a single or in multiple) resulting molecules should be represented only once without any replication. At the same time, since (data) atoms in different roles can comprise multiple references, this structure information has to be represented separately.

Due to these fundamental differences, OODBMS require new concepts when applying parallelism in query processing [8]. Concentrating on complex-object structures in this paper, we examine an implementation architecture for a parallel complex-object DBMS which supports fine-grained parallelism at various system layers. After having described the overall client/server-based architecture, we concentrate on the mapping of this architecture to the operating system services, i.e. processes.

[9] recently described how to parallelize OO7 benchmark's complex traversal operations. In contrast to our approach, their implementation encapsulates parallel strategies in methods belonging to objects. They did not try to integrate exploitation of parallelism in a more general query processing framework. Similarly, [10] explored OO traversals. In contrast to [9], their implementation is based on a vertical partitioning of object data and storing these data in relations. They concentrated on main memory processing and a declarative interface for path traversals which allows for system embedded traversal execution outside of application code in methods. [11] investigates parallel query processing algorithms. In contrast to our work, they also rely on vertical data partitioning and obtain their results from simulations. Issues about system architecture are not discussed in their work. Finally, [12] investigates parallel query execution when mapping object-oriented data to a relational DBMS and executing the same or different operations on various relations in parallel in a parallel relational DBMS.

In the next section, we describe our PRIMA system as far as needed for the following discussions and measurements. In Sect. 3, we describe how parallelism can be achieved and exploited in query processing within the system. Sect. 4 presents our measurements based on the OO7-Benchmark. The paper concludes with the most important results.

2 The PRIMA System

In this section, we give a concise overview of the PRIMA system for the subsequent discussion about issues of parallelism. For more detailed descriptions see [13, 14].

Data Model

First of all, we outline the relevant characteristics of the MAD model (cf. Fig. 2.1). Well-known concepts of the relational model help us to explain similarities and differences, when mapping entity and relationship types of the real word using the concepts of the data model:

Relations are named **atom types** and tupels are now called **atoms**, which represent entities of the real world. Atoms consist of **attributes** of various data types, are uniquely identifiable, and belong to their corresponding atom type. Atom identification is achieved by a special **identifier type** which is implemented by system-supplied surrogates. Values of this type are called **identifiers**.

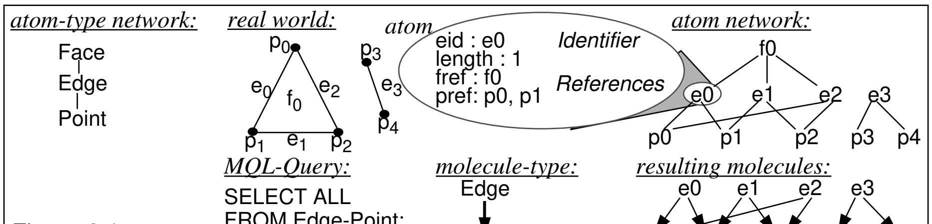

Figure 2.1: Illustration of MAD model concepts

All relevant relationship types between entity types are explicitly specified in the DB schema and represented in the DB. These relationship types, simply called **reference types**, are represented in an explicit and symmetrical way. As a result, the DB schema can be expressed by undirected graphs (network-like) of atom types. Atoms are connected to each other by **references** according to the reference types specified in the DB schema. As an important consequence, the DB can be viewed as an undirected network of atoms. References are implemented by lists of identifiers belonging to atoms of exactly one type, namely the referenced atom type.

Based on the atom network, complex objects, so called **molecules**, are dynamically definable. The SQL-like, set oriented, and descriptive query language MQL (molecule query language) is used to define their structure, named **molecule type**. An example of simple structural overlap of (simple) molecules is illustrated in Fig. 2.1 by the four resulting molecules characterized by the root atoms e0, e1, e2, and e3. Sharing of subcomponent types (subcomponents) and cyclic references can lead to meshed and recursive molecule types (molecules).

System Architecture

Like other systems, e.g. DASDBS [7], our system is implemented as a DBMS kernel architecture (cf. Fig. 2.2). This kernel system which implements an application-independent complex-object interface is responsible for all tasks of data and meta-data management, for mapping these structures to storage devices, as well as for transaction control. The data system implements operations on molecule sets offered at the interface of the kernel. These operations are internally transformed to operations on atoms. The access system transforms these operations on atoms to operations on blocks of the available storage structures. The storage system is responsible for buffer and file management. For parallel data access, it provides data distribution to multiple disks.

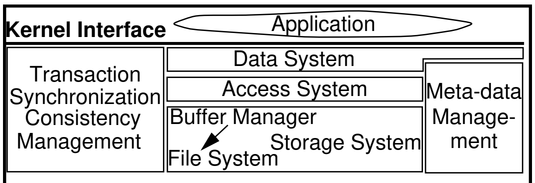

Figure 2.2: Architecture of the PRIMA system

Every layer is divided into a client/server architecture consisting of multiple services each implementing a dedicated functionality. In Fig. 2.2, this is indicated for the Storage System. Interfaces for every service expose set orientation in order to enable parallel processing.

3 Parallelism

Based on a functional decomposition, a client/server model defines the framework for database processing in PRIMA. Now, we introduce the notions as used in this paper and present basic concepts used to achieve parallelism in query processing.

A client requests some specific functionality, called **service**, from a **server process** which implements this service. To achieve real parallelism, multiple server processes must exist at a time executing concurrent requests simultaneously. A set of server processes implementing a service is called **server**. The execution of a service in a process is called **task**. **Server tasks** which need some other services in order to calculate their results have to invoke further **client tasks**.

When considering parallel query processing in DBMS a couple of parameters determine the system's performance, e.g. the underlying hardware, the runtime system offering parallel processing primitives, the algorithms used for specific operations, and, finally, mapping of services to operating system primitives. In the sequel, we will discuss these issues as far as necessary for the subject of this paper.

From the hardware's point of view, our runtime system especially supports shared-memory multiprocessor systems which enable very efficient communication and load distribution between clients and servers. Due to the already mentioned structural overlap of complex objects, we do not consider complex-object distribution in shared nothing systems.

From the runtime system's point of view task invocations are performed asynchronously in order to enable multiple independent server-task invocations and parallel execution of a client task with its server tasks. Furthermore, in order to allow pipelining between multiple stages of client-task/server-task relationships, a server task can return partial results to its client task. An efficient implementation of a runtime system [14] enables processing of fine-grained parallel tasks in multiple processes.

Considering the operating system primitives, parallelism appears only, if multiple processes execute tasks simultaneously. In the following, we illustrate the possibilities provided in PRIMA for mapping tasks and services to processes [15] (cf. Fig. 3.1).

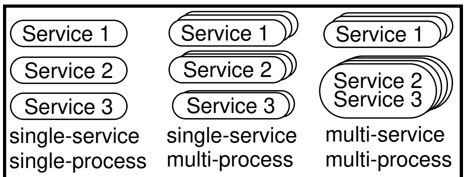

Figure 3.1: Service-Process-Mapping

We have chosen a multi-tasking approach, i.e., a single process can execute multiple tasks. In order to exploit waiting situations when executing a task, e.g. when waiting for results of server tasks, the client task voluntarily breaks its execution. Thereafter, the server process can continue with another task waiting for its execution. This implementation has two strong advantages over an alternative single-tasking implementation. First, it saves a lot of administration overhead (context switches, scheduling) in the operating system because OS administration is limited to a small and fixed number of processes. This enables higher degrees of parallelism by fine-grained parallel tasks. Second, this realization makes it possible to implement our own DBMS specific task-scheduling strategies which can exploit task semantics.

As described so far, every service can be mapped to a separate unique server process (single-service/single-process). However, to enable parallel execution of requests to the same service we allow a static replication of server processes (single-service/multi-process). In such an implementation, tasks computing a specific service may be started or continued in any available process of this server. Obviously, common data structures to be located in shared memory are required and access to these data structures has to be synchronized carefully. Finally, w.r.t. mapping of services to processes, we can combine multiple (closely cooperating) services within a single program (multi-service/multi-process) which avoids unnecessary context switches when invoking a task in another server. It particularly enables integration of all services into a single program, which then may run in multiple processes.

The presented process structures offer highest flexibility when mapping services to processes. Allocating services in processes and installing processes in a running system is

called configuring the system, the resulting constellation is called **configuration**.
In contrast to other systems, e.g. DBS3 [16], we do not exploit light-weight processes (tasks, threads) of modern operating systems for two reasons. First, current implementations do not allow for easy integration of our own DBMS specific scheduling strategies. Second, there were no efficient implementations available on a SEQUENT when starting the PRIMA implementation. In comparison to implementation architectures of other parallel relational systems, e.g. VOLCANO [17] or XPRS [18], we enable finer granules of parallelism. Whereas these systems parallelize query execution plans on the level of algebra operators we dynamically determine degrees of parallelism at initialization time of tasks on every level of our architecture. This means, in our system exploitation of parallelism is not visible on the algebra level and invisible to other operators and we do not introduce special operators. For a more detailed discussion see [8].

4 Measurements

In order to report on the benefits and potential gains of parallelism within an application environment, we set up the PRIMA system with special measurement tools and an appropriate workload to be executed on a SEQUENT Symmetry (S27). With a number of selected measurements, we want to explore the effect of various configurations on query response times. The OO7-Benchmark, shortly described in the next subsection served as a guideline to design a standard workload used in our measurements.

4.1 OO7-Benchmark

Because OO7 is completely described in multiple publications, we will just give an outline; for the benchmark's definition we refer to the literature, e.g. [19, 20]. OO7 models a synthetical complex-object hierarchy of design objects (cf. Fig. 4.1) which is assumed to be typical for CAD databases. The database stores modules which are composed of complex assemblies. Complex assemblies are aggregates composed of other complex assemblies or of basic assemblies. Basic assemblies consist of composite parts which contain atomic parts. Atomic parts are connected to each other in a predefined way. The operations to be executed in OO7 are adjusted to OODBMS, i.e. they primarily evaluate pointer-swizzling behavior and navigation through object structures. Furthermore, operations only cover very limited query facilities, i.e. simple path expressions.

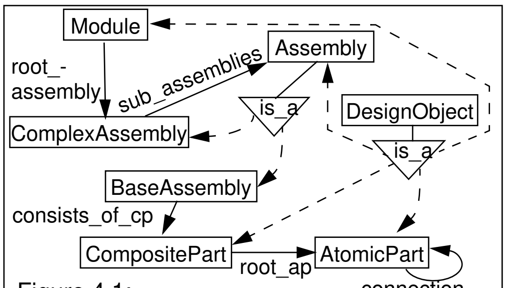

Figure 4.1:
Relevant Parts of the OO7-Schema

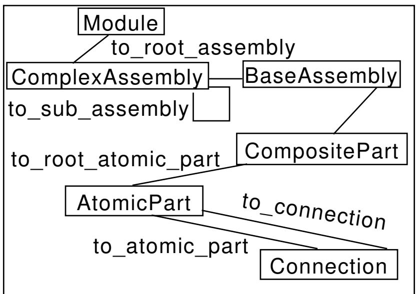

Figure 4.2: OO7-Schema of MAD

When implementing this benchmark on PRIMA, we had to adapt the benchmark's schema to the MAD model and the operations to MQL [21] (cf. Fig. 4.2). Top-level classes, which are not subclasses of other classes, attributes, and types can be mapped to atom types, attributes, and types of MAD, respectively. Simple references in OO7, implementing part-subpart relationships, can be expressed by reference types between corresponding atom types. Class - subclass relationships can be modeled explicitly in

MAD because OO7 contains only inheritance of attributes but not inherited values, methods, consistency constraints, etc. As a consequence, we can model all subclasses as individual atom types which comprise all inherited attributes.

After the schema transformation, we had to analyze the OO7 operations. Due to space limitations, we concentrate on the following benchmark operations:

Q5: A query which returns the numbers of those basic assemblies which consist of younger composite parts (in the benchmark this is called Single-Level Make),

T1: Traversal of the whole complex-object hierarchy (Raw Traversal), and

T6: Traversal of the complex-object hierarchy from modules to their atomic parts, but without traversing the connections between atomic parts (Sparse Traversal).

Our MQL queries for Q5, T1, and T6 are listed in the Appendix. Mapping Q5 to MQL is straight-forward and simple. Because the resulting molecules are very small the Data System has to accomplish only small amount of work. Translating traversal operations to queries is a bit more difficult because such statements include definition of complex recursive molecules. Due to processing of these recursions, query execution will consume substantial cpu-resources. Since the query for T1 is much more complicated than the query for T6, we measured query response times for T6 only.

In order to compare query response times to performance of pure traversals which do not construct complex objects, we additionally implemented T1 and T6 on top of our Access System. However, we provide two traversal algorithms, which are expected to show different performance behavior due to varying I/O-requirements and task administration overhead:

- depth-first (dft) corresponds to the specified depth-first traversal-algorithm in the benchmark, i.e., it follows references for every atom immediately in a depth-first order. This implementation does not allow for much parallelism and because of the huge number of invoked tasks, it induces a lot of administration overhead.
- set-oriented (st) corresponds to a breadth-first traversal-algorithm. However, in contrast to dft, it always requests as much atoms as possible from the access system. This means, a single task reads all atoms on a layer in the benchmark’s hierarchy, where module is layer 0, all referenced complex assemblies form layer 1, and so on. Due to these set-oriented requests, this approach does not provoke that much overhead than our dft implementation. Furthermore, it promises optimal performance if the DBMS buffer cannot store all requested pages because it causes less page I/O than dft, because set-oriented processing prevents unnecessary multiple reads of the same page [22].

4.2 Environment

For our measurements, we use a SEQUENT Symmetry (S27) which is a closely coupled multiprocessor system (8 processors, 6MHz Intel 80386), running DYNIX V3.0.18, and offering shared memory for communication and task management. Every processor has a cache of 64 kbytes. Processors and shared memory are connected by a 64-bit system bus with a channel bandwidth of 80 MB/second. Data has been allocated on a single disk (Fujitsu M2382K, about 25 ms avg. response time, 3 MB/second transfer rate).

All measurements described are executed in a conventional environment, i.e., neither the hardware environment nor the operating system have been modified. Of course, the machines are used exclusively during measurements. Queries and traversals are executed on a small database which covers about 5.7 MB (4MB data, 1.7 MB address mapping information) for those parts which are relevant in this context. The DBMS buffer comprises 0.8 MB. The employed PRIMA DBMS kernel consists of 14 services [8] which can be divided into the following **server groups** (compare to Fig. 2.2):

- data-system services (compiler, optimizer, query execution),
- access-system services (distribution, address information, basic storage, btree),
- storage-system services (buffer management, file management),
- transaction-management services (synchronization, logging&recovery, consistency-management, transaction management), and
- meta-data management service.

4.3 Configurations

By the measurements we want to compare the performance of the already described load w.r.t. different configurations differing in the degree of achievable inter- and intra-service parallelism. The configurations vary in the way of coupling services and groups of services within programs as well as in the number of processes running these programs. The considered configurations are:

- {1, 3}-services:We start measurements with configurations where every process incorporates a single service. This means, every process has to manage only tasks which are directed to the incorporated service. In order to allow for intra-service parallelism we need multiple processes executing the same service. Therefore, *3-services* refers to configurations in which every service runs in 3 processes.
- {1, 3}-groups: Major drawbacks of the previously described configurations are the vast number of processes and context switches which are inherently necessary when executing tasks in different services. In order to avoid these problems, we considered further configurations integrating all services which belong to the same server group into individual programs. The term *n-groups* denotes configurations, where each of these programs runs in n processes. Hence, 1-groups enables inter-service parallelism between services in different groups, but not inter-service parallelism between services in the same group and no intra-service parallelism at all. With n=3, we allow inter-service parallelism because tasks of services allocated in the same program can be executed simultaneously in different processes. Moreover, for the same reason, we can achieve intra-service parallelism of degree 3.
- {1, 5, 7, 10}-system: In order to further reduce the number of context switches, we finally integrated all services into a single program. This approach enables highest possible flexibility in load balancing since every process can serve every request. However, at the same time, it induces more task administration overhead than the previously described configurations, because every process has to manage tasks for all services. Running this program n times leads to the configuration called *n-system*. With n=1, there is no parallelism at all, with n>1, we enable inter- as well as intra-service parallelism. For n<8, we assume to have one processor available for every process all the time. Note, one processor is required for operating system processes and for our client process. For n>8, there are more processes than processors, and, therefore, processes may be interrupted due to operating system scheduling.

Of course, it is conceivable to define further configurations, however, the presented ones cover all relevant and interesting aspects to be investigated here, i.e., possible intra- as well as inter-service parallelism and operating system overhead.

4.4 Results

Table 1 summarizes the results of our measurements. It shows response times for depth-first raw traversal (dft(T1)), set-oriented raw traversal (st(T1)), set-oriented sparse traversal (st(T6)), sparse-traversal query (T6-Q) and single-level make query (Q5) within the different configurations, explained in Sect. 4.3. All numbers indicate response times in seconds.

	dft (T1)	st (T1)	st (T6)	T6-Q	Q5
1-services	3748	578	15.4	127	**43**
3-services	***3958 !***	501	13.7	155	59
1-groups	3403	590	15.8	134	46
3-groups	3944	494	13.6	**121**	46
1-system	**2944**	944	18.1	***192 !***	***72 !***
3-system	3494	466	13.1	142	51
7-system	3566	**425**	**10.6**	128	49
10-system	3581	***1845!***	***27.9!***	151	61

Table 1: Measurement results

In dft(T1), 1-system yields the best response times. This was expected because every request is performed sequentially by dft and, therefore, every atom is read by an individual request (task). Accordingly, this atom-oriented procedure naturally does not allow for any remarkable parallelism in the system. Due to the sequential and atom-oriented nature of the algorithm, further decomposition of the retrieval requests is useless. For example, when processing single steps of the retrieval within multiple services using separate processes (1-groups, 1-services) causes additional context switches and inter-process communication (of course via shared memory). This automatically leads to increased response times. Furthermore, since there are hardly any concurrent tasks to be executed in parallel, configurations implementing a single service in multiple processes (3-services, 3-groups, 3-,7-, 10-system) obviously cannot yield decreased response times. On the contrary, these configurations require more administration overhead for synchronization of central data structures and task scheduling and, therefore, cause worse results.

Independent from aspects of parallelism, st(T1) leads to considerably improved response times because of the following dependencies: First, since set-oriented requests prevent multiple reads of the same page, they provoke less I/O (dft: 82386 page requests causing 36193 page I/Os, st: 20411/19979). Second, the number of requests reduces drastically from over 40000 atom requests to about 40 requests for sets of atoms. As a consequence, the number of internal tasks decreases because the latter are set-oriented, too. This ends in drastically reduced task administration overhead. At the same time, set-oriented requests allow for more parallelism in internal processing. Even if our data actually is stored on a single disk, we divide atom requests asking for multiple atoms into 7 tasks (number of available processors) each processing distinct atom sets or, in the case of base scans, each reading distinct pages. This means, at the disk level every I/O is synchronized, but buffer management, address calculation and atom projection can be performed in parallel. In this improved environment, our 1-system configuration yields bad results in comparison to other configurations because it does not enable any parallelism. Contrarily, inter-service parallelism achieves substantial performance improvements. Due to sequential I/O intra-service parallelism attains only comparatively small improvements. In both set-oriented traversals our 7-system configuration achieves best response times. The extremely bad results for the 10-system configuration are caused by operating system (OS) process scheduling and by our implementation of busy waiting locks on central data structures. Short-term busy waiting locks are very efficient, if the OS does not interrupt processes. With 10-system, however, we always have a lot of tasks to be executed which keep all processes busy. Therefore, the OS has to interrupt running processes holding some locks. As a consequence, other processes requesting such a lock cannot be continued until the interrupted process will be resumed. At the same time, the waiting process does voluntarily give up the processor.

A comparison of query response times for T6-Q and Q5 between 1-system and 1-services again proves that inter-service parallelism and pipelining can yield considerable

performance improvements. Here, 1-system results in maximum response times whereas configurations which enable more exploitation of inter-service parallelism return better results. Results obtained with 3-services and 3-groups demonstrate that query execution apparently only marginally benefits from intra-service parallelism. Since Q5 and T6-Q require construction of a lot of (partial) molecules which could be done in parallel, this observation was surprising and will require some more detailed investigations in the future. Finally, compared to 1-services with 3-services response times increase due to the vast number of processes and context switches.
Summarizing, there is actually no optimal configuration for every load. As expected, our 1-system configuration optimally supports sequential processing. Set-oriented requests with high demands for I/O require data parallelism with various tasks retrieving pages from a couple of disks. Contrarily, within construction of molecules, pipelining can improve response times significantly. Therefore, system decomposition into multiple dedicated server processes yielded optimal performance for the current system implementation. Finally, the 7-system configuration leads to at least almost optimal results despite some additional administration overhead. Configurations where the number of processes exceeds the number of available processors (e.g. 10-system) can lead to drastic performance deteriorations. Therefore, we conclude that an the 7-system configuration is a very attractive candidate for a standard configuration supporting mixed workload consisting of various concurrent applications.

5 Summary

In this paper, we presented a client/server-based implementation architecture for a parallel complex-object DBMS which enables pipelining and independent parallelism between as well as within system services. In particular, we concentrated on various possibilities for mapping services to processes and evaluated these strategies by measurements. The measurements yielded some remarkable observations deserving further examinations. First of all, set-oriented traversal operations (st) additionally can benefit from I/O-parallelism when data is distributed across multiple disks. Therefore, we will integrate this feature in a next step. Second, pertaining queries, we observed performance improvements by pipelining between different services/layers. Third, the evaluation of a complex query containing a recursion operator showed the utmost importance of carefully implementing and considering parallel processing within single operators. In these cases, pipelining does not achieve optimal usage of processor resources and, therefore, does not result in optimal response times. Fourth, those configurations which comprise all services in every process, turned out to achieve nearly optimal response times in almost all operations considered despite their inherent administration overhead. Then, however, the number of processes must not exceed the number of available processors. Finally, compared with these results, it turned out that careful design of algorithms and system architecture is more important to achieve performance improvements than system configuration. Consequently, we will look for generally applicable strategies for intra-operator parallelism on complex objects especially considering the problem of overlapping structures.

References

[1] Bancilhon F., Ferran G.: The ODMG Standard for Object Databases, Proc. DASFAA '95, April 1995, pp. 273-283

[2] Pistor P.: Object-Orientation in SQL3: State and Tendency, (in German) Informatik Spektrum, Springer Verlag, Vol. 16, No. 2, 1993, pp. 89-94

[3] Chamberlain D., Mattos N., Cheng J., DeMichiel L.: Extending relational database technology for new applications, IBM Systems Journal, Vol. 33, No. 2, 1994, pp. 264-279

[4] Lieuwen D.F., DeWitt D.J., Mehta M.: Pointer-based Join Techniques for Object-Oriented Databases, Technical Report tr1099, University of Wisconsin, 1992
[5] Shekita E.J., Carey M.J.: A Performance Evaluation of Pointer-Based Joins, Proc. ACM SIGMOD Conf., June 1990, pp. 300-311
[6] Härder, T. et al.: PRIMA - A DBMS Prototype Supporting Engineering Applications, Proc. 13th VLDB Conf., 1987, pp. 433-442
[7] Schek, H.J., et al.: The DASDBS Project: Objectives, Experiences, and Future Prospects, IEEE Trans. on Knowledge and Data Engineering, Vol. 2, No. 1, 1990, pp. 25-43
[8] Gesmann, M.: Parallel Query Execution in Hierarchically Layered Dataflow-Driven Complex Object DBMS, Research Report, University of Kaiserslautern, 1996
[9] DeWitt, D.J., et al.: Parallelizing OODBMS traversals: a performance evaluation, The VLDB Journal, Vol. 5, No. 3, 1996, pp. 3-18
[10] Boncz, P.A., Kwakkel, F., Kersten, M.L.: High Performance support for OO traversals in Monet, CWI University of Amsterdam
[11] Thakore, A.K., Su, St.: Performance Analysis of Parallel Object-Oriented Query Processing Algorithms, Distributed and Parallel Databases, Vol. 1, No. 2, 1994, pp. 59-100
[12] Rys, M., Norrie, M.C., Schek, H.-J.: Intra-Transaction Parallelism in the Mapping of an Object Model to a Relational Multi-Processor System, appears in Proc. VLDB '96
[13] Mitschang B.: A Molecule-Atom-Datamodel for Enhanced Applications, (in German), Informatik-Fachberichte 195, Springer Verlag, 1988
[14] Gesmann, M.: Performance Evaluation of the Remote Cooperation System in PRIMA, Proc. 3rd Int. Conf. on Parallel and Distributed Systems, 1994, pp. 257-260
[15] Gesmann, M., Grasnickel, A., Schöning, H.: A Remote Cooperation System Supporting Interoperability in Heterogeneous Environments, Int. Workshop RIDE-IMS, 1993, pp. 152-160
[16] Bergsten, B., Couprie, M., Valduriez, P.: Prototyping DBS3, a Shared-Memory Parallel Database System, Int. Conf on Parallel and Distributed Information System, 1991, pp. 226-234
[17] Graefe, G.: Volcano, an Extensible and Parallel Query Evaluation System, IEEE Trans. on Knowledge and Data Engineering, Vol. 6, No. 1, 1994, pp. 120-135
[18] Hong, W., Stonebraker, M.: Optimization of Parallel Query Execution Plans in XPRS, Int. Conf on Parallel and Distributed Information System, 1991, pp. 218-225
[19] Carey M.J. et al.: A Status Report on the OO7 Benchmarking Effort, Proc. OOPSLA, 1994, pp. 414-426
[20] Carey M.J., DeWitt D.J., Naughton J.F: The OO7 Benchmark, Proc. ACM SIGMOD Conf., 1993, pp. 12-21
[21] Heck, A.: OO7 Benchmark on PRIMA, (in German), diploma thesis, University of Kaiserslautern, 1996
[22] Gesmann, M.: Fine-Grained Parallel Navigational Access in a Complex-Object DBMS, submitted to Int. Conf. on Parallel and Distributed Information Systems (PDIS) '96

Appendix

- Query Q5: Single-Level Make

```
SELECT BaseAssembly FROM BaseAssembly-CompositePart
WHERE CompositePart.buildDate > BaseAssembly.buildDate;
(* this statement returns BaseAssemblies and not only their number *)
```

- Traversal T1: Raw Traversal Speed (Query, not contained in our measurements)

```
SELECT Module (count_atomic_parts := COUNT( X.AtomicPart(ALL_REC).ID ))
FROM   Module.to_root_assembly-ComplexAssembly-BaseAssembly-CompositePart.
       to_root_atomic_part- X:= (SELECT ALL
                                 FROM AtomicPart.to_connection-Connection
                                 RECURSIVE Connection.to_atomic_part-AtomicPart )
RECURSIVE ComplexAssembly.to_sub_assembly-ComplexAssembly;
```

- Traversal T6: Sparse Traversal Speed (Query)

```
SELECT Module (count_atomic_parts := COUNT( AtomicPart(ALL_REC).ID ))
FROM   Module.to_root_assembly-ComplexAssembly-BaseAssembly-
       CompositePart.to_root_atomicPart-AtomicPart
RECURSIVE ComplexAssembly.to_sub_assembly-ComplexAssembly;
```