Parallelization and Object-Orientation: A Database Processing Point of View

David Taniar
Gippsland School of Computing
and Information Technology
Monash University
Australia
David.Taniar@fcit.monash.edu.au

J. Wenny Rahayu
School of Computer Science
and Computer Engineering
La Trobe University
Australia
wenny@latcs1.lat.oz.au

Abstract

To improve performance of object-oriented database processing, integration between parallel and object-oriented technologies is sought. In this paper, we concentrate on path expression queries. Parallelization of path expression queries can be achieved through simultaneous processing among objects (inter-object parallelization), or concurrent processing among classes (inter-class parallelization). These two parallelization models view parallel object-oriented query processing from two different angles, particularly from an object point of view and from a class point of view, respectively. We also present performance measurements through an implementation which highlight the strengths and the weaknesses of each parallelization model.

1 Introduction

The expressiveness of object-oriented data modelling has been one of the strengths of *Object-Oriented Database* (OODB), which also gives rise to highly complex data structures and access patterns, with a consequent adverse impact on database performance [6, 11]. Moreover, as database sizes grow to terabyte magnitude, there is a critical need to investigate methods for parallel execution of object-oriented database queries [18, 22].

Parallelism can be beneficial in the context of database processing for various reasons, such as to increase system throughput, and to decrease response time [8, 16]. The system throughput may be increased by applying inter-query parallelization, whereas query response time may improve by intra-query parallelization focusing at inter-operation and intra-operation parallelization. Parallelism allows a query to be split into sub-queries. Each of these sub-queries is allocated a number of processors on which to operate. Furthermore, multiple sub-queries may be processed simultaneously.

Apart from the performance benefits of parallelism, the integration between parallelism and object-orientation is also motivated by the following facts. The first fact is that objects are conceptually concurrent [5]. An object has its own thread of control. It can execute in parallel with other objects. This ability reveals potential applications of objects and object-orientation in parallel processing.

The second fact is that parallel machines have become increasingly popular [1, 15]. High performance parallel machines are no longer a monopoly of supercomputers. Parallel architectures now cover a broad range of architectures, i.e., from fast Local Area Networks connecting parallel servers and workstations (eg., quad-processor Pentiums, Sun workstations, DEC Alpha servers), to massively parallel processing systems *MPP* (eg., CM5).

It is the aim of this paper to describe how parallelization and object-orientation can be coupled to increase the performance of object-oriented database processing. Our previous work [20] concentrated on parallelization of inheritance queries. In this paper, we would like to explore parallelization techniques for path expression queries.

The rest of this paper is organized as follows. Section 2 explains parallelization elements which include parallel architectures and object partitioning. Section 3 describes path expression queries. Sections 4 and 5 present two parallelization techniques available for path expression queries, namely *inter-object parallelization* and *inter-class parallelization*. Section 6 presents some implementation results. And finally, section 7 draws the conclusions.

0-8186-8551-4/98 $10.00 © 1998 IEEE

2 Parallelization

There are two key factors in parallel object-oriented database processing: *distribution* and *processing* strategies [8]. Distribution deals with object partitioning in which particularly causing parallelism, whereas the processing strategy chooses the most efficient execution method that will be carried out by each processor. Since object distribution, being a key role of parallelism, is influenced by the parallel architecture, parallel architectures are discussed first, then followed by object partitioning methods.

2.1 Parallel Database Architectures

Parallel database architectures are normally classified into *four* categories: *shared-memory*, *shared-disk*, *shared-nothing*, and *shared-something* architectures [3, 23]. These architectures are shown in Figure 1.

Shared-memory architecture is an architecture where all processors share a common main memory and secondary memory. Processor load balancing is relatively easy to achieve, because data is located in one place. However, this architecture suffers from memory and bus contention, since many processors may compete for an access to the shared data.

In a *shared-disk* architecture, the disks are shared by all processors, each of which has its own local main memory. As a result, data sharing problems can be minimized, and load balancing can largely be maintained. On the other hand, this architecture suffers from congestion in the interconnection network when many processors are trying to access the disks at the same time.

A *shared-nothing* architecture, also known as a *distributed memory* architecture, provides each processor with a local main memory and disks. The problem of competing for access to the shared data will not occur in this system, but load balancing is difficult to achieve even for simple queries, since data is placed locally in each processor, and each processor may have unequal load. Because each processor is independent of others, it can be easy to scale up the number of processors without adversely affecting performance.

Finally, a *shared-something* architecture compromises the extensibility limitation of shared-memory and the load balancing problem of shared-nothing. There are a number of variations to this architecture, but basically each node is a shared-memory architecture connected to an interconnection network a la shared-nothing. Multiple disks (i.e., RAID) can also be attached to the network (or in each shared-memory node) to increase I/O bandwidth. Obvious features of a shared-something architecture include flexibility in the configuration (i.e., number of nodes, size of nodes) and lower network communication traffic as the number of nodes is reduced. Intra-query parallelization can be isolated to a single multiprocessor shared-memory node, as it is far easier to parallelize a query in a shared-memory than in a distributed system, and moreover, the degree of parallelism on a single shared-memory node may be sufficient for most applications. On the other hand, inter-query parallelization is consequently achieved through parallel execution among nodes.

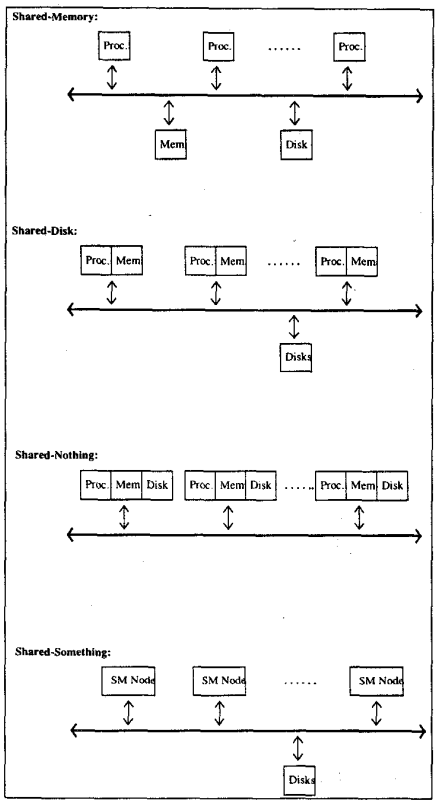

Figure 1. Parallel Database Architectures

2.2 Object Partitioning

Object partitioning is used to distribute objects over a number of processors. Each processor is then executed simultaneously with other processors. Depending on the architecture, object partitioning can be done physically or logically. In a shared-nothing architecture, objects are placed permanently over several disks, whereas in a shared-memory architecture, objects are assigned logically to each processor. Regardless of the adopted architecture, object partitioning plays an important role in parallel query processing since parallelism is achieved through object partitioning.

There is a number of well-known object partitioning strategies, namely *round-robin*, *hash*, and *range* partitioning [8]. Figure 2 gives an illustration of these partitioning methods.

The simplest technique is *round-robin* partitioning, where each complex object in turn is allocated to a processor in a clock-wise manner. Although the division of the root object may be equal, objects within one partition are not grouped semantically. Moreover, due to the fluctuation of the fan-out degree of the root class, some root objects might have a lot associated objects, while others have only a few, resulting in a skewness[1] problem occurring.

To make a partition more meaningful (by grouping objects having the same semantics or features), partitioning can be based on an attribute of the root class. One type of attribute-based partitioning is *hash partitioning*, where a hash function is applied. The result of this hash function determines the processor where the object will be placed. As a result, objects within one partition occupy the same hash value. This arrangement is best for exact match retrieval based on the partitioning attribute, where the processor containing the desired objects can be accessed directly. The problem of hash partitioning includes processing objects of a certain range, where hash partitioning cannot directly detect object location. A range-based partitioning is then needed.

Range partitioning spreads objects based on a given range of the partitioning attribute. Consequently, processing objects on a particular range of the partitioning attribute can be directed to a small subset of processors containing the desired range of objects. However, both hash and range partitioning risk *root object skew*[2], in addition to *association skew* as occurs in round-robin partitioning. Furthermore, retrieval processing based on a non-partitioning attribute cannot make use of the hash/range partitioning.

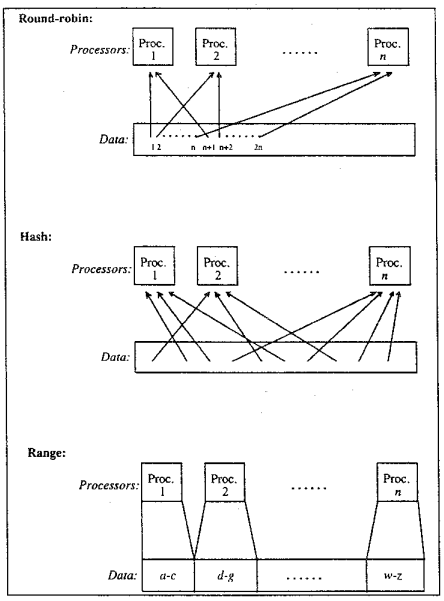

Figure 2. Object Partitioning

3 Path Expression Queries

Path expression queries are queries involving multiple classes along the association and aggregation hierarchies [2, 12]. This is one of the most common query forms in OODB. These queries are usually processed through path traversal.

Figure 3 shows an example of a class schema together with its instantiations. A typical query from this schema is to select objects which satisfy some predicates of both class *A* and *B* [4, 12].

OQL.

```
Select a
From a in A, b in a.rel1
Where a.attr1 = const AND
      b.attr1 = const;
```

[1] *skew* is when the variance of data distribution is greater than the mean.

[2] the skewness of the number of root objects in each partition.

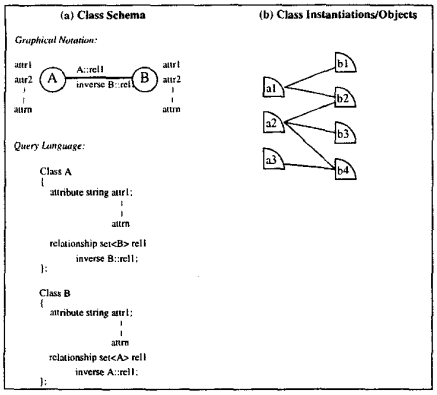

Figure 3. Class Schema and Instantiations

In this paper, class *A* is referred to as a *root class*, whereas class *B* is called an *associated class*. Further, objects of a root class are *root objects*, and objects of an associated class are *associated objects*.

4 Inter-Object Parallelization

Inter-object parallelization is a method whereby an object is processed simultaneously with other objects. Since path expression queries involve multiple classes along aggregation hierarchies, inter-object parallelization exploits the associativity within complex objects. All associated objects connected to a root object assemble a complex object. This associative approach views a complex object as a cluster, and consequently processing these objects can be done together.

Inter-object parallelization is accomplished by partitioning all complex objects rooted of a particular class into a number of partitions, in which each partition is allocated to a different processor. As a result, each processor works independently without a need for communicating with other processors. The partitioning method used is either *round-robin*, *range* or *hash* partitioning [8, 10]. Whatever partitioning method is used, it will not be that important to the associated objects, as the initial partitioning has lost its effect on them.

Using this associative approach, objects along the association path that are not reachable from the root object will not be processed. This method is very attractive because of not only the filtering feature, but also it is low in overhead. It does not require any checking, because processing root objects and their associated objects is done by pointer navigation from the root object to all of its associated objects. When there is no pointer left, it skips to the next root object. In this case, objects that do not form a complex object described in the query predicate are discarded naturally.

Figure 4 shows an inter-object parallelization from the example in Figure 3. Lower case letters are used to indicate OIDs. The number of associated objects for a root object is known as the *fan-out* degree of that root object. In this example, the fan-out degree of $a1$, $a2$ and $a3$ are equal to 2, 3 and 1, respectively. It clearly shows that using this clustering approach, processing a root object (say $a2$) can be done together with all its associated objects (e.g., $b2$, $b3$ and $b4$).

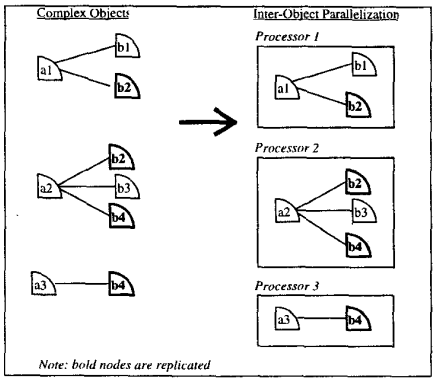

Figure 4. Inter-Object Parallelization Model

A problem of inter-object parallelization is that when the cardinality of the association is *many-many* or *many-one*, object replication is unavoidable. Associated objects referred by more than one root object will need to be replicated. In the example, objects $b2$ and $b4$ are replicated as they accompany root objects $a1$, $a2$ and $a3$.

5 Inter-Class Parallelization

Inter-class parallelization is a method whereby a query involving multiple classes and each class appearing in the query predicate is evaluated simultaneously. Inter-class parallelization considers each predicate as an independent task, and the objects of a particular class are attached to the predicate to be evaluated. As a result, the

entire process is composed of many independent tasks, which may run concurrently.

Basically, inter-class parallelization consists of two phases: *selection* phase and *consolidation* phase. The selection phase is a process where the predicate of each class is invoked independently regardless of the associative relationship. In the consolidation phase, the results from the selection phase are consolidated to obtain the final results.

Inter-class parallelization does not filter unnecessary objects prior to processing. *Non-associated* objects will be processed, although these objects will not be part of the query results. The processing performance of a class will be down graded by $(1-\alpha)$ times 100% percent, where α is a probability of an object of having an association with objects from a different class. This problem will not exist if both classes have *total participation* in the association relationship ($\alpha=1$).

Inter-class parallelization also determines access plans of path expression queries. Figure 5 shows two examples of access plans. When there is only one selection involved in the query, only the class involved in the selection operation is processed in the selection phase.

5.1 Selection Phase

There are two options for implementing a selection phase, especially when the two classes in a path expression query are involved in a selection. The options are *sharing resources* and *queuing for resources*.

- Sharing resources is a manifest of concurrent processing. The two classes share resources (ie. processors) at the same time. The resources must be divided into two groups: each group to serve one class. The division is not necessarily equal depending on the size of each class. Determining an appropriate number of processors for each class is critical. Otherwise, it will create load imbalance as one class might have finished processing while others have not. Figure 6 shows a selection phase where the resources are divided into 2 groups.

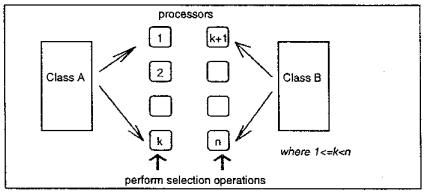

Figure 6. Selection Phase (Resource Division)

- Queuing for resources is typical in a pipeline processing model. Once a class takes the control, all resources will be allocated to it. There is no need to divide the resources. The usage of processors will be optimal, because when a class has finished, another class will occupy the idle processors. In this way, load balancing can always be maintained. Figure 7 shows class *A* and class *B* are queuing to use the resources.

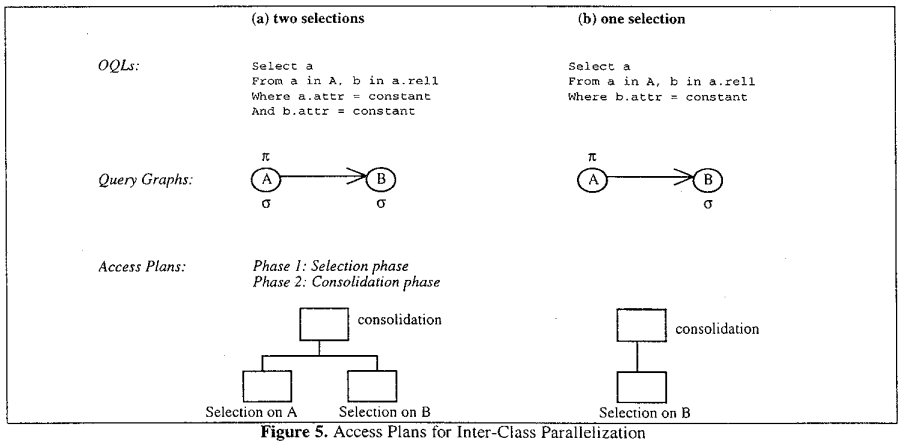

Figure 5. Access Plans for Inter-Class Parallelization

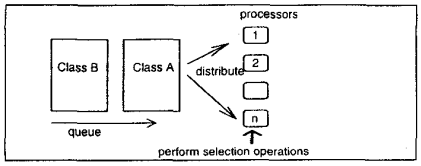

Figure 7. Selection Phase (Queuing up for resources)

The difference between "sharing resources" and "queuing for resources" can be illustrated by two queues for "sharing resources" and one queue for "queuing for resources". Because only the average workload of each processor is considered (not the response time of each item in the queue), one queue model is more efficient, because it guarantees that all processors (service providers) will be busy when the queue is not empty.

5.2 Consolidation Phase

Inter-class parallelization is an "independence class processing" based parallelization model. The development of this class-independence processing is influenced by the concept of object copying used in object-oriented query processing [14].

Basically, the results of a query are a *copy* of objects satisfying the selection predicates. In the absence of the selection predicates, the query results are the same as the original objects. Figure 8(a) shows an example of a simple query to retrieve all student objects. The result of this query is pointed by variable *a* which is the same copy of all student objects.

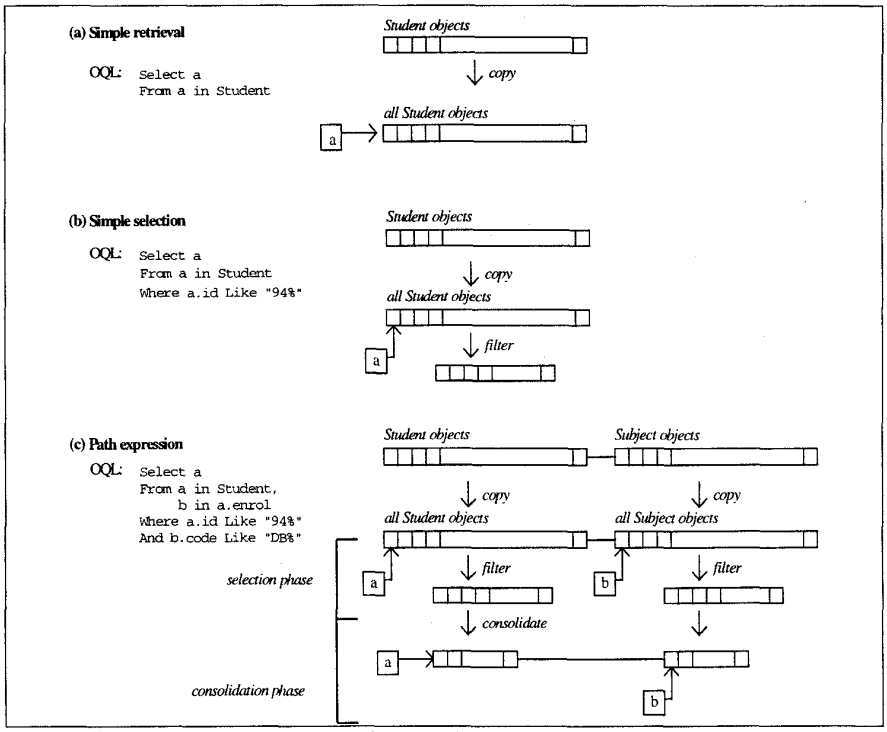

Figure 8. Object Copying in Query Retrieval Operations

In the presence of selection predicates, filtering is carried out to the copied objects. Figure 8(b) shows that variable *a* points to student objects which satisfy the selection predicate (i.e., ID Like "94%").

Using the same principle, when the selection predicates span to classes in a path expression, object copying and filtering can be performed for each class independently. Running through each root object once again to check whether the root object not pointing to a NULL value is done thereafter. Figure 8(c) shows a path expression query, the process to obtain the results, and the query results.

A consolidation process is performed by means of a "NOT NULL" association evaluation of the root object.

6 Implementation

The experimental environment was a DEC Alpha 2100 model with 4 CPUs running at 190MHz. The total performance of the system is around 3000Mips and 8Gflops. The size of main memory was 2Gb, and each CPU was equipped with 4Mb cache. The processors are all based on the same 64-bit RISC technology. The 64-bit technology breaks the 2-gigabytes limitations imposed by conventional 32-bit systems. Subsequently, the usage of very large memory is common to Digital Alpha servers. Very large memory systems significantly enhance the performance of very large database applications by caching key data into memory. The underlying operating system was Digital UNIX, and the algorithms were implemented in C++.

The Alpha system structure is shown in Figure 9. Four CPUs, each is equipped with a sufficient cache, are connected to a shared memory through a high-speed bus system.

In the experimentations, a two-class path expression query was constructed. The objects were generated by a random number generator, in which the degree of fan-out and selectivity were also created.

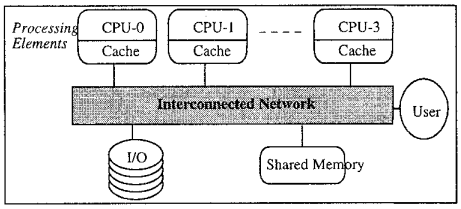

Figure 9. The Alpha System Structure

6.1 Inter-Object Parallelization

Inter-object parallelization is well-recognized mainly due to its filtering feature. The effect of selectivity degree on filtering will be investigated. The cost of inter-object parallelization includes the processing costs for the root class and the associated class. The proportion of the root class processing cost and the associated class processing cost, especially in the presence of *association skew*, and the effect of skewness to speed up, will be examined.

Figure 10 shows the performance of inter-object parallelization by varying the selectivity factor. When the selectivity degree is low, the elapsed time taken to answer the query is also low, regardless of the fan-out degree. This is because most of the associated objects are not accessed and subsequently the fan-out degree gives only little impact. As the selectivity degree grows, the processing cost also increases, especially for those medium to high fan-out degrees.

The impact of low fan-out, when the selectivity degree is high, is not as big as those with higher fan-out degree. This demonstrates that when the selectivity is high, the processing cost is determined by the number of accesses to the associated class which is partly indicated by the fan-out degree.

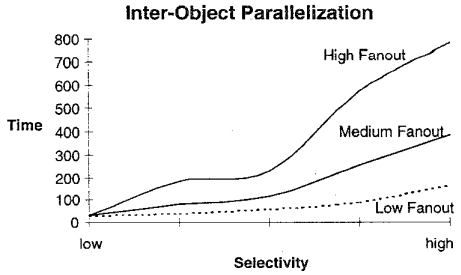

Figure 10. Performance of Inter-Object Parallelization

Figure 11 shows a comparison between the processing cost for the root class and the associated class, particularly in regard to the association skewness. When the association skew is low, which refers to the associated objects being distributed quite evenly (note that using a round-robin partitioning, the root class is divided equally to all processors), the processing cost for the associated class is also low. However, when the association skew is getting worse, the processing cost for the associated class is becoming higher too, especially when the degree of skewness is really high. In contrast, the processing cost for the root class is quite steady, despite the association

skewness degree. This is because the root class has been divided quite equally. Depending on the fan-out degree which determines the number of accesses to the associated class, and the degree of association skew, the processing cost for the associated class can become dominant, especially when the aforementioned two factors are indicated to be quite high.

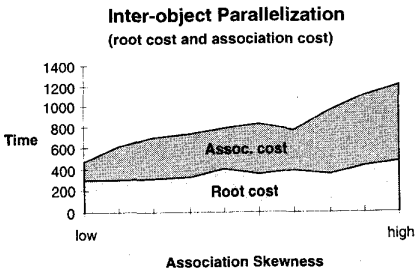

Figure 11. Processing costs for the root class and the associated class.

Figure 12 shows the effect of skew on the performance of the inter-object parallelization. The result shows that the skewness affected the improvement greatly. Only when the skewness is low, is near-linear speed-up attainable. This indicates that without a careful treatment of the skew problem, performance improvement is barely achieved.

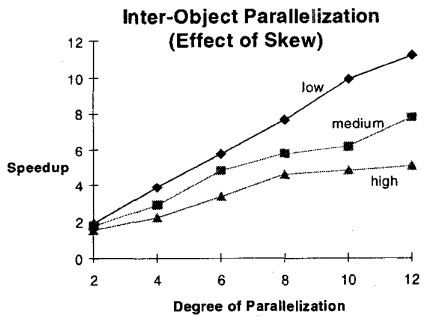

Figure 12. Performance of inter-object parallelization in the presence of skew.

6.2 Inter-Class Parallelization

As inter-class parallelization is divided into two phases, selection phase and consolidation phase, these two elements will be investigated in the overall cost for inter-class parallelization.

Figure 13(a) shows the comparison between the processing costs for the selection and the consolidation, particularly when the query involves two selection operations: one selection operation on each class. The selection cost is shown to be dominant, and quite constant regardless of the selectivity factor. It is because all objects from the two classes in the query need to be accessed. The consolidation cost is shown to be minor and increases when the selectivity factor is high. This indicates that, using a shared-memory/distributed cache main-memory architecture, the consolidation cost for this particular query type is low.

Figure 13(b) presents a performance of inter-class parallelization for queries having selection operations on the root class and no selection operations on the associated class. The selection path is shown to be quite constant and smaller than the one having two selection operations. In this query type, the selection operation is the cost for going through all root objects only. The consolidation cost is shown to be non-trivial. With the increase of the selectivity degree, the consolidation cost also increases. This cost includes the cost for accessing the associated objects for each selected root object.

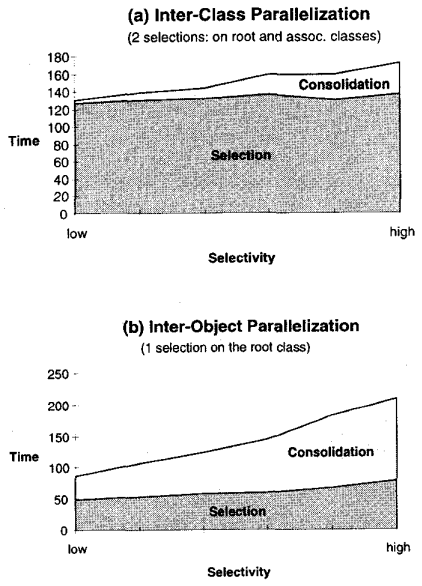

Figure 13. Performance of Inter-Class Parallelization

Figure 14 shows a comparison between queries having two selections (one on each class), one selection on the root class, and one selection on the associated class. When the selectivity degree is low, performance of the query having a selection on the root class is demonstrated to be the best. This is because the selection cost is lower than that of the queries with two selections, and the consolidation cost seems to be lower than that of the queries with one selection on the associated class. As the selectivity degree increases, the filtering feature provided by the selection operation on the root class becomes ineffective. Hence, performance of the query having a selection on the associated class becomes the best. This is because the selection part of this query is lower than that of queries having two selections, and the consolidation part of this query seems to be not as high as that of queries with a selection on the root class.

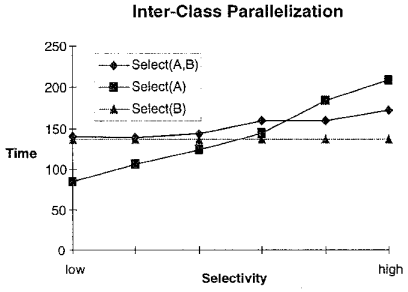

Figure 14. Performance of inter-class parallelization of a variety of query types.

7 Conclusions

In this paper we have presented an application of parallel processing on object-oriented path expression queries. Parallelization models for path expression queries are available in two forms: *inter-object* parallelization which exploits the associativity of complex objects, and *inter-class* parallelization which produces process independency. Inter-object parallelization will function well if a filtering mechanism in the form of selection operation exists. On the other hand, inter-class parallelization relies upon independency among classes, not the filtering feature. These two parallelization models form the basis for parallelization of more complex object-oriented queries.

References

[1] Almasi G., and Gottlieb, A., *Highly Parallel Computing*, Second edition, The Benjamin/Cummings Publishing Company Inc., 1994.

[2] Banerjee, J., Kim, W. and Kim, K-C., "Queries in Object-Oriented Databases", *Proceedings of the 4th International Conference on Data Engineering*, pp. 31-38, February 1988.

[3] Bergsten, B., Couprie, M., and Valduriez, P., "Overview of Parallel Architecture for Databases", *The Computer Journal*, vol. 36, no. 8, pp. 734-740, 1993.

[4] Bertino, E, et al., "Object-Oriented Query Languages: The Notion and The Issues", *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, pp. 223-237, June 1992.

[5] Booch, Grady, *Object-Oriented Analysis and Design with Applications*, second edition, The Benjamin/Cummings Publishing Company, Inc., 1994.

[6] Carey, M.J. and DeWitt, D.J., "Of Objects and Databases: A Decade of Turnmoil", *Proceedings of the 22nd VLDB Conference*, Bombay, India, 1996.

[7] Cattell, R.G.G. (ed.), *The Object Database Standard: ODMG-93*, Release 1.1, Morgan Kaufmann, 1994.

[8] DeWitt, D.J. and Gray, J., "Parallel Database Systems: The Future of High Performance Database Systems", *Communication of the ACM*, vol. 35, no. 6, pp. 85-98, 1992.

[9] Elmasri, R. and Navathe, S.B., *Fundamental of Database Systems*, Second Edition, The Benjamin/Cummings Publishing Company, 1994.

[10] Graefe, G., "Query Evaluation Techniques for Large Databases", *ACM Computing Surveys*, vol. 25, no. 2, pp. 73-170, June 1993.

[11] Hurson, A.R. and Pakzad, S.H., "Object-Oriented Database Management Systems: Evolution and Performance Issues", *IEEE Computer*, Feb 1993.

[12] Kim, W., "A Model of Queries for Object-Oriented Databases", *Proceedings of the 15th International Conference on Very Large Data Bases VLDB*, Amsterdam, pp. 423-432, 1989.

[13] Leung, C.H.C. and Taniar, D., "Parallel Query Processing in Object-Oriented Database Systems", *Australian Computer Science Communications*, vol. 17, no. 2, pp. 119-131, 1995.

[14] Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall, 1988

[15] Milne, J., "Power serve!", *Computer Week*, pp. 25-27, January 26, 1996.

[16] Ozkarahan, E., *Database Machines and Database Management*, Prentice-Hall, 1986.

[17] Rahayu, W., Chang, E. and Dillon, T.S., "A Methodology for the Design of Relational Databases from Object-Oriented Conceptual Models Incorporating Collection Types", *Proceedings of the 18th International Conference on Technology of Object-Oriented Languages and Systems TOOLS Pacific*, Melbourne, pp. 13-23, 1995.

[18] Selinger, P.G., "Predictions and Challenges for Database Systems in the Year 2000", *Proceedings of the 19th VLDB Conference*, pp. 667-675, Dublin, Ireland, 1993.

[19] Taniar, D. and Rahayu, W., "Object-Oriented Collection Join Queries", *Proceedings of the International Conference on Technology Object-Oriented Languages and Systems TOOLS Pacific'96*, Melbourne, pp. 115-125, 1996.

[20] Taniar, D., "Inheritance and Parallelization: Emerging Object-Oriented and Parallel Technologies for High Performance Database Systems", to appear, *Proceedings of the High Performance Computing HPC'97 Asia*, Seoul, Korea, 1997.

[21] Thakore, A.K. and Su, S.Y.W., "Performance Analysis of Parallel Object-Oriented Query Processing Algorithms", *Distributed and Parallel Databases* 2, pp. 59-100, 1994.

[22] Valduriez, P., "Parallel Database Systems: Open Problems and New Issues", *Distributed and Parallel Databases 1*, pp. 137-165, 1993.

[23] Valduriez, P., "Parallel Database Systems: The Case for Shared-Something", *Proceedings of the International Conference on Data Engineering*, pp. 460-465, 1993.