# Parallel Database Processing on the KSR1 Computer

Emy Tseng, David Reiner
emy@ksr.com, reiner@ksr.com

Kendall Square Research Corporation
170 Tracer Lane, Waltham, MA 02154

## ABSTRACT

The Kendall Square Research high performance computer (KSR1) provides a spectrum of parallel database processing techniques to achieve scalability and performance in a shared memory environment. The techniques include running multiple transactions in parallel, decomposing queries into parallel subqueries, running multiple instances of the DBMS and partitioning data over disks. These techniques enable on-line transactions to be run in parallel at high throughput rates and decision-support queries to be parallelized and executed very rapidly.

This paper focuses upon two of the parallel database processing techniques used on the KSR1 -- the Kendall Square Query Decomposer and the Oracle Parallel Server. The Query Decomposer intercepts costly decision support queries and decomposes them into subqueries which are executed in parallel. Parallel Server enables multiple ORACLE instances to run simultaneously on the same database.

## THE KSR1

The KSR1 is a highly parallel computer system designed to be scalable to hundreds of processors. Each processor is a RISC-style superscalar 64-bit unit operating at 40 peak MIPS and 40 peak MFLOPS. On the KSR1, the ALLCACHE™ memory hardware maps distributed memory into a uniform address space. This achieves the simplicity of programming in a shared memory environment while preserving the benefits of distributed memory -- scalability and high performance.

Targeted at technical, scientific, and commercial users, the KSR1 is a general-purpose computer. Its operating system is an extension of OSF/1, and its software suite includes C, C++, Fortran, COBOL, and TUXEDO/T, in addition to the ORACLE7 DBMS.

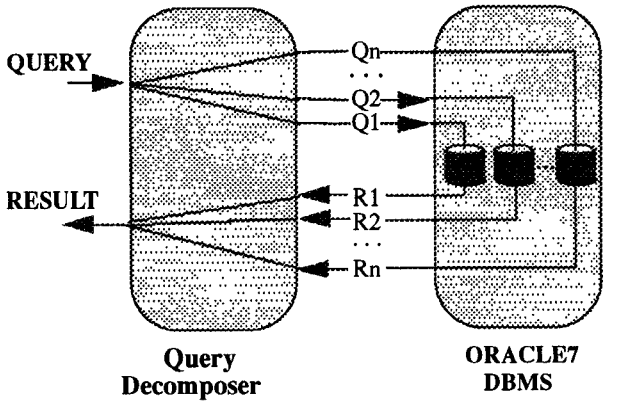

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0453...$1.50

## QUERY DECOMPOSER

The Kendall Square Query Decomposer leverages the parallelism of the KSR1 computer to greatly speed the execution of decision support queries. The Query Decomposer works in conjunction with the underlying ORACLE7 RDBMS in order to automatically parallelize SQL queries.

Using existing ORACLE data-striping techniques, the database administrator partitions large database tables over multiple disks in order to maximize parallel reads from disk. There may be tens or even hundreds of partitions for a given table. These partitions may be populated randomly or with a hash function that scatters tuples into small clusters.

**Figure 1 Flow of Processing with KSR Query Decomposer**




When a query is submitted to ORACLE, the Query Decomposer intercepts it at a common processing point (see Figure 1 above). Queries that do not benefit from decomposition are passed to ORACLE unchanged. Those that do are transformed based on the data access strategy selected by the ORACLE query optimizer. The Query Decomposer parallelizes index scans where appropriate, as well as full table scans against large tables. The Query Decomposer generates a number of subqueries to match the underlying physical data parti-

tions of one of the partitioned tables, called the "driving" table.

Decisions made automatically by the Query Decomposer include the number of subqueries, the choice of the driving table, the minor query transformations to handle aggregate functions, and the method of combining subquery results. Each subquery looks very much like the original query, with changes that include an additional predicate to restrict the subquery to just one data partition (see Figure 2 below).

**Figure 2 Straightforward Decomposition of Query**

Assume department (dept) and employee (emp) tables are partitioned into multiple files across disks. Assume an index on dept.deptno, with none on emp.title. The initial query is:

```
SELECT empname, empno, deptloc
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND emp.title = 'MTS'
```

The ORACLE7 query optimizer reports that it will use a nested loop join with emp as the outer table and dept as the inner one. The ith subquery generated by the Query Decomposer is below, with "i" and "i+1" replaced by actual values:

```
SELECT empname, empno, deptloc
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND emp.title = 'MTS'
AND emp.rowid >= '0.0.i'
AND emp.rowid <'0.0.i+1'
```

Each subquery executes against a different partition of emp, based on ORACLE's interpretation of the two conditions on emp.rowid. Subquery results are combined by the Query Decomposer using a Union All operation.

By default, one subquery is generated per partition of the driving table, but this ratio may be adjusted (through query- or application-level directives to the Query Decomposer) so that each subquery covers multiple partitions if desired. This allows flexibility in applying parallel machine resources to a particular query or class of queries.

The Query Decomposer submits the subqueries in parallel to the ORACLE server over multiple, coordinated connections, so subqueries see a consistent view of the database. All of the subqueries are executed in parallel, each accessing only its own part of the driving table, and doing further joins as required. Since partitions are approximately equal in size, initial partition scans are unaffected by data skew. Common lookup tables and index pages are brought in only once from disk and are shared among subqueries when needed.

The Query Decomposer combines subquery results and returns these results to the application that submitted the original query. If the original query calls for grouped or sorted data, so do the subqueries and the Query Decomposer combines the results. This effectively gives users a parallel sorting capability for decision support queries. Aggregate functions are correctly computed when subquery results are combined (see Figure 3 below).

**Figure 3 Query with Aggregate Function**

Assumptions are as in the previous query. The initial query is:

```
SELECT deptno, AVG(salary)
FROM emp GROUP BY deptno
```

The ith subquery generated is:

```
SELECT deptno, SUM(salary), COUNT(salary)
FROM emp
WHERE emp.rowid >= '0.0.i'
AND emp.rowid <'0.0.i+1'
GROUP BY deptno
```

Each subquery executes against a different partition of emp. The Query Decomposer creates a two-column temporary table, filled once for each deptno with SUM(salary), COUNT(salary) tuples. There is at most one such tuple for each partition.

Subquery results are combined, calculating average salaries with the following query which is executed once against the temporary table for each deptno:

```
SELECT (SUM(sum_col)/(SUM(count_col))
FROM temporary table
```

Conceptual and logical database design are unchanged when the Query Decomposer is used. Physical database design is slightly different in that large tables should be striped across disks by the DBA, with one table partition per disk drive. Also, hash clusters, supported in ORACLE7, are an especially attractive design alternative. If tables are relatively static in size, hash clustering may be used with small hash buckets (called "scatter clustering"), and with a secondary index whose initial fields are the same as those used in calculating the hash key. This combination maximizes IO parallelism for subqueries, while minimizing total IOs.

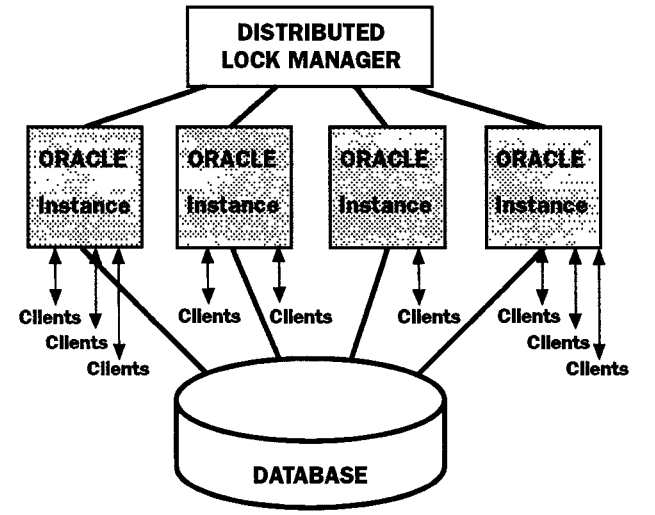
## PARALLEL SERVER

The ORACLE Parallel Server (V7) is a software capability that allows multiple ORACLE instances to execute concurrently against the same database in order to provide greater database throughput. An ORACLE instance consists of the collection of processes used to run the ORACLE RDBMS.

On the KSR1, different instances coordinate database access through the Kendall Square Distributed Lock Manager (see Figure 4 below). The Distributed Lock Manager

(DLM) supports communication among the instances and coordinates modifications of the database blocks. The DLM consists of one or more lock manager processes.

**Figure 4 Parallel Server Architecture**




Parallel Server uses the DLM to maintain cache coherency and concurrency control. Every instance has its own buffer cache in which copies of the database blocks are stored. Each instance acquires locks on the database blocks. An instance only releases a lock if another instance needs to modify that block. The DLM coordinates the lock requests and notifications ensuring a consistent view of that database block among the different instances.

The DLM also coordinates the use of transaction locks. Each transaction holds a distributed lock in exclusive mode. If another process needs that transaction to commit before continuing, it attempts to acquire that lock and waits until it can. When the transaction commits, the lock is released, thereby allowing the waiting processes to continue.

On massively parallel machines with a message-passing architecture, one instance of ORACLE runs per processing node. In contrast, KSR1's shared memory environment allows the database administrator (DBA) to run as many or as few ORACLE instances as needed. When transaction rates become very high, one single instance of ORACLE may not take advantage of all the processing power that the KSR1 offers. This is due to latch contention and other database throughput limitations in the underlying DBMS. In this case, Oracle's Parallel Server facilitates processing a database at extremely high transaction rates.

The DBA has flexibility in determining the number of ORACLE instances that are needed, and in configuring the ratio of ORACLE application clients to instances to handle the load on the database. Instance boundaries can be drawn according to application needs rather than hardware limitations. Also, the DBA can let the KSR1 OS dynamically allocate resources to the different instances on demand or can dedicate specific resources to an instance. This flexibility allows simpler and more effective system administration. The DBA is free to tune Parallel Server in order to optimize performance for any collection of applications.

## CONCLUSION

No single parallel technique is sufficient to handle diverse transactions and very high throughput rates against a database. Thus, the KSR1 provides a spectrum of parallel database processing techniques to achieve scalability and high performance. This paper has focused upon two of the parallel database processing techniques used on the KSR1 -- the Kendall Square Query Decomposer and ORACLE Parallel Server.

## REFERENCES

1. Kendall Square Research Corporation. *KSR1 Principles of Operation*
2. ORACLE Corporation. *ORACLE RDBMS Version 7 Parallel Server Administrator's Guide*
3. D. Reiner, J. Miller, and D. Wheat. "The Kendall Square Query Decomposer", *Proceedings, Second Parallel and Distributed Information Systems Conference*, San Diego, 1993.