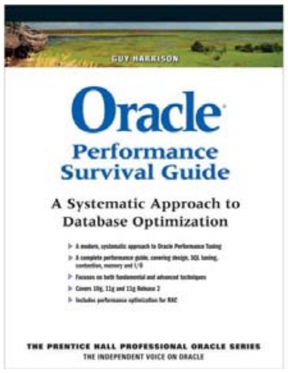

GUY HARRISON
Oracle
Performance Survival Guide
A Systematic Approach to Database Optimization
THE PRENTICE HALL PROFESSIONAL ORACLE SERIES
THE INDEPENDENT VOICE ON ORACLE

Oracle Parallel SQL

Part 1

Extracted from the book "Oracle Performance Survival Guide" by Guy Harrison

Parallel SQL allows a SQL statement to be processed by multiple threads or processes simultaneously.

Today's widespread usage of dual and quad core processors means that even the humblest of modern computers running an Oracle database will contain more than one CPU. While desktop and laptop computers might have only a single disk device, database server systems typically have database files spread – striped – across multiple independent disk devices. Without parallel technology – when an SQL statement is processed in *serial* – a session can only make use of one of these CPUs or disk devices at a time. Parallel processing can improve the performance of suitable SQL statements to a degree which is often not possible by any other method. Parallel processing is available in Oracle Enterprise Edition only.

This is first of a series of articles in which we'll look at how Oracle can parallelize SQL statements and how you can use this facility to improve the performance of individual SQLs or the application as a whole.

Understanding parallel SQL

In a serial - non-parallel - execution environment, a single process or thread[1] undertakes the operations required to process your SQL statement and each action must complete before the succeeding action can commence. The single Oracle process may only leverage the power of a single CPU and read from a single disk at any given instant. Because most modern hardware platforms include more than a single CPU and because Oracle data is often spread across multiple disks, serial SQL execution is unable to take advantage of all of the available processing power.

For instance, consider the following SQL statement:

```
  SELECT   *
    FROM   sh.customers
ORDER BY   cust_first_name, cust_last_name, cust_year_of_birth
```

If executing without the parallel query option, a single process would be responsible for fetching all the rows in the CUSTOMERS table. The same process would be responsible for sorting the rows to satisfy the ORDER BY clause. Figure 1 illustrates the workflow.

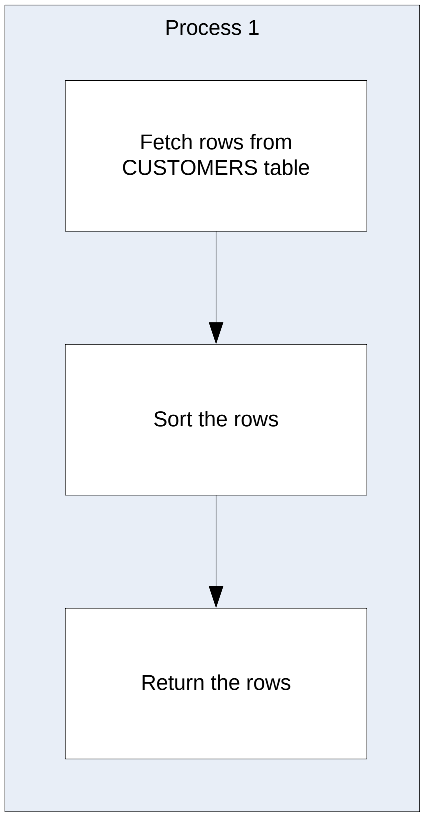


FIGURE 1 SERIAL EXECUTION OF A SQL STATEMENT

We can request that Oracle execute this statement in parallel by using the PARALLEL hint:

[1] A process is an unit of execution with its own private memory. A thread is also a unit of execution, but shares memory with other threads within a process.

```
  SELECT   /*+ parallel(c,2) */ *
    FROM   sh.customers c
ORDER BY   cust_first_name, cust_last_name, cust_year_of_birth
```

If parallel processing is available, the CUSTOMERS table will be scanned by two processes in parallel. A further two processes will be employed to sort the resulting rows. A final process – the session which issued the SQL in the first place - will combine the rows and return the result set. Figure 2 illustrates this sequence of events.

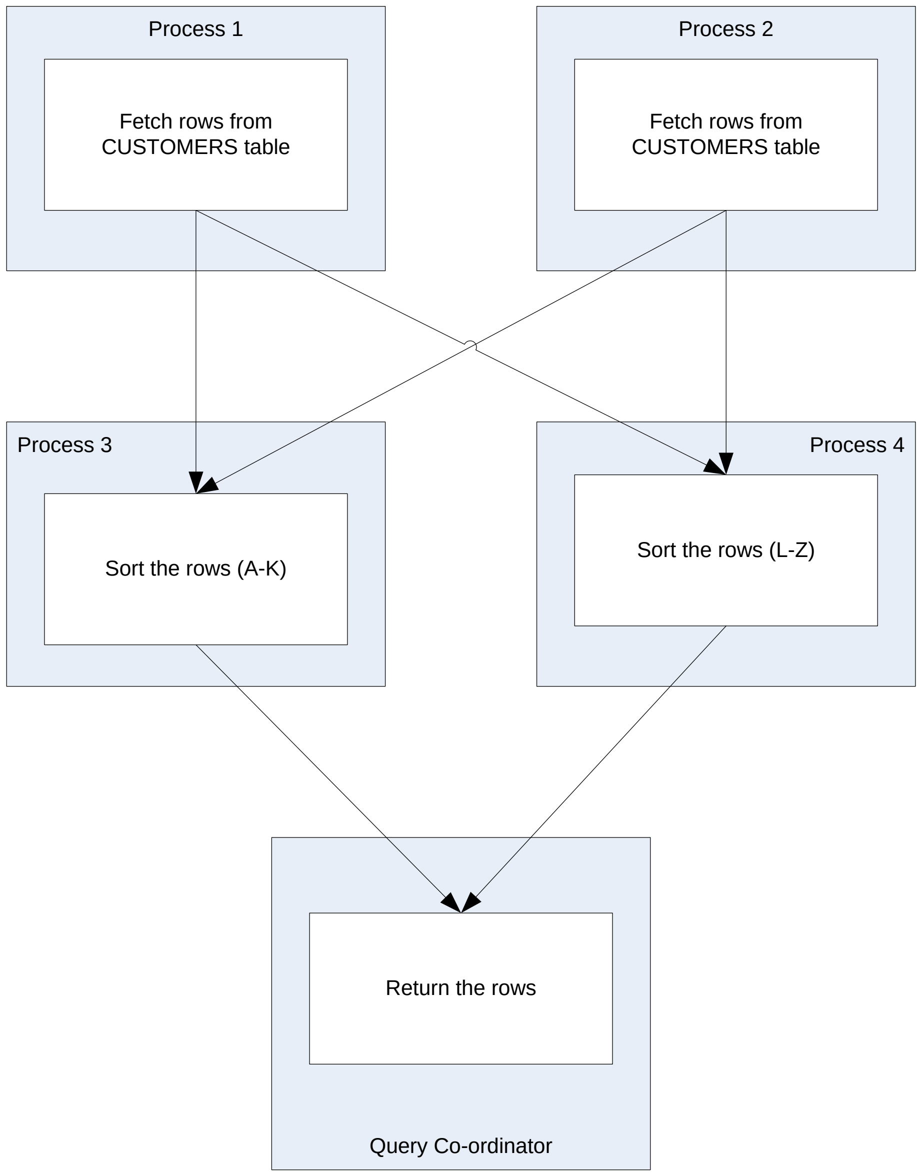


FIGURE 2 PARALLEL EXECUTION

Oracle supports parallel processing for a wide range of operations, including queries, DDL and DML:

- Queries that involve table or index range scans.
- Bulk insert, update or delete operations.
- Table and index creation.

Parallel processes and the Degree of Parallelism

The Degree of Parallelism (DOP) defines the number of parallel streams of execution that will be created. The number of parallel processes is more often twice the degree of parallelism. This is because each step in the execution plan needs to feeds data into the subsequent step so two sets of processes are required to maintain the parallel stream of processing.

Figure 3 (page 5) shows how parallel slaves are allocated for a DOP of two.

Parallel slave pool

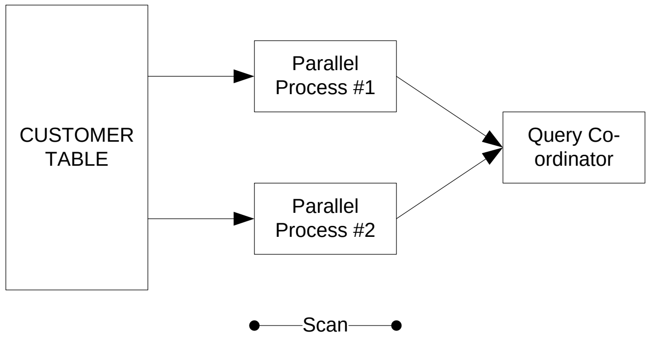
The Oracle server maintains a “pool” of parallel slave processes available for parallel operations. The Database configuration parameters PARALLEL_MIN_SERVERS and PARALLEL_MAX_SERVERS determine the initial and maximum size of the pool.

Parallel query IO

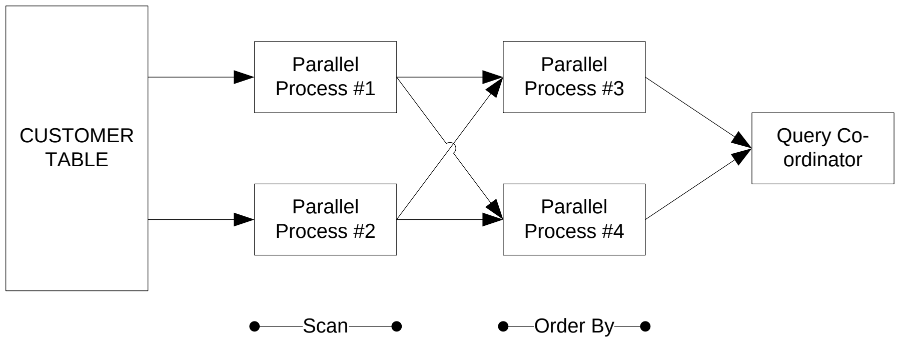
The buffer cache helps reduce disk IO by buffering frequently accessed data blocks in shared memory. Oracle has an alternate IO mechanism – *direct path* IO – which it can use if it determines that that it would be faster to bypass the buffer cache and perform the IO directly.

In Oracle 10g and earlier, parallel query always uses direct path IO, and serial query will always use buffered IO. In 11g, Oracle can use buffered IO for parallel query (from 11.2 onwards) and serial queries may use direct path IO. However, it remains true that parallel queries are less likely to use buffered IO and may therefore have a higher IO cost than serial queries.

```
SELECT /*+ parallel(c,2) */ *
  FROM customers c
```



```
SELECT /*+ parallel(c,2) */ *
  FROM customers c
 ORDER BY cust_last_name,cust_first_name
```



```
SELECT /*+ parallel(c,2) */ cust_last_name,count(*)
  FROM customers c
 GROUP BY cust_last_name
 ORDER BY 2 desc
```

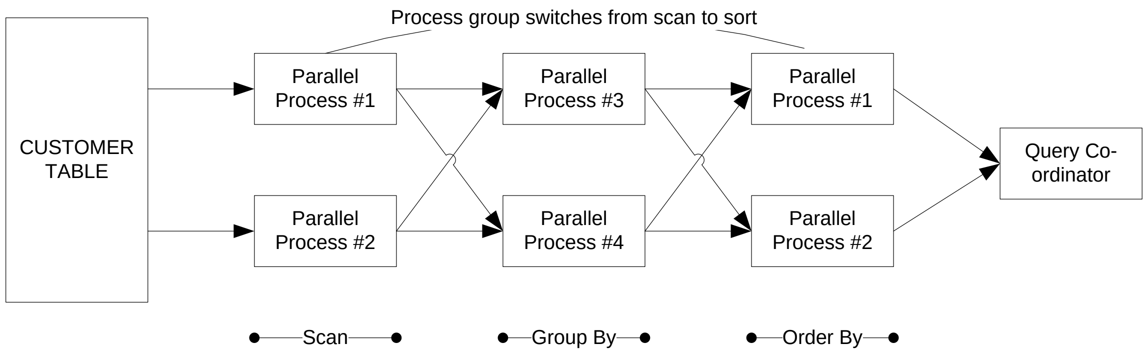


FIGURE 3 PARALLEL PROCESS ALLOCATION FOR A DEGREE OF PARALLELISM (DEGREE OF PARALLELISM) OF 2

Parallel performance gains

The performance improvements that you can expect to obtain from parallel SQL depend on the suitability of your host computer, Oracle configuration and the SQL statement. If all the conditions for parallel processing are met, you can expect to get substantial performance improvements in proportion to the Degree of Parallelism employed.

On many systems, the limit of effective parallelism will be determined by segment spread, not by hardware configuration. For instance, if you have 32 CPUs and 64 independent disk devices, you might hope for effective parallelism up to at least a DOP of 32 or maybe even 64. However, if the table you are querying is spread over only 6 disks, then you are likely to see performance improvements reduce as you increase the Degree of Parallelism beyond 6 or so.

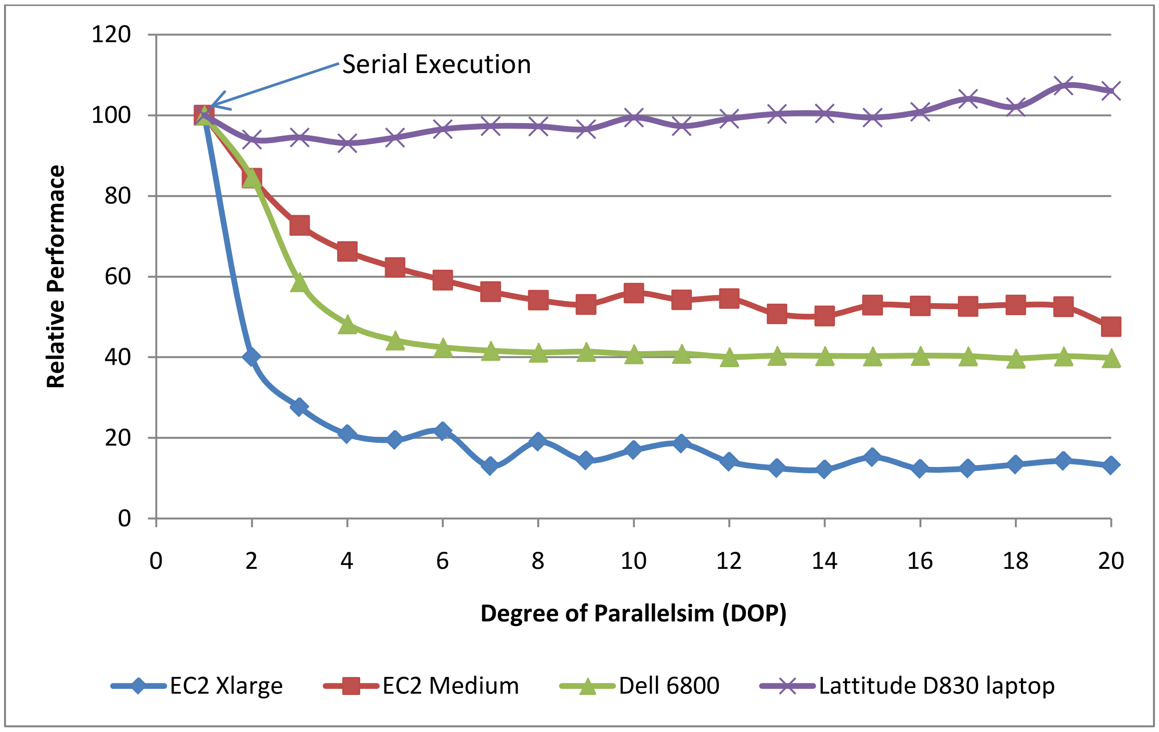


FIGURE 1 IMPROVEMENT GAINS FOR VARIOUS DEGREE OF PARALLELISMS ON VARIOUS HOST CONFIGURATIONS

Figure 4 illustrates the improvements gained when increasing the Degree of Parallelism for a SQL statement that performs a table scan and GROUP BY of a single table. The host configurations shown are:

- An Amazon CPU intensive Extra Large EC2 image. This is a virtual server running in Amazon's AWS cloud that has the equivalent of 8x2.5 GHz CPUs and has storage on a widely striped SAN.
- An Amazon CPU intensive Medium EC2 image. This is similar to the extra large image, but has only 2 CPUs
- A Dell 6800 4 CPU server, with disk storage on a widely striped SAN using ASM.
- A Dell latitude D830 laptop (my laptop). It's dual core, but all data files are on a single disk.

In each case, the parallel SQL was the only SQL running.

These examples show that for suitably configured systems, performance gains were greater the more CPUs that were available. However, attempting to use parallel on a host which is unsuitable (as in my laptop) is futile at best and counter-productive at worst.

The performance gains achieved through parallale processing are most dependent on the hardware configuration of the host. To get benefits from parallel procesisng, the host should possess multiple CPUs and data should be spread across multiple disk devices.

Deciding to use parallel processing

A developer once saw me use the parallel hint to get a rapid response to an ad-hoc query. Shortly thereafter, every SQL that developer wrote included the parallel hint and system performance suffered as the database server became overloaded by excessive parallel processing.

The lesson is obvious – if every concurrent SQL in the system tries to use *all* of the resources of the system then parallel will make performance worse, not better. Consequently, we should use parallel only when doing so will improve performance without degrading the performance of other concurrent database requests.

Here are some of the circumstances under which you can effectively use parallel SQL:

Your server computer has multiple CPUs

Parallel processing will usually be most effective if the computer which hosts your Oracle database has multiple CPUs. This is because most operations performed by the Oracle server (accessing the Oracle shared memory, performing sorts, disk accesses) require CPU. If the host computer has only one CPU, then the parallel processes may contend for this CPU and performance might actually decrease.

Almost every modern computer has more than one CPU; dual-core (2 CPUs in a single processor slot) configurations are the minimum found in systems likely to be running an Oracle server including the desktops and laptops running development databases. However, databases running within Virtual machines may well be configured with only a single CPU.

The data to be accessed is on multiple disk drives

Many SQL statements can be resolved with few or no disk accesses when the necessary data can be found in the Oracle buffer cache. However, full table scans of larger tables—a typical operation to be parallelized—will tend to require significant physical disk reads. If the data to be accessed resides on a single disk, then the parallel processes will line up for this disk and the advantages of parallel processing might not be realized.

Parallelism will be maximized if the data is spread evenly across the multiple disk devices using some form of striping.

The SQL to be parallelized is long running or resource intensive

Parallel SQL suits long running or resource intensive statements. There is an overhead in activating and coordinating multiple parallel query processes, and in co-coordinating the flow of information between these processes. For short lived SQL statements, this overhead might be greater than the total SQL response time.

Parallel processing is typically used for:

- Long-running reports.
- Bulk updates of large tables.
- Building or rebuilding indexes on large tables.
- Creating temporary tables for analytical processing.
- Rebuilding a table to improve performance or to purge unwanted rows.

Parallel processing is not usually suitable for transaction processing environments. In these environments, large numbers of users process transactions at a high rate. Full use of available CPUs is already achieved because each concurrent transaction can use a different CPU. Implementing parallel processing might actually degrade overall performance by allowing a single user to monopolize multiple CPUs. In these environments, parallel processing would usually be limited to MIS reporting, bulk loads, index or table rebuilds which would occur at off-peak periods.

Parallel processing is suitable for long running operations in low-concurrency environments. Parallel processing is less suitable for OLTP style SQLs.

The SQL performs at least one full table, index or partition scan

Parallel processing is generally restricted to operations that include a scan of a table, index or partition. However, the SQL may include a mix of operations, only some of which involve scans. For instance, a nested loops join that uses an index to join two tables can be fully parallelized providing that the driving table is accessed by a table scan.

Although queries that are driven from an index lookup is not normally parallelizable, if a query against a partitioned table is based on a local partitioned index, then each index scan can be performed in parallel against the table partition corresponding to the index partition. We'll see an example of this in the next installment of this series of articles.

There is spare capacity on your host

You are unlikely to realize the full gains of parallel processing if your server is at full capacity. Parallel processing works well for a single job on an underutilized, multi-CPU machine. If all CPUs on the machine are busy, then your parallel processes will bottleneck on the CPU and performance will be degraded.

Controlling parallel query

Oracle tries to automate the configuration of the system to maximize the performance of parallel operations. However, there's still a lot of scope for manually tweaking the database and SQL for optimal parallel performance.

Determining the degree of parallelism

An optimal degree of parallelism (DOP) is critical for good parallel performance. Oracle determines the DOP as follows:

- If parallel execution is indicated or requested, but no Degree of Parallelism is specified, then the default Degree of Parallelism is set to twice the number of CPU cores on the system. For a RAC system, the Degree of Parallelism will be twice the number of cores in the entire cluster. This default is controlled by the configuration parameter PARALLEL_THREADS_PER_CPU.
- From Oracle 11g release 2 onwards, If PARALLEL_DEGREE_POLICY is set to AUTO, then Oracle will adjust the Degree of Parallelism depending on the nature of the operations to be performed and the sizes of the objects involved.
- If PARALLEL_ADAPTIVE_MULTI_USER is set to TRUE, then Oracle will adjust the degree of parallel based on the overall load on the system. When the system is more heavily loaded, then the degree of parallelism will be reduced.
- If PARALLEL_IO_CAP is set to TRUE in 11g or higher, then Oracle will limit the Degree of Parallelism to that which the IO subsystem can support. The IO subsystem limits can be calculated by using the procedure DBMS_RESOURCE_MANAGER.CALIBRATE_IO.
- A degree of parallelism can be specified at the table or index level by using the PARALLEL clause of CREATE TABLE, CREATE INDEX, ALTER TABLE or ALTER INDEX.
- The PARALLEL hint can be used to specify the degree of parallelism for a specific table within a query
- Regardless of any other setting, the degree of parallelism cannot exceed that which can be supported by PARALLEL_MAX_SERVERS. For most SQL statements, the number of servers required will be twice the Degree of Parallelism.

As we saw in Figure 4, increasing the degree of parallelism beyond the optimal point fails to result in further performance increases. However, increasing the DOP beyond optimal can have a significant negative effect on overall system performance. Although the SQL being parallelized may not degrade significantly as the Degree of Parallelism increases, load on the system continues to increase and can cause other SQLs running concurrently to suffer reduced response time.

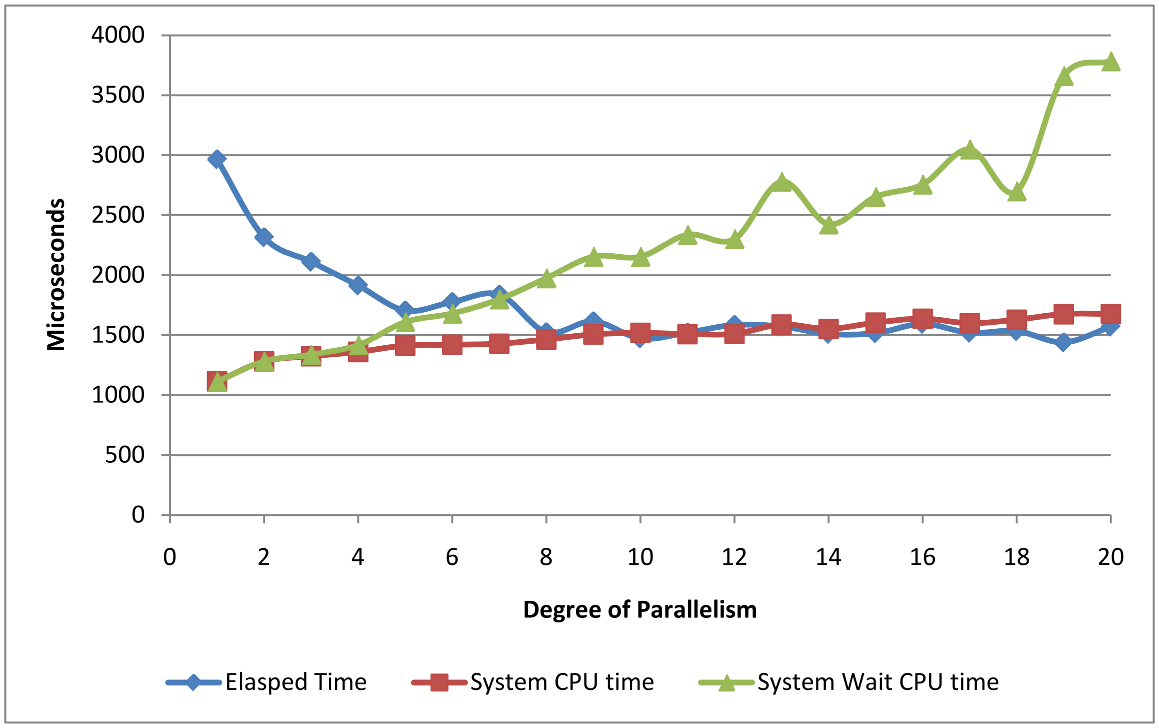


FIGURE 2 INCREASING THE DOP CAUSES INCREASES IN SYSTEM CPU WAIT TIMES

Figure 5 shows how increasing the DOP influences CPU utilization. As we hit the optimal Degree of Parallelism – about 8 for this system – the improvement in elapsed time flattens out. However the time sessions spend waiting for CPU to become available continues to increase. Other sessions wishing to access the CPU will need to wait, resulting in degraded response time.

Increasing the Degree of Parallelism beyond the optimal level may overload the host, degrading the performance of other SQLs.

The parallel hints

The PARALLEL hint can be used to invoke parallel processing. In its simplest form, the hint takes no argument as in the following example:

```
SELECT /*+ parallel */ *  FROM sh.sales s
```

It's legal, but not always necessary to specify a table name or alias in the hint:

```
SELECT /*+ parallel(s) */  *  FROM sh.sales s
```

The hint can request a specific degree of parallelism:

```
SELECT /*+ parallel(s,8) */ * FROM sh.sales s;
```

The NOPARALLEL hint can be used to suppress parallel processing:

```
SELECT  /*+ noparallel */ COUNT ( * ) FROM sales;
```

In 11G release 2, the AUTO option allows you to request that the AUTO setting for PARALLEL_DEGREE_POLICY be used to calculate the Degree of Parallelism:

```
SELECT  /*+ parallel(auto) */ COUNT ( * ) FROM sales;
```

Parallel configuration parameters

Determining the optimal Degree of Parallelism, especially when taking concurrent system activity into account, is a daunting task. Luckily, Oracle has invested significant effort into automating the process. Each release of Oracle has increased the level of intelligent automation of parallel configuration. In general, you should try Oracle's automation before attempting to manually configure automatic processing.

Nevertheless, significant tweaking is possible; Table 1 lists the significant configuration parameters that you can adjust in order to optimize parallel SQL.

NAME	DESCRIPTION
parallel_adaptive_multi_user	When set to TRUE, Oracle will adjust the Degree of Parallelism to account for the load on the system. On a heavily loaded system, Oracle will reduce the Degree of Parallelism from the requested or default degree.
parallel_degree_limit	In Oracle11g Release 2 and higher, places an absolute limit on the DOP that can be achieved. A value of CPU prevents the DOP from exceeding that specified by parallel_threads_per_cpu. A value of IO sets the maximum to the IO limit determined by running DBMS_RESOURCE_MANAGER.CALIBRATE_IO. AUTO allows Oracle to select a value. An integer value corresponding to a specific DOP might also be specified.
parallel_degree_policy	In 11G release 2 and onwards, this parameter controls the means by which the Degree of Parallelism will be calculated. MANUAL equates to the behaviour in 11.1 and earlier. If AUTO, then the Degree of Parallelism will be calculated based on the types of operations in the SQL statement and the sizes of the tables. AUTO also allows parallel queries to fetch data from the buffer cache rather than using direct path IO and will queue parallel processes if the requested degree of parallel execution is not immediately available.
parallel_execution_message_size	Sets the size of buffers for communication between the processes involved in parallel processing.
parallel_force_local	From 11.2 onwards, this parameter – if set to TRUE – suppresses multi-instance parallelism on RAC clusters.

parallel_io_cap_enabled	This 11G parameter if set to TRUE will limit the Degree of Parallelism to that which Oracle thinks the IO subsystem can support. To use the parameter, you should first use DBMS_RESOURCE_MANAGER.CALIBRATE_IO to determine IO limits.
parallel_max_servers	The maximum number of parallel servers that can be started. This provides an absolute limit on the amount of concurrent parallel operations than can execute.
parallel_min_percent	If set to non-zero, this parameter determines the minimum acceptable Degree of Parallelism for a query. If the Degree of Parallelism requested or determined cannot be provided due to system load or other parallel processes which are using the parallel server pool, then the Degree of Parallelism will be reduced only to the value of PARALLEL_MIN_PERCENT. For instance, if your query required 8 and only 5 were available (5/8=62%), then your query would execute in parallel if PARALLEL_MIN_PERCENT was below 62. If PARALLEL_MIN_PERCENT was above 62, your statement will either terminate with an error or – if PARALLEL_DEGREE_POLICY is set to AUTO – will be queued for later execution.
parallel_min_servers	The minimum number of parallel servers – the number that will be initialized when the database is first started.
parallel_min_time_threshold	Specifies the amount of elapsed time (in seconds) required for a SQL statement to be automatically parallelized. If the estimated elapsed time of an SQL statement exceeds the threshold then Oracle will automatically parallelize the SQL. The default of AUTO results in Oracle automatically calculating a value.
parallel_threads_per_cpu	Sets the number of parallel threads that can be applied per CPU. Oracle will generally restrict the Degree of Parallelism so that this limit is not exceeded.

TABLE 1 PARALLEL CONFIGURATION PARAMETERS

Summary

In this first installment of a two part series we've looked at the parallel execution facilities provided by the Oracle RDBMS, how to use these to improve SQL performance and how to optimize the performance of SQL running in parallel.

Parallel processing uses multiple processes or threads to execute a single SQL statement. Providing that the system is suitably configured, parallel processing can result in big improvements in SQL throughput, though at the cost of an increased load on the system.

The Degree of Parallelism defines the amount of parallelism that is applied to your SQLs. For simple SQLs the Degree of Parallelism will equate to the number of parallel processes, but in most non-trivial statements twice as many processes will be required in order to achieve a pipeline of parallel processing.

Parallel processing may be indicated if:

- The database server has multiple CPUs
- The data is distributed across multiple disk devices
- The SQL is long running or resource intensive
- There are free resources available on the system to support the additional overhead associated with parallel processing.
- The SQL involves a full table or index scan, or locally partitioned index lookups.

In the next installment, we'll focus on monitoring and optimizing parallel SQL statements.

Guy Harrison is a director of Research and Development at Quest Software, and has over 20 years of experience in database design, development, administration, and optimization. Guy is an Oracle ACE, and is the author of Oracle Performance Survival Guide *(Prentice Hall, 2009)* and MySQL Stored Procedure Programming *(OReilly with Steven Feuerstein) as well as other books, articles and presentations on database technology. Guy is the architect of Quest's Spotlight® family of diagnostic products and has led the development of Quests Toad® for Cloud Databases. Guy can be found on the internet at http://www.guyharrison.net/, on e-mail at guy.harrison@quest.com and is @guyharrison on twitter.*