Detailed Characterization of a Quad Pentium Pro Server Running TPC-D[1]

Q. Cao, **P. Trancoso**[2], **J.-L. Larriba-Pey**[3], **J. Torrellas**, **R. Knighten**[4] and **Y. Won**[4]

Department of Computer Science
University of Illinois at Urbana-Champaign, IL 61801
{qiangcao,torrellas}@cs.uiuc.edu trancoso@intercol.edu
larri@ac.upc.es {bob.knighten,youjip.won}@intel.com
http://iacoma.cs.uiuc.edu

Abstract

While database workloads consume a major fraction of the cycles in today's machines, there are only a few public-domain performance studies that characterize in detail how these workloads exercise the machines. This fact is due to the complexity of setting up and tuning database workloads, the high cost of the equipment required to evaluate them, and the frequent use of proprietary systems.

In this paper, we help redress this problem by presenting a detailed performance characterization of the TPC-D benchmark running on a 4-processor Pentium Pro SMP multiprocessor with Windows NT and Microsoft's SQL Server. We use the Pentium Pro built-in hardware event counters and software tools that monitor system activity. Our results show that TPC-D queries have a relatively low CPI. The CPIs, which are 1.27 on average for the 17 read-only queries, are comparable to values observed for technical workloads. The major sources of processor stall cycles are the instruction fetch bottleneck and data misses in the secondary cache. Kernel time is negligible, as queries spend less than 6% of their time on average in the kernel. Other results show that branch prediction is effective in TPC-D and that the exclusive state in the cache tags is largely unnecessary. Finally, we compare our results to the ones published for TPC-C.

1 Introduction

Commercial workloads like databases and web servers are the dominant class of workloads currently installed in server-type machines. Moreover, they are expected to further increase their lead in installed base in the next decade [12]. To a large extent, they are responsible for the current dramatic increase in sales of small-scale shared-memory multiprocessor machines [8].

Surprisingly for such an important class of workloads, only limited information about their performance characteristics is available, at least in the public domain. Although transaction processing benchmarks have finally been standardized (the TPC benchmarks), they have not been analyzed as much as their technical counterparts, namely the SPEC and Linpack benchmark suites. Major reasons for this are the complexity of setting up and tuning commercial benchmarks, the relatively high cost of the equipment required to evaluate them, and the frequent use of proprietary systems like DataBase Management Systems (DBMS). This is unfortunate, given that technical and commercial workloads may exercise computer systems in different ways.

Fortunately, in the last two years, there have been several public-domain characterization analyses of commercial workloads. For example, Trancoso *et al.* [13] performed a simulation-based study of 3 queries from the TPC-D benchmark [15] on a shared-memory multiprocessor. The TPC-D benchmark models a Decision Support System (DSS) application like stock management in a large department store. The study focused on memory access issues, and pointed out that the data access patterns have significant spatial locality but little temporal locality.

Another study by Barroso *et al.* [1] characterized TPC-C [14] and 6 TPC-D queries running on Oracle DBMS, and a web index search engine. These applications were executed on an Alpha-based multiprocessor. The TPC-C benchmark models an On-Line Transaction Processing (OLTP) system like an airline reservation system. The study focused on memory access issues and was performed using a combination of machine monitoring and simulation. It was shown that OLTP and DSS workloads behave differently: OLTP has a larger working set and needs large off-chip caches for good performance. Consequently, machines have to be configured differently for each type of workload. The authors also found that the fraction of kernel time in these workloads is low.

Another study by Keeton *et al.* [6] characterized TPC-C running on an Informix DBMS in a Quad Pentium Pro SMP. They focused on many performance issues, not just on the memory system. They found that several architectural features known to be effective in technical workloads are not as effective in commercial workloads. Such features include out-of-order execution, superscalar issue and retire, and branch prediction. However, they found that caches are effective in reducing memory traffic.

There have been other simulation-based studies. For example, the following two studies have used TPC-B (which is similar to TPC-C) and query 6 from TPC-D. Lo *et al.* [7] studied the effects of simultaneous multithreading (SMT) on database workloads. The authors concluded that an 8-context SMT processor can achieve a 3-fold increase in instruction throughput. Ranganathan *et al.* [10] studied the impact of out-of-order issue processors on the performance of database workloads. Their results indicate that out-of-order issue, together with multiple instruction issue, is effective in improving the performance of database workloads.

The contribution of our paper is to perform a detailed architectural characterization of the TPC-D benchmark running on a real machine. While a similar characterization has been done for TPC-C, no such work has been done for TPC-D. All past characterization studies of TPC-D have focused on the memory system only, have examined at most 6 queries and, when obtained on a real machine, have used a reduced-sized database. Instead, in this work, we examine many performance aspects, including CPI, resource stalls, cache misses,

[1]This work was supported in part by the National Science Foundation under grants NSF Young Investigator Award MIP 94-57436, ASC-9612099, and MIP-9619351, DARPA Contract DABT63-95-C-0097, and gifts from Intel Corporation.

[2]Pedro Trancoso is with Intercollege Limassol, Cyprus.

[3]Josep-Lluis Larriba-Pey is with Universitat Politècnica de Catalunya, Barcelona, Spain.

[4]Robert Knighten and Youjip Won are with Intel Corporation, Oregon, USA.

branch prediction, data sharing, bus utilization, and I/O activity. Furthermore, we examine all 17 read-only TPC-D queries, and use a standard-size database. Our platform is a 4-processor Pentium Pro SMP multiprocessor with Windows NT and Microsoft's SQL Server. To collect the data, we use the Pentium Pro built-in hardware counters and software tools that monitor system activity.

Our results show that all TPC-D queries have a relatively low CPI, which is comparable to values observed in popular technical workloads. The major sources of processor stall cycles are the instruction fetch bottleneck and data misses in the secondary cache. We also show that kernel time is a negligible 6% of the execution time, that branch prediction is effective in TPC-D, and that the exclusive state in the cache tags is largely unnecessary. We also present a comparison of our results to published results for TPC-C.

The organization of this paper is as follows. Section 2 describes our experimental setup, Section 3 analyzes the experimental results, and Section 4 concludes.

2 Experimental Setup

In this section we describe the hardware and software setup for our experiments, and the methodology used.

2.1 Hardware Setup

The system used in this work is a Data General AViiON AV 20000 [3], a SMP server with 4 Pentium Pro processors. The building block for this system is an Intel SHV II (Standard High Volume) motherboard, which holds 4 200-MHz Pentium Pro processors. Figure 1 shows a diagram of the motherboard. Each Pentium Pro processor has a split primary cache: an 8-Kbyte four-way set-associative instruction cache and an 8-Kbyte two-way set-associative, dual-ported data cache. In addition, each processor has a unified four-way set-associative 512-Kbyte secondary cache. Caches have 32-byte lines and are non blocking. Each processor is connected to a 66-MHz split-transaction on-board bus. The system is configured with 512 Mbytes of main memory and 2 Gbytes of virtual memory. The system's total disk storage is 80 Gbytes, composed of 20 4-Gbyte disk drives connected using an Adaptec 2941 SCSI adapter.

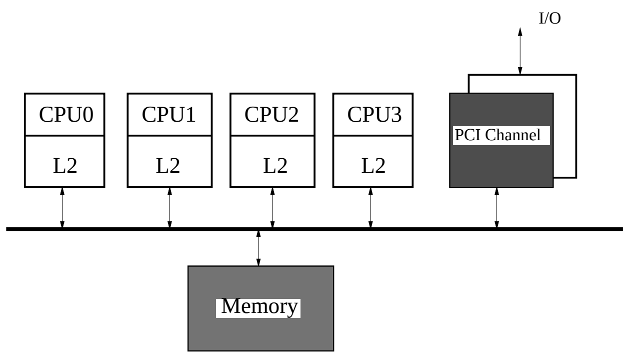


Figure 1: Intel SHV motherboard.

2.2 Software Setup

The software setup required includes the operating system, DBMS, and database application. We describe each of these in the next sections.

2.2.1 Operating System & DBMS

We use Microsoft's Windows NT operating system and SQL Server DBMS. The version of SQL Server used includes automated allocation of main-memory storage to hold temporary results, an optimization engine that supports parallel execution plans and, for improved concurrency of accesses, row-level locking on both data and index pages.

To take advantage of multiprocessor systems, SQL Server supports parallel query execution. Parallelization can reduce the execution time of an operation by having multiple threads working simultaneously and running on different processors. On the other hand, there is some overhead involved, including repetition of operations, synchronization, and resource contention. In SQL Server, parallel execution can be disabled by using a user-specified value for the parallelism threshold. In this case, the parallel version of the query is used only if the query's estimated running time is larger than this threshold. For all the experiments presented in this work, we used a parallelism threshold of 0.

2.2.2 Database Application: TPC-D

TPC-D models a DSS application used by a business to manage, sell, or distribute a product worldwide. The benchmark is composed of 17 different read-only queries named Q1 to Q17. The data in the database is stored in 8 different tables: *Lineitem*, *Customer*, *Supplier*, *Part*, *Partsupp*, *Order*, *Nation*, and *Region*. The benchmark specifies the size of the data tables according to a scale factor. This factor ranges from 1 to 10000, which correspond to a total data size of 1 Gbyte to 10 Tbytes, respectively. In this work, we use a scale factor of 1, which corresponds to a total data size of 1 Gbyte.

Different queries access a different number of tables. For example, while Q1 and Q6 access only one table, Q8 accesses 8 tables at least once. Of the 17 queries, 10 have embedded subqueries. Table 1 lists the main operations performed by the queries. A description of these operations and the main algorithms that can be used can be found in [4].

Query	Scan	Join	Sort	Aggregate
Q1	1	-	1	2
Q2	9	8	1	1
Q3	3	2	1	2
Q4	2	1	1	2
Q5	5	4	2	2
Q6	1	-	-	2
Q7	6	5	1	2
Q8	8	7	1	2
Q9	5	4	3	2
Q10	4	3	2	1
Q11	6	5	3	4
Q12	2	1	2	2
Q13	2	1	1	2
Q14	2	1	-	2
Q15	3	2	1	5
Q16	3	2	2	3
Q17	3	2	-	3

Table 1: Operations in the execution plans of TPC-D queries.

2.3 Methodology

We measure each of the 17 read-only queries of TPC-D individually. In practice, the running time of a query depends on the state of the system before its execution. For example, Q2 can take from 2 to 6 seconds. The longest execution time is obtained when the query is executed right after re-starting

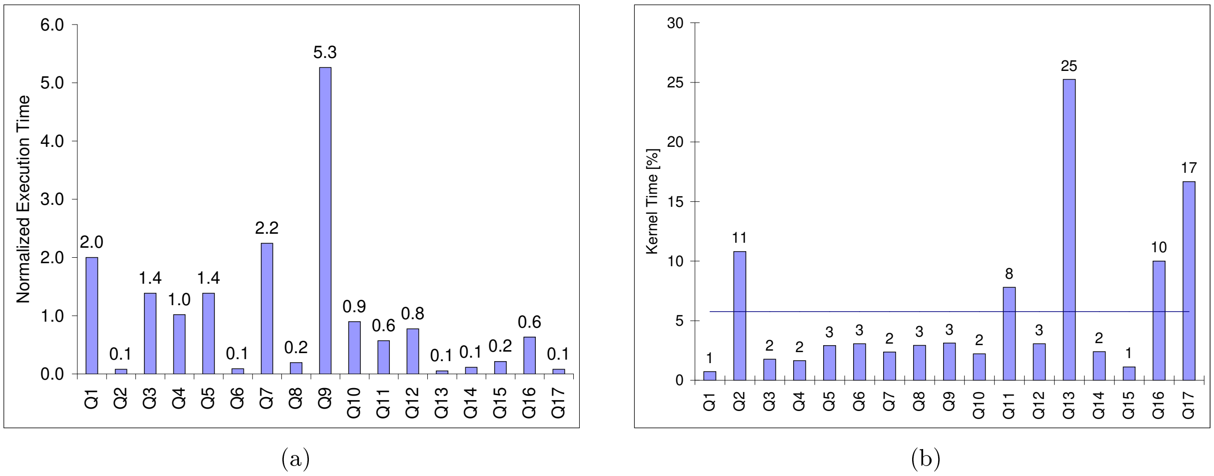


Figure 2: Normalized query execution time (a) and percentage of kernel time for each query (b).

both the machine and the SQL Server. The shortest execution time is obtained when the query is executed after repeated execution of the same query. This last scenario, which we call *warm start*, is the one that we use. We chose this approach because the results are repeatable and the experiment is realistic since, in real situations, a query execution is usually preceded by other query executions.

The Pentium Pro processor has two 40-bit built-in hardware event counters [5]. Each of them can count occurrences of many types of events, including cache misses and retired instructions. We use software tools internal to Intel and Data General to read these counters and process the information at a higher level. For example, we use interval sampling. In this case, we specify a time interval and, at multiples of this time interval, the counters are automatically saved and reset. At the end of the measurement session, the set of saved values can be visualized to obtain the dynamic behavior of the measured events. Furthermore, the tools also call Windows NT system calls to measure higher-level events that are beyond the reach of the hardware event counters, such as I/O activity.

3 Experimental Results

In this section, we analyze the results collected from the execution of the TPC-D queries on the 4-way multiprocessor system. We examine execution time, user and kernel time, CPI, resource stalls, cache misses, branch mispredictions, data sharing, bus utilization, and I/O activity. Finally, we compare TPC-D to TPC-C.

3.1 Execution Time

The execution time of each query is presented in Figure 2-(a). Because our goal is to study query behavior and not to benchmark the machine or DBMS, we compute the average execution time for the 17 queries and normalize each query's execution time to that average value.

Figure 2-(a) shows that different queries have quite different execution time. The execution time ranges from 0.1 to 5.3 times the average value. Query Q9 has the longest execution time in our experiment. This query has one select operation on table *Part* that has a low selectivity. A low selectivity means that the number of output rows is not much smaller than the number of input rows. Therefore, the rest of the operations (join, sort, and aggregate operations) will have more work. In addition, this query joins six tables, four of which are the ones with the largest sizes in the benchmark.

3.2 User and Kernel Time

We now analyze the contribution of user and kernel time in the total execution time. In all queries, the SQL Server DBMS runs as a user process in the Windows NT operating system. Most of the code is executed in user mode, except for calls to the Windows NT *Service Layer*, the *Executive* routines, and the *Kernel* [11].

Figure 2-(b) shows the percentage of the execution time spent in the kernel. The difference between 100 and the bars shown in the figure is the percentage of user time. The horizontal line in the figure is the average percentage of kernel time for all 17 queries. From this figure, we observe that the percentage of kernel time is very small for most of the queries. For 12 out of 17 queries, the percentage of kernel time is smaller than 4%. This result agrees Barroso *et al.* [1], who found that the kernel time for six TPC-D queries running Oracle DBMS was negligible.

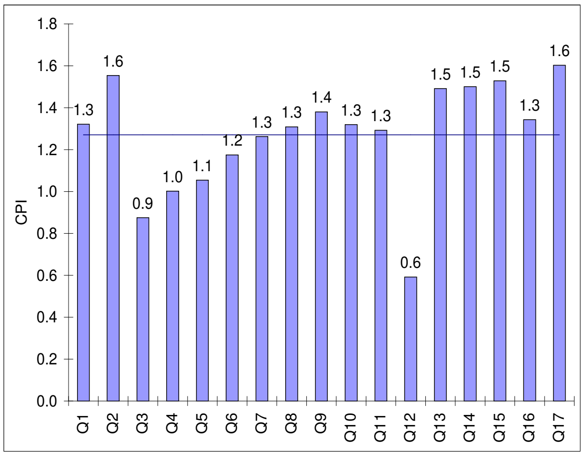
The likely reason for the noticeable kernel time in queries Q2, Q13, and Q17 is their short execution time. For example, Q13, which has 25% of kernel time, has the shortest execution time. As a result, the kernel overhead at the beginning and at the end of the execution has a high weight. Overall, because the kernel time is not significant, only user data is presented in the next sections.

Experimental results show that all the processors in the system behave similarly. Consequently, for the numbers shown in this paper, we take the average across the 4 processors.

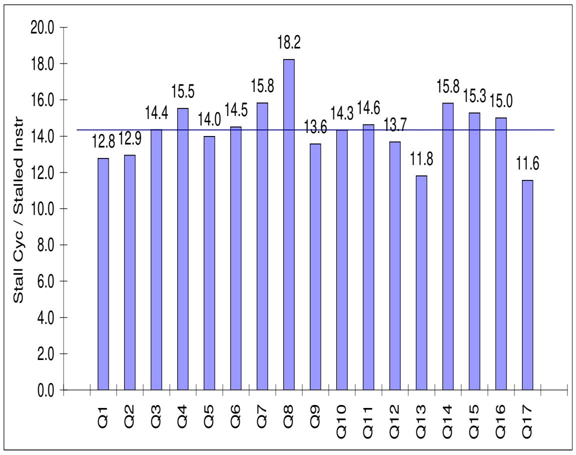
3.3 Cycles Per Instruction (CPI)

The instructions executed by the Pentium Pro processor, also known as macro-instructions, are internally translated into RISC micro-operations. The number of micro-operations per macro-instruction depends on the complexity of the macro-instruction. These micro-operations are the unit executed in the processor's pipeline. At every cycle, the processor can retire up to three micro-operations. For this reason, the Pentium Pro is said to be a three-issue superscalar processor. We will, however, present the CPI in terms of the number of cycles per macro-instruction.

Figure 3-(a) presents the CPI values for each query. In addition, the average CPI for all 17 queries is shown as an horizontal line. From Figure 3-(a), we see that the CPI ranges



(a)



(b)

Figure 3: CPI values for each query (a) and number of resource-related stall cycles per stalled instruction for each query (b).

from 0.6 (in Q12) to 1.6 (in Q2 and Q17). The average CPI is 1.27. This value is lower than the one found for TPC-C [6] (2.52) and is comparable to the values found for the SPEC95 technical benchmarks [2] (0.5-1.5). While the CPIs of TPC-D are relatively modest, it is still important to examine all the potential sources of processor stall cycles.

3.4 Resource Stalls

One type of stall cycles are resource-related stalls. These stalls occur when there are not enough register renaming resources, memory buffer entries, or available functional units for the execution to proceed. A branch misprediction recovery is also considered a resource stall by the Pentium Pro event counters.

Figure 3-(b) presents, for each query, the number of resource-related stall cycles per stalled instruction. The horizontal line represents the average value for all the queries. From the results collected, we observe that the number of stall cycles per stalled instruction ranges from 11.6 to 18.2. The average value across queries is 14.3 cycles. While these numbers may look large, if we compute the number of resource-related stalls per retired instruction, the result is only about 0.15. Furthermore, we will see in Section 3.7 that resource-related stalls are relatively minor contributors to the overall number of stall cycles.

3.5 Cache Misses

Another source of stall cycles is cache misses. In the system that we measure, processors have a sophisticated two-level cache hierarchy (Section 2.1). The local cache miss rates of the queries are shown in Table 2. The table shows the average values of the miss rates, as well as the range of values observed in different queries. Both read and write accesses are included. From the range of values, we note that different queries have different cache behavior. For example, the L1 instruction miss rate for Q13 (5.8%) is about 20 times higher than the one for Q6 (0.3%). An important observation from this table is the maximum L2 data miss rate of 35.2%. It corresponds to query Q6. This high data miss rate indicates that some TPC-D queries would benefit from larger L2 caches.

The differences in cache miss rates can often be explained from the different access patterns. For example, query Q6 has the highest L2 data miss rate (35.2%), while query Q1 has the lowest one (1.9%). Both queries are composed of a simple select operation on the *Lineitem* table. The execution plans show that both queries use the same index structure. What helps explain the differences in miss rates is that the selectivity of Q6 (1.4% of the table rows satisfy the select condition) is much higher than the one of Q1 (approximately 96.4% of the rows satisfy the condition). In other words, the locality of the data accesses in Q6 is worse than in Q1, which causes the higher L2 miss rate in Q6.

L1 Inst.	2.2%	(0.3 - 5.8)
L1 Data	2.6%	(1.4 - 5.1)
L2 Inst.	0.9%	(0.0 - 3.0)
L2 Data	12.2%	(1.9 - 35.2)

Table 2: Local cache miss rates of the queries. We show the average value and, in parenthesis, the range containing the minimum and the maximum values.

Finally, we note from Figures 3-(a) and 3-(b) that queries Q2, Q13, and Q17 have a high CPI value and a low number of resource-related stall cycles per stalled instruction. By examining the L1 and L2 miss rates for these queries, we find that they have higher values than the rest of the queries: their average L1 instruction miss rate is 5.2%, which is more than double the average across all the queries.

3.6 Branch Mispredictions

Another source of stall cycles is branch mispredictions. The Instruction Fetch Unit (IFU) of the Pentium Pro fetches a 32-byte line from the instruction cache every clock cycle. The instruction pointer may be computed from the output of the Branch Target Buffer (BTB), exception/interrupt status, and misprediction indications [5]. The branch predictor uses a two-level algorithm proposed by Yeh and Patt [16] on a 512-entry branch prediction buffer. To assess the accuracy of the branch prediction, Table 3 shows the branch misprediction rate, the BTB miss rate, and the branch instruction rate for the queries. The branch misprediction rate is the number of mispredicted branches over the total number of branches. The branch instruction rate is the number of retired branch instructions over the total number of retired instructions. The table shows both average values and the range of values.

Overall, the workload behaves relatively well. On average, only 4.3% of the branches are mispredicted. This is about one

Branch misprediction rate	4.3%	(1.7 - 8.2)
BTB miss rate	38.4%	(29.5 - 59.6)
Branch instruction rate	18.0%	(8.2 - 26.4)

Table 3: Values of the branch-related parameters.

third of the 14.3% misprediction rate of TPC-C [6]. The BTB miss rate is on average 38.4%.

We note that the queries with the highest CPIs also suffer the most from branch misprediction. Specifically, queries Q2, Q13, and Q17 have some of the highest branch misprediction rates (5.9%, 7.9%, and 4.5%, respectively) and BTB miss rates (51.4%, 59.6%, and 51.2%, respectively) of all the queries.

3.7 Relative Effect of Factors

We now compare the effect of the different factors that determine the stall time. These factors, which are compared in Figure 4 are: IFU stalls (*IFU*), resource-related stalls (*Res*), branch mispredictions (*Branch*), L2 data misses (*L2 D*), L2 instruction misses (*L2 I*), L1 data misses (*L1 D*), and L1 instruction misses (*L1 I*). Figure 4 shows their relative contribution to the stall cycles for each query. The last bar represents the average value for all the queries. Note that the *L1 D* and *L1 I* categories only include the effect of the round-trip latency to the L2 cache. If the access also misses in the L2 cache, the round trip latency from the L2 cache to the memory is added to *L2 D* or *L2 I*.

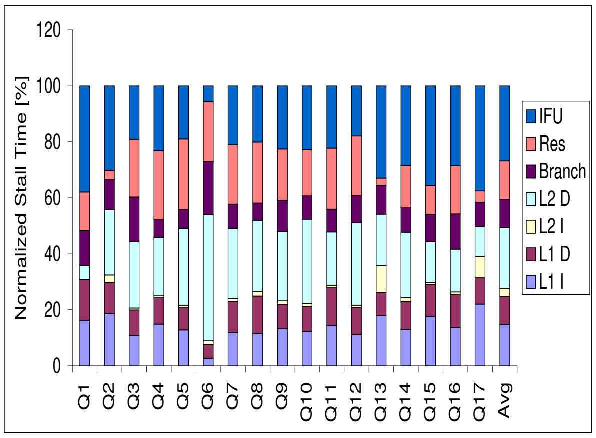


Figure 4: Relative effect of the stall factors.

For the figure, we calculated the weight of each component by multiplying the number of events by its average estimated cost. Specifically, a L1 cache miss that hits in the L2 cache has an average round-trip latency of 7 cycles. A L1 cache miss that goes all the way to memory has an average round-trip latency of about 120 cycles. The average branch misprediction penalty is 15 cycles. Finally, the IFU and resource-related stalls are obtained directly from the event counters. Of course, these sources of stall can be overlapped with each other and with the execution of instructions. Consequently, Figure 4 can only be used to assess the relative importance of the different stall-producing factors. It cannot be used to estimate the impact of these factors on the CPIs of the queries.

From Figure 4, we see that IFU stalls are the dominant factor, followed by L2 data misses, and L1 instruction misses. Let us consider the miss factors first. TPC-D would benefit from running on a machine with a larger L2 cache and a larger L1 instruction cache. If a split L2 cache were used, the data cache should be larger than the instruction one. This can be deduced from the small contribution of the stall from L2 instruction misses. In fact, a unified 512-Kbyte L2 cache effectively captures the instruction footprint.

IFU stalls have a relatively high weight in higher-CPI queries, namely Q2, Q13, and Q17. These queries, in addition, have high cache miss rates and branch misprediction rates. This suggests that their cache misses and branch mispredictions cause at least part of the IFU stalls. Finally, we see that the resource-related stalls have a relatively small weight.

3.8 Data Sharing

One issue that we would like to examine is the use of different cache line states in the machine. The multiprocessor uses a *MESI* protocol to maintain cache coherence [9]. A cache line can be in one of the following states: *Modified*, *Exclusive*, *Shared*, or *Invalid*. *Modified* is the state when the processor has modified the cache line and caches the only copy of the line, whereas *Exclusive* means that the cache line is still clean and that the processor is the only one that caches it. If there are two or more processors that cache the data in their caches, the state is *Shared*. The *Exclusive* state is used to enhance performance. Loading the line in state *Exclusive* has the advantage that, when the processor decides to write to it, the line becomes *Modified* without the need for a bus transaction [9].

Figures 5-(a) and 5-(b) show the state of a L2 cache line when it is accessed. Figure 5-(a) corresponds to data read operations while Figure 5-(b) corresponds to data writes. Each bar is broken down into the number of cache lines in each state: *Invalid* (I), *Shared* (S), *Exclusive* (E), and *Modified* (M).

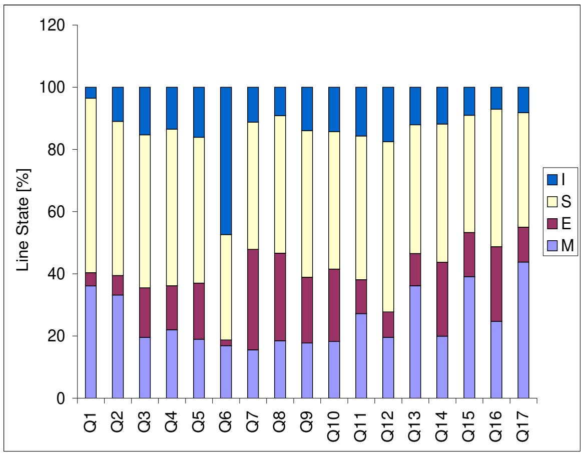
The first observation from these figures is that a significant fraction of the accesses target data in state *Modified*. Specifically, 25% of the reads and 85% of the writes find the data in state *Modified*. This means that, even though these TPC-D queries are read-only, highly-accessed data is modified. This data corresponds to metadata and to private buffers that store temporary results of the queries. The second observation is that it is rare for a processor to write to an *Exclusive* line. Since it is on writes that the *Exclusive* state is useful, these measurements suggest that little is gained with the *Exclusive* state for TPC-D. A similar conclusion was reached by Keeton *et al.* for TPC-C [6].

3.9 Bus Utilization

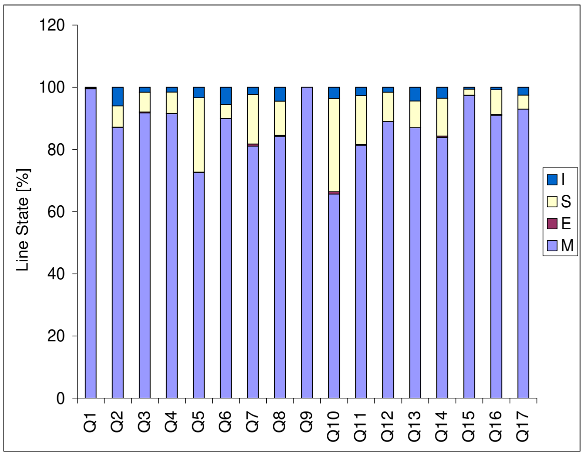
Another aspect of the system that we want to monitor is the load on the busses. The multiprocessor used in our experiments has two busses: the L2 cache bus and the system bus. The L2 cache bus, which connects the processor to the L2 cache, runs at 200 MHz and is 64-bit wide. The system bus connects the processor subsystem to the memory. The bus is split into address and data and is fully pipelined. It runs at 66 MHz and has a peak bandwidth of 533 Mbytes/s.

Figures 6-(a) and 6-(b) show the L2 cache bus and system bus utilization, respectively, for all the queries. Each bar in the figures corresponds to a different query. The horizontal line shows the average value for all the queries.

Focusing first on the L2 cache bus, we see that its utilization is low: its average utilization is only 13% and no query uses it more than 25%. It can be shown that the utilization is generally correlated with the combined L1 instruction and data miss rates. For example, the group of queries with the highest miss rates (Q2, Q13, and Q17), also have the highest L2 bus utilization. Obviously, the reason is that if the L1 caches intercept few requests, the L2 bus is more utilized.



(a)



(b)

Figure 5: State of a L2 cache line when the processor is about to issue a data read (a) or a data write (b) on it.

The system bus utilization is also low on average, as shown in Figure 6-(b). The utilization is 1-20%. This bus utilization is correlated with the combined L2 instruction and data miss rates.

Overall, we see that these busses are amply dimensioned for average conditions in the TPC-D queries. However, from sampling the query execution at different times, we often observe variations in the bus utilization of a factor of 10. Therefore, the extra bandwidth may be necessary to support the more demanding queries all the time.

3.10 Input & Output Activity

I/O is the traditional performance bottleneck in commercial applications. Commercial applications have large data sizes that need to be brought from disk to memory. In addition, as queries execute, they generate temporary data that needs to be stored in large tables. All this data may overflow memory and have to be moved back and forth between memory and disk. To reduce the amount of I/O, DBMS systems use two main techniques. One of them is index structures, which enable the application to access only the data pages that are really needed. The second technique is advanced buffer management algorithms to keep in memory the pages that are reused the most.

DBMS systems make frequent use of index structures. In addition, SQL Server uses what it calls *index covering*. Index covering occurs when an index is created on all of the attributes to which a query refers. In this case, since all the data that we need is stored in index structures, the data is retrieved directly from there instead of from data pages. Because the density of index pages is much higher than that of data pages, less physical I/O is required.

To assess the impact of some of these issues, Figure 7-(a) shows the size of the data read from or written to disk for each query. Furthermore, Figure 7-(b) shows, for each query, the ratio of the total size of the database tables accessed by the query over the total amount of I/O during the query execution from Figure 7-(a). Note that the bars in Figure 7-(b) are divided by 1000. Consequently, a value of 0.1 means that the combined size of the tables accessed by the query is 100 times larger than the amount of I/O recorded.

Figure 7-(a) shows that the execution of these queries generally involves little I/O. For example, queries Q1, Q6, Q8, Q13, Q14, Q15, and Q17 transfer less than 15 Kbytes of data. Other queries transfer more (1.4-202.7 Mbytes), but in most of the cases, the volumes transferred are modest. Furthermore, if we examine the ratio between the size of the database tables accessed and the total amount of I/O (Figure 7-(b)), we see that this ratio is usually very high. Although it is not clear from the figure, all queries have a ratio greater than 1. In addition, for 7 queries, this ratio ranges from 55,300 to 146,600.

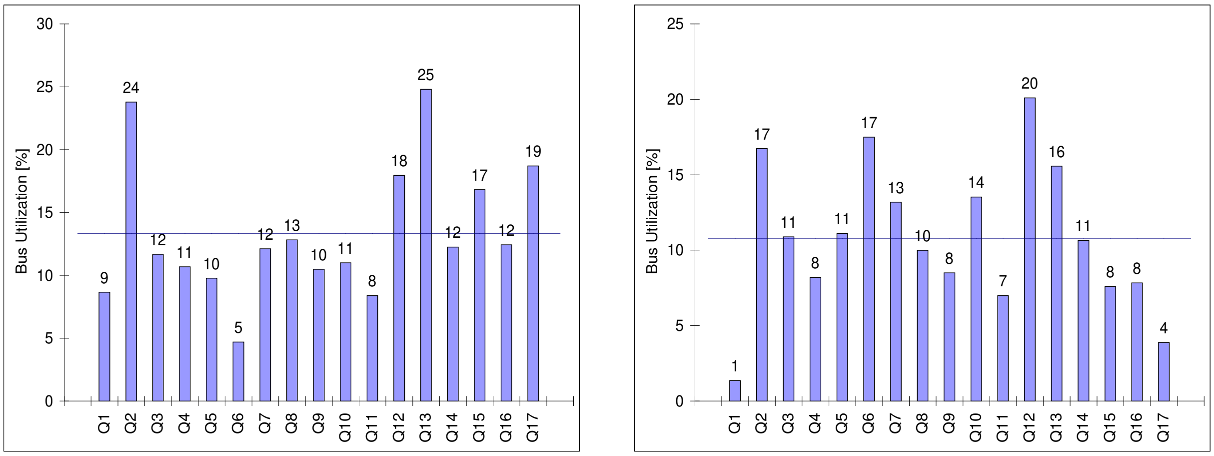
From these high numbers, therefore, we see that techniques to reduce the amount of I/O data transfer work. These techniques include indexing and smart buffer management. Recall that a measured query execution is always preceded by several executions of the same query. During the warm up runs, the buffer manager keeps useful pages. Finally, it can be shown that the size of the data read from disk is at least 5 times that of the data written to disk for all queries. This indicates that any overflow of the temporary tables created during query execution can not contribute much to the total amount of I/O recorded.

3.11 Comparing TPC-D to TPC-C

Before we conclude, we briefly compare our TPC-D results to those obtained by Keeton *et al.* [6] on TPC-C. Although we use SQL Server and they used Informix, the comparison is meaningful because the hardware platforms are very similar. Indeed, in both cases, the platform used is a 4-processor Pentium Pro server, and both machines have a similar configuration. Consequently, most of the differences in the measured data come from the different workload behavior.

The differences between the two workloads are briefly summarized in Table 4. On-line transaction processing workloads like TPC-C have short queries that access little data and involve both reading and writing. In addition, many users are issuing queries at the same time and, as a result, there is a high network activity. Decision support workloads like TPC-D, instead, have long, complicated queries that often access much data in read-only mode. At any point in time, only a few queries are in progress.

The TPC-D experiments are performed with one query at a time, while the TPC-C experiments involved multiple, concurrent user sessions. Consequently, we expect that the TPC-C experiments induce more context switches and, as a result, result in a higher operating system overhead. Indeed, our results show an average of 6% kernel time, while Keeton *et al.* report a 22% kernel time.



(a) (b)

Figure 6: Utilization of the L2 cache bus (a) and the system bus (b).

Characteristic	TPC-C	TPC-D
Problem domain	On-line transaction	Decision support
Rows accessed per query	Few	Many
Dominant access mode	Reads and writes	Reads
Concurrent users	Many	Few
Network activity	High	Low

Table 4: Comparing TPC-C and TPC-D.

The data cache behavior of TPC-D is better than the one of TPC-C. One major reason is that typical TPC-D operations like scans and joins often have good spatial locality. The short TPC-C queries, instead, often test or update a handful of locations with little spatial proximity. Furthermore, if we run multiple TPC-C queries concurrently on the same processor, the data cache can be trashed. Finally, the working sets of TPC-C are larger than those of TPC-D. Overall, we record average data miss rates of 3% and 12% for the L1 and L2 caches, respectively. Keeton *et al.*, instead, reported miss rates of 7% and 18%, respectively.

Instruction fetch is more effective in TPC-D than in TPC-C. TPC-C queries have many tests, which result in relatively more conditional branches. In addition, the code is not as loop intensive as in TPC-D. As a result, TPC-D has lower instruction miss rates and branch misprediction rates. Indeed, we record average instruction miss rates of 2% and 1% for the L1 and L2 caches, respectively, while Keeton *et al.* reported instruction miss rates of 6% and 4%, respectively. Similarly, we find an average branch misprediction rate of 4.3%, while Keeton *et al.* reported a branch misprediction rate of 14.3%.

Finally, the state of L2 cache lines when they are accessed by the processor is similar in both workloads. Indeed, if we consider read accesses, which are the significant ones in TPC-D, the state of the cache line (Figure 5-(a)) for the average query is similar to the state reported by Keeton *et al.*.

4 Conclusion

In this paper, we presented a detailed performance characterization of the TPC-D benchmark running on a 4-processor Pentium Pro SMP multiprocessor. Our results showed that all TPC-D queries have a relatively low CPI, which is comparable to values observed for popular technical applications. We also identified the major factors that cause stall cycles and, therefore, inhibit lower CPIs. These factors are the instruction fetch bottleneck and data misses in the secondary cache. Unfortunately, we cannot quantify their exact impact on the CPI because of the overlapping of operations that takes place in the processor and memory hierarchy. Our data also shows that kernel time is negligible, that branch prediction is effective and that, like in TPC-C, the exclusive state in the cache tags is largely unnecessary. Finally, compared to TPC-C, TPC-D queries tend to have less kernel time, lower cache miss rates, and lower branch misprediction rates.

Acknowledgments

We would like to thank Meena Kandaswamy, Bradley Seevers, Nitin Singhvi, and Padmashree Apparao at Intel, and Darrell Suggs and Kenneth Cantrell at Data General, for their help with the experiments. We also thank the referees and the graduate students in the I-ACOMA group, especially Jose Martinez, for their comments on this work.

References

[1] L.A. Barroso, K. Gharachorloo, and E. Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 3–14, June 1998.

[2] D. Bhandarkar and J. Ding. Performance Characterization of the Pentium Pro Processor. In *Proceedings of the Third International Symposium on High-Performance Computer Architecture*, pages 288–297, February 1997.

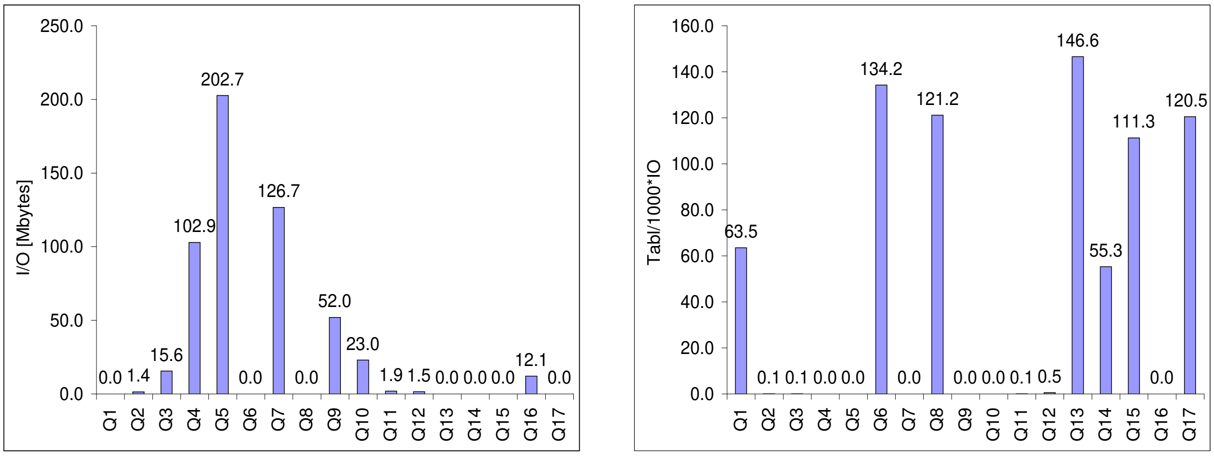
[3] Data General. http://www.dg.com/aviion. 1998.

[4] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc., second edition, 1994.

[5] Intel Corporation. *Pentium Pro Family Developer's Manual*, 1996.

[6] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker. Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-98)*, pages 15–26, June 1998.

[7] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, H. Levy, and S. Parekh. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-98)*, pages 39–51, June 1998.



(a) (b)

Figure 7: Total I/O during query execution (a) and ratio of the total size of the database tables accessed by the query over the total I/O during query execution (b).

[8] G. Papadopoulos. How I Learned to Stop Worrying and Love Shared Memory? *Keynote speech at the Third International Symposium on High-Performance Computer Architecture*, February 1997.

[9] M. S. Papamarcos and J. H. Patel. A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, pages 348–354, June 1984.

[10] P. Ranganathan, K. Gharachorloo, S. Adve, and L. Barroso. Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 307–318, October 1998.

[11] D. Solomon. *Inside Windows NT, Second Edition.* Microsoft Press, 1998.

[12] P. Stenstrom et al. Trends in Shared Memory Multiprocessing. *IEEE Computer*, pages 44–50, December 1997.

[13] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torrellas. The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors. In *Proceedings of the Third International Symposium on High-Performance Computer Architecture*, pages 250–260, February 1997.

[14] Transaction Processing Performance Council. *TPC Benchmark C Standard Specification Revision 3.4*, 1998.

[15] Transaction Processing Performance Council. *TPC Benchmark D Standard Specification Revision 1.3.1*, 1998.

[16] T.-Y. Yeh and Y. Patt. Two-Level Adaptive Training Branch Prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, pages 51–61, November 1991.