A Detailed Comparison of Two Transaction Processing Workloads

Robert Stets, Kourosh Gharachorloo, and Luiz André Barroso (1)

Western Research Laboratory
Hewlett-Packard Company
1501 Page Mill Road
Palo Alto, CA 94304

ABSTRACT

Commercial applications such as databases and Web servers constitute the most important market segment for high-performance servers. Among these applications, online transaction processing (OLTP) workloads provide a challenging set of requirements for system designs since they often exhibit inefficient executions dominated by a large memory stall component. A number of recent studies have characterized the behavior of transaction processing workloads and proposed architectural features to improve their performance. These studies have typically used a workload based on either the TPC-B or the TPC-C benchmark, with many of them opting for the simpler TPC-B benchmark. Given that the TPC-B and TPC-C workloads exhibit dramatically different characteristics on certain architectural metrics (such as cycles-per-instruction), it becomes important to find out whether the results or conclusions of these previous studies are heavily biased due to their choice of workload.

This paper presents a detailed comparison of the debit-credit (modeled after TPC-B) and order-entry (modeled after TPC-C) transaction processing workloads in the context of various architectural choices. Our experiments use the Oracle commercial database engine for running the workloads, with results generated using both full system simulations and actual runs on Alpha multiprocessors. Our results confirm that certain characteristics of these workloads, such as cycles-per-instruction (CPI) and dirty miss frequency, are indeed quite different. Nonetheless, it turns out that the overall impact of most architectural choices (e.g., out-of-order execution, on-chip integration of system modules, chip multiprocessing) are surprisingly similar for the two workloads. Furthermore, the above similarity between the two workloads is sometimes due to non-intuitive effects that would be difficult to predict without conducting the experiment with both workloads. The findings in this paper make it easier to compare results from studies that use one or the other workload. Overall, we observe that for a wide range of architectural decisions that we considered, using the simpler TPC-B workload leads to virtually the same conclusions as using the more complex TPC-C workload. Finally, we show that these same conclusions hold across two generations of the Oracle database engine.

(1) Barroso has moved to Google, Inc., 2400 Bayshore Parkway, Mountain View, CA, 94043.

1 Introduction

Commercial applications such as databases and Web servers constitute the largest and fastest-growing segment of the market for high-performance servers. While applications such as decision support (DSS) and Web index search have been shown to be relatively insensitive to memory system performance [2], a number of recent studies have underscored the radically different behavior of online transaction processing (OLTP) workloads [2, 4, 5, 7, 10, 13, 16]. In general, OLTP workloads lead to inefficient executions with a large memory stall component and present a more challenging set of requirements for processor and memory system design. This behavior arises from large instruction and data footprints and high communication miss rates that are characteristic for such workloads [2]. At the same time, the increasing popularity of electronic commerce on the Web further elevates the importance of achieving good performance on OLTP workloads.

Most of the previous studies on transaction processing are based on workloads modeled after either the TPC-B [20] or the TPC-C [21] benchmark. The TPC-B benchmark models a *debit-credit* workload in a banking environment, and consists of a single light-weight update-intensive transaction that operates on four types of database records with simple relationships among them. The TPC-C benchmark is substantially more complex and models an *order-entry* workload in a wholesale supplier environment. TPC-C involves a mix of five transaction types of different complexity that operate on a database of nine different record types with non-trivial relationships among them. Furthermore, while TPC-B primarily focuses on stressing the back-end database server, TPC-C simulates a complete computing environment with user and network interactions.

For the purpose of studying the performance of back-end database server architectures, there are two key issues that distinguish the TPC-B and TPC-C workloads. First, it is much easier to experiment with TPC-B. The more complex database for TPC-C, along

with multiple transaction types and the inclusion of user terminals and networking activity as part of the workload, make the TPC-C workload substantially more challenging to setup and to tune. Second, compared to the single update-intensive transaction in TPC-B, which stresses the memory and IO subsystems of the back-end server, the longer running TPC-C transactions lead to a higher processor utilization, with a slightly lower level of I/O and memory activity.

The simpler setup and tuning requirements for TPC-B (relative to TPC-C) have made it the more popular workload to model in a number of recent architectural studies [1, 2, 3, 8, 9, 14, 16]. Furthermore, the TPC-B workload has been successfully scaled down to make it amenable to simulation studies focusing on processor and memory system behavior [2]. Nonetheless, the TPC-B and TPC-C workloads have significantly different architectural behavior in some respects. For example, due to its somewhat worse memory system behavior, TPC-B exhibits a CPI (cycles-per-instruction) metric that is typically 1.5 to 2 times larger than that of TPC-C on most platforms. Given the above differences, it becomes important to find out whether the results or conclusions of the architectural studies based on one workload (e.g., TPC-B) would fundamentally change if we were to use the other workload (e.g., TPC-C).

This paper presents a detailed comparison of the debit-credit (modeled after TPC-B) and order-entry (modeled after TPC-C) transaction processing workloads. To the best of our knowledge, this is the first detailed side-by-side study of these two workloads across a wide range of architectural features. Our performance results for OLTP are based on executions of the Oracle commercial database engine (version 8.0.4) running under Compaq Tru64 Unix. As part of this study, we have successfully scaled down the TPC-C workload (using a similar methodology to that previously used for TPC-B [2]) to make it more amenable to experimentation. We use a combination of full system simulation (including operating system activity) and actual runs on Alpha multiprocessor platforms for our experiments.

We begin by studying the general processor and memory system behavior of the debit-credit and order-entry database workloads. We next study the workloads in the context of various architectural choices such as out-of-order processors, aggressive on-chip integration of system modules, and chip multiprocessing (CMP). Our results show that even though certain characteristics of the two workloads (such as CPI and dirty miss frequency) are dramatically different, the overall impact of most architectural choices are surprisingly similar. This similarity sometimes stems from somewhat non-intuitive effects. The performance impact of chip multiprocessing (as in Piranha [3]) on the two workloads is an illustrative case. Simple reasoning based on the basic characteristics of the two workloads would suggest that TPC-C should benefit less from CMP due to its higher processor utilization and lower memory system activity. However, our results show that CMP improves the performance of both workloads by virtually the same factor due to a combination of architectural effects. The above is a clear example where simple intuitive reasoning based on the basic characteristics of the two workloads can lead to erroneous conclusions, and stresses the importance of a comparison study like ours which is based on actual measurement.

Finally, we evaluate the impact of changes in the underlying database management system (DBMS) by repeating the above architectural experiments on two consecutive major releases of the Oracle server software. Yet again, we find that although significant improvement is observed in all levels of the memory hierarchy with the newer release, the important architectural trends remain fundamentally unchanged.

The comprehensive comparison presented in this paper allows architects interested in transaction processing to better understand the differences between TPC-B and TPC-C workloads. Furthermore, these findings make it easier to compare results from studies that use one or the other workload, and in some cases allow for an approximate extrapolation of results for one workload based on actual results for the other workload. Overall, we believe that studying either of these workloads is likely to provide architects and system designers with better insights for building more efficient server architectures for commercial applications. Furthermore, our results show that for a wide range of architectural decisions that we considered, using the simpler workload leads to virtually the same conclusions as using the more complex workload.

The rest of the paper is structured as follows. The next section provides a brief history of transaction processing benchmarks along with a description of our two workloads. Section 3 presents our experimental methodology, including the hardware platforms and simulation environment used in this study. We present a detailed characterization and measurement of the two workloads with different processor and memory system architectures in Section 4. Finally, we discuss related work and conclude.

2 Background on Transaction Processing Workloads

This section provides a description and comparison of our two transaction processing workloads. Transaction processing workloads are used in day-to-day business operations, and generally consist of a large number of online clients who continually issue short running

transactions, each accessing and/or updating a small fraction of the database data. The transaction processing server must be able to service a large number of transactions concurrently to meet the response time requirements of live operation, while guaranteeing the integrity of the database.

With the increasing use of computers for business applications, the first transaction processing benchmarks began to appear in the mid-eighties [19]. One of the first benchmarks used was TP1, a batch-mode benchmark that modeled a system serving ATM transactions. Next came the DebitCredit benchmark, which was a first attempt to measure not just server performance but also that of the network and clients. DebitCredit also introduced scaling rules that are in common use today. The need for better comparison among competing systems further drove the standardization of database benchmarks, culminating in the creation of the Transaction Processing Performance Council (TPC) in 1988. The first transaction processing benchmark by the TPC, TPC-A, was introduced in 1989. TPC-A extended DebitCredit by stipulating limits on response time, and adding rules to ensure measurement of sustainable performance, as well as tests for ACID (atomicity, consistency, isolation, and durability) requirements. TPC-B was introduced in 1990 as a simplified version of TPC-A, without the network or clients included as part of the workload. TPC-B was strongly advocated by vendors that did not offer complete OLTP solutions in their product portfolios, and therefore had a special interest in measuring server performance in isolation. In 1992, TPC-C was released. The main motivation for the development of TPC-C was to provide a richer set of transactions types (TPC-B has only one) and database relations that would more closely represent the complexities of real online transaction processing workloads.

The two OLTP workloads used in this study are modeled after TPC-B and TPC-C.[1] TPC-C is the currently approved OLTP benchmark from TPC, and is the most important benchmark used by vendors of server-class systems. Both TPC-A and TPC-B were declared obsolete as official benchmarks in 1995, having been superseded by TPC-C.

2.1 Oracle Database Engine

Our OLTP workloads run on top of the Oracle 8.0.4 DBMS. The Oracle DBMS runs on both uniprocessors and shared memory multiprocessor machines. Recent benchmark results demonstrate that the software scales well on current SMP systems. The server executes as a collection of Unix processes that share a common large shared memory segment, called the *System Global Area*, or SGA. Oracle has two types of processes, *daemons* and *servers*. The daemons run in the background and handle miscellaneous housekeeping tasks such as checking for failures and deadlocks, evicting dirty database blocks to disk, and writing redo logs. The server processes are the ones that actually execute database transactions and account for most of the processing. Both daemon and server processes share the same code, but have their own private data segment called the *Program Global Area*, or PGA. Daemons and servers primarily use the SGA for communication and synchronization, but also use signals to wake up blocked processes.

The SGA is roughly composed of two regions, namely the *block buffer* and *meta-data* areas. The block buffer area caches the most recently used database disk blocks in main memory, and typically accounts for 80% of the SGA size. The meta-data area contains the directory information that is used to access the block buffers, in addition to space for miscellaneous buffers and synchronization structures.

Database engines maintain two important types of persistent data structures: the database tables and the redo log. The latter keeps a compressed log of committed transactions, and is used to restore the state of the database in case of failure. Committing only the log to disk (instead of the actual data) allows for faster transaction commits and for a more efficient use of disk bandwidth. Oracle has a single daemon process, the *log writer,* that groups commit logs from independent transactions into a single disk write for more efficient use of the disk bandwidth.

We use Oracle in a dedicated mode for our OLTP workloads, whereby each client process has a dedicated server process for serving its transactions. The communication between a client and its server process occurs through a Unix pipe. To guarantee the durability of a committed transaction, commit requests block the corresponding server process until the redo information has been written to the log. To hide I/O latencies, including the latency of log writes, our OLTP runs are usually configured with multiple server processes (e.g. 7-8) per processor.

2.2 Debit-Credit Workload (TPC-B-like)

Our *Debit-Credit* (DB) workload is modeled after the TPC-B benchmark [20]. This benchmark models a banking system, with each transaction simulating a balance update to a randomly chosen account. The account balance update involves updates to four tables: the branch table, the teller table, the account table itself, and a history table. In terms of basic database operations, it consists of 3 row selections with data retrieval and update, and 1 row insertion. Each transac-

[1] Our OLTP workloads use the same SQL statements, table layouts, and database populations as indicated in the benchmark specifications. As our work is focused on the server, the workloads do not adhere to constraints outside the server (*e.g.*, number of client terminals, sufficient storage space for 60 days, *et cetera.*).

tion has fairly small computational needs, but the workload is I/O intensive because of the four table updates per transaction. Fortunately, even in a full-size TPC-B database (e.g., a few hundred GBs), the history, branch, and teller tables can be cached in physical memory, and frequent disk accesses are limited to the account table and the redo log.

2.3 Order-Entry Workload (TPC-C-like)

Our *Order-Entry* (OE) workload is modeled after the TPC-C benchmark [21]. TPC-C models the activities of a large wholesale supplier that operates from a number of warehouses, each serving ten sale districts, with each district serving 3,000 customers. Since we are only concerned with server performance, our workload is configured to run similarly to TPC-B in that only the activity in the server is evaluated.

TPC-C's database schema is significantly more complex than that of TPC-B. It consists of nine tables of various sizes and organization. Eight of the tables are scaled with the number of warehouses in the database. They are the Warehouse, Stock, District, Customer, Order, New-Order, Order-Line, and History tables. The Item table has a fixed size of 100,000 rows since there is a fixed number of stocked product types.

Instead of the single transaction of TPC-B, TPC-C has five transaction types: New-Order, Payment, Order-Status, Delivery, and Stock-Level. Clients submit a semi-random mix of such transactions to the server, such that 45% are New-Order transactions, 43% are Payment transactions, and the remaining are equally divided among the other transaction types. Since about 90% of the server activity is spent on New-Order and Payment transactions, we describe those in more detail.

The New-Order transaction is classified as "mid-weight" with respect to its computational requirements. Each transaction places orders for an average of 10 items (usually from the "home" warehouse with respect to the client terminal location) and submits orders for each item. The New-Order transaction touches all but one of the tables in the database. It performs an average of 10 row selections with data retrieval, 11 row selections with data retrieval and update, and 12 row insertions.

The Payment transaction is classified as "light-weight". The transaction updates the customer balance, as well as sales statistics on the district and warehouse levels. Each transaction performs 3 row selections with data retrieval and update, and 1 row insertion. In addition, 60% of the time the customer identification is done through the customer last name (a non-primary key), which on average requires another two row selections with data retrieval. This transaction is the closest in behavior to the single TPC-B transaction.

2.4 Workload Setup

The Debit-Credit and Order-Entry workloads have similar basic functionality, however experimentation with Order-Entry is much more challenging. First, for a given server performance level, Order-Entry requires a much larger database size in order to fully utilize the system. Second, the presence of five transaction types makes it necessary to run the workload longer in order to obtain a representative snapshot. Both these issues complicate experimentation, particularly when simulation is used. Finally, the higher complexity of Order-Entry, in terms of its database schema and transaction mixes, results in a much more difficult setup and tuning effort when compared to Debit-Credit. Careful tuning of this class of workloads is essential to any benchmarking or experimentation since a poorly configured OLTP workload is frequently I/O-bound or shows artificially high contention for locks, both of which can overshadow important architectural effects.

Both of our workloads were scaled down from the sizes that are mandated by the TPC scaling rules (which require a system with a higher performance to use a proportionally larger database size). Such scaling is critical to enable architectural experimentation, particularly for simulation, since the systems we analyze would have otherwise required database sizes of several hundreds of gigabytes. The scaling was done carefully in order to preserve the processor and memory system behavior of the full-size workloads. (The techniques for scaling are described in earlier work [2].) After scaling, the workloads were extensively tuned to achieve the best possible performance in the systems used for our experiments. The resulting workload sizes make the experiments run mostly out-of-memory. Our Debit-Credit workload uses 40 branches, with a total database size of 1.2GB. Our Order-Entry workload uses eight warehouses, with a 3.0GB database size. Both workloads use a 600MB SGA (the shared memory area is mostly used for caching database blocks), and are configured with 8 server processes per CPU.

Our monitoring experiments run 1500 transactions, while our simulations run 500 transactions. Our measurement interval was chosen to eliminate database startup effects such as loading of the database block cache.

3 Methodology

This section describes the hardware platforms and the simulation environment used in this study.

3.1 Hardware Platforms

Our monitoring experiments are performed on two different AlphaServer platforms. Our AlphaServer

8400 consists of eight 612MHz Alpha 21164 (in-order issue) processors. Each processor has 8KB direct-mapped on-chip instruction and write-through data caches, a 96KB on-chip combined 3-way set associative second-level cache (Scache), and a 4MB direct-mapped board-level cache (Bcache). The block size is 32 bytes at the first level caches and 64 bytes at lower levels of the cache hierarchy. The dependent load latencies measured on the AlphaServer 8400 are 11ns for a Scache hit, 55ns for a Bcache hit, 360ns for a miss to memory, and 542ns for a dirty miss (cache-to-cache transfer). The Alpha 21164 processor provides a rich set of event counters that can be used to construct a detailed view of processor behavior, including all activity within the three-level cache hierarchy [4]. We used these counters in a similar way to a previous study [2] to monitor the processor and memory system behavior of our workloads.

Our AlphaServer DS20 consists of two 500MHz Alpha 21264 (out-of-order issue) processors. Each processor has 64KB 2-way associative on-chip instruction and data caches, and a 4MB direct-mapped L2 cache. The dependent load latencies measured on the AlphaServer DS20 are 38ns for an L2 hit, 190ns for a miss to memory, and 350ns for a dirty miss (cache-to-cache transfer).

3.2 Simulation Environment

For our simulations, we use the SimOS-Alpha environment (our Alpha port of SimOS [17]), which was used in a previous study of commercial applications and has been validated against Alpha multiprocessor hardware [2]. SimOS-Alpha is a full system simulation environment that models the hardware components of Alpha-based multiprocessors (processors, MMU, caches, disks, console) in enough detail to run Alpha system software. Specifically, SimOS-Alpha models the micro-architecture of an Alpha processor [18] and runs essentially unmodified versions of Digital Unix 4.0 and PAL code.

The ability to simulate both user and system code under SimOS-Alpha is essential given the rich level of system interactions exhibited by commercial workloads. For the runs in this study, the kernel component is approximately 25% of the total execution time (user and kernel). In addition, setting up the workload under SimOS-Alpha is particularly simple since it uses the same disk partitions, databases, application binaries, and scripts that are used on our hardware platforms to tune the workload.

SimOS-Alpha supports multiple levels of simulation detail, enabling the user to choose the most appropriate trade-off between simulation detail and slowdown. Its fastest simulator uses an on-the-fly binary translation technique to position the workload into a steady state. For its medium-speed (in simulation time) processor module, SimOS-Alpha models a single-issue pipelined processor. Finally, the slowest speed processor module models a multiple issue out-of-order processor.

Processor Speed	1 GHz
Cache line size	64 Bytes
L1 data cache size (on chip)	64 KB
L1 data cache associativity	2-way
L1 instruction cache size (on chip)	64 KB
L1 instruction cache associativity	2-way
L2 cache size (off chip)	4 MB
L2 cache associativity	1-way
Multiprocessor Configuration	8 processors

Table 1: Base system parameters.

Table 1 presents the memory system configuration assumed for our base system with all system-level modules being external to the processor chip, except for first-level caches and second-level cache tags. Our multiprocessor configuration consists of 8 processor nodes with distributed memory and a directory-based cache coherence scheme. Our simulations model a sequentially consistent memory system. Table 2 presents the memory latencies for the various configurations we study. The configurations represent the second-level cache (L2), memory controllers (MC), and coherence controller and network router (CC/NR) being successively integrated with the processor. The table shows latencies for L2 cache hit, local memory, remote memory (2-hop), and dirty in a remote cache (3-hop).

Configuration	**L2 Hit**	**Local**	**Remote**	**Remote Dirty**
Base (4-way L2)	30	100	175	275
L2 integrated, DRAM	25	100	175	275
L2, MC integrated	15	75	225	275
L2, MC, CC, NR integrated	15	75	150	200

Table 2: Memory latencies (in cycles) for the chip-level integration experiments.

Parameter	Piranha (P8)	Next-Generation Microprocessor (OOO)
Processor Speed	500 MHz	1 GHz
Type	in-order	out-of-order
Issue Width	1	4
Instr. Window Size	-	64
Cache Line Size	64 bytes	64 bytes
L1 Cache Size	64 KB	64 KB
L1 Cache Associativity	2-way	2-way
L2 Cache Size	1 MB	1.5 MB
L2 Cache Associativity	8-way	6-way
L2 Hit/Fwd Latency	16 ns / 24 ns	12 ns / NA
Memory Latency	80 ns	80 ns

Table 3: Parameters for experiments on the impact of chip multiprocessing.

For measuring the impact of chip multiprocessing, we use a slightly different set of parameters. Table 3 shows the parameters we use for our chip multiprocessing experiments which compare Piranha [3] to an aggressive next-generation out-of-order processor.

4 Impact of Workload Choice on Architecture Trade-offs

This section presents results of actual hardware measurements and simulations for our two transaction processing workloads. We begin by presenting general execution time and cache behavior measurements. Next we consider the performance impact of various architectural choices: out-of-order processors, chip-level integration of system components, and chip multiprocessing. Finally, we evaluate the above results across two different versions of the database management software. Because we use different versions of hardware, Oracle DBMS, and operating system, the results in this section cannot be directly compared with those in previous papers [1,2,3,14] that use the Debit-Credit (TPC-B-like) workload.

4.1 General Execution Time Breakdown

We analyze execution time behavior using the hardware event counters on the 8-processor AlphaServer 8400 platform. Figure 1 presents the basic breakdown of execution time into four components: single- and double-issue cycles, and instruction- and data-related stall cycles. We also show the cycles per instructions (CPI) directly calculated from the event counts. Both workloads suffer a significant fraction of stall cycles with instruction stalls being almost as important as data stalls. The main difference is that Order-Entry (OE) exhibits 1.6 times better CPI than Debit-Credit (DC).[2] This difference is primarily due to more complex and compute-intensive queries in OE, and, as will be shown in the next section, the better locality of that workload.

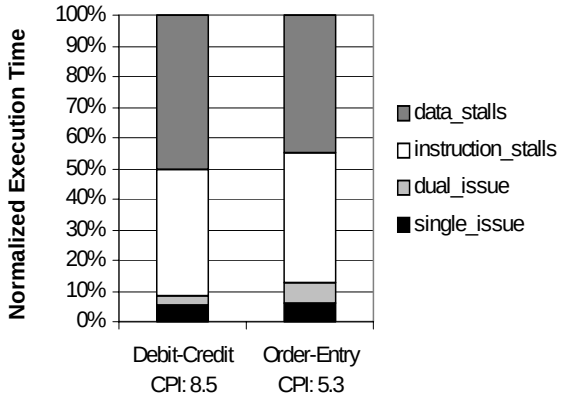

Figure 1. Basic cycle breakdown.

Using a wider set of event counts, combined with latency estimates, we calculate the approximate breakdown of cycles into the more detailed categories shown in Figure 2. The shaded segments show the stall cycles spent at each level of the memory hierarchy. We also show segments corresponding to single- and dual-issue cycles. The remaining segments are pipeline and address translation related stalls. Overall, the memory hierarchy performance of OE is better than DC. Nevertheless, both workloads are sensitive to latency at all levels of the memory hierarchy. In addition, the "memory barrier" component is significantly larger in DC; this is related to the higher frequency of synchronization operations due to the much simpler transactions in DC.

4.2 Cache Behavior

Table 4 presents the cache miss rates at all levels of the cache hierarchy, measured on the 8-processor AlphaServer 8400. The local miss ratio is the rate of accesses that miss in the cache relative to those that reach that cache. The fraction of dirty misses is relative to Bcache misses. The results show that both workloads overwhelm all the on-chip caches. While the Bcache exhibits some locality, its miss rate is still significant, with DC exhibiting a 1.6x higher miss rate. The other significant difference is the higher fraction of dirty misses in DC. Again, this is likely due to the much shorter transactions in DC, which increases the

[2] The higher CPI for Debit-Credit compared to a previous paper [2] is mostly due to a two times faster processor frequency.

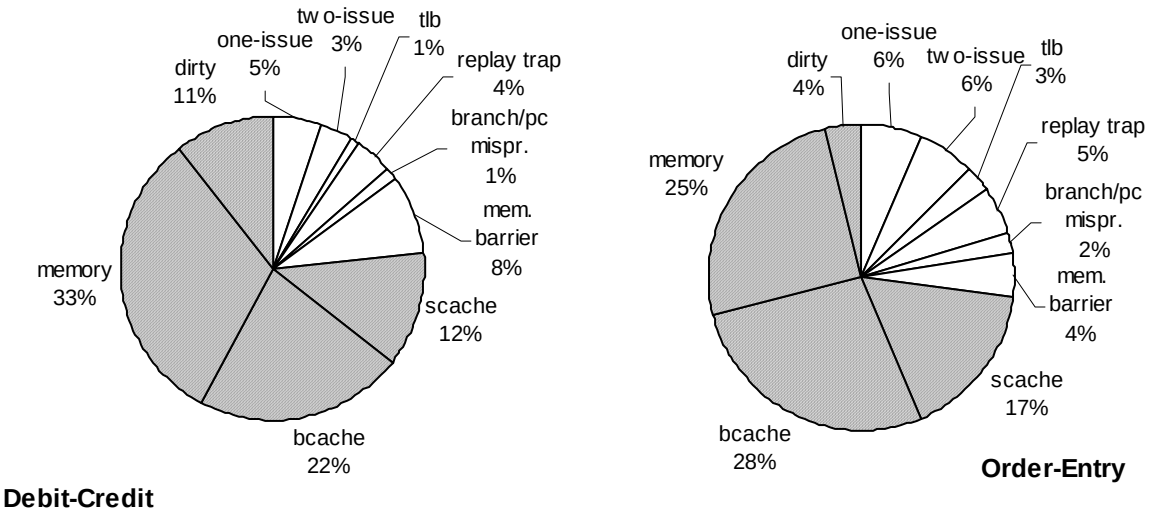

Figure 2. Detailed estimated cycle breakdown. Memory subsystem cycles are shown in the shaded pattern.

Parameter	Debit-Credit	Order-Entry
Icache (global)	13.6%	13.4%
Dcache (global)	22.2%	31.0%
Scache (global)	7.5%	7.5%
Bcache (global)	1.6%	1.0%
Scache (local)	32.3%	28.2%
Bcache (local)	21.4%	13.5%
Dirty miss fraction	17.8%	9.6%

Table 4: Cache miss rate in a 21164 system for Debit-Credit and Order-Entry workloads.

relative frequency of events that are associated with inter-process communication (such as commits). Finally, the on-chip cache behavior is similar.

Using full-system simulation, we study the impact of L2 cache sizes and associativity on the performance of the two workloads. Figure 3 shows the normalized execution time for various cache sizes with one- and four-way associativity, using an 8-processor configuration (base configuration in Table 2). All times are normalized to the execution of the 1MB direct-mapped cache for each workload. The figure also shows the breakdown of execution into busy, L2 hit, and L2 miss times (L2 miss times include both memory and dirty misses). Both workloads show similar gains from larger caches. However, DC shows greater benefit from associativity.

Our measurement results on the AlphaServer 8400 are biased by the fact that the Alpha 21164 processor has very small first-level caches (8KB/8KB), which makes hits in the Scache a significant factor in overall performance. For more modern processors, larger first-level caches would capture many of the hits in the Scache. We use simulation to study the effect of larger first-level caches. Figure 4 shows the miss rates for the two workloads as the function of the first-level cache size for an 8-processor configuration. We assume separate instruction and data caches that are 2-way set-associative. User and kernel misses are shown. At the user level, both workloads exhibit a higher number of instruction cache misses relative to data cache misses. Consistent with our hardware measurements, DC exhibits higher miss rates than OE for both instructions and data. OE has better locality, and its relative gain from larger first-level caches is higher as well. At 64KB cache sizes, which is typical in current microprocessors, both workloads still exhibit a significant number of first-level cache misses.

4.3 Out-of-Order Processors

The results presented so far have been based on in-order processors. This section presents simulation results using the multiple-issue out-of-order model described in Section 3.2, along with measurements on Alpha 21264-based multiprocessors. Our out-of-order simulation results are based on an aggressive four-wide issue processor with four integer units (our workloads have no floating-point instructions), two load-store units, and a window size of 64 instructions.

Figure 5 presents results comparing in-order vs. out-of-order processors. In this section we focus on the two left-most bars for each workload; the remaining bars will be discussed in the next section. The left-most bar represents a single-issue in-order (INO) processor, while the bar next to it represents the 4-

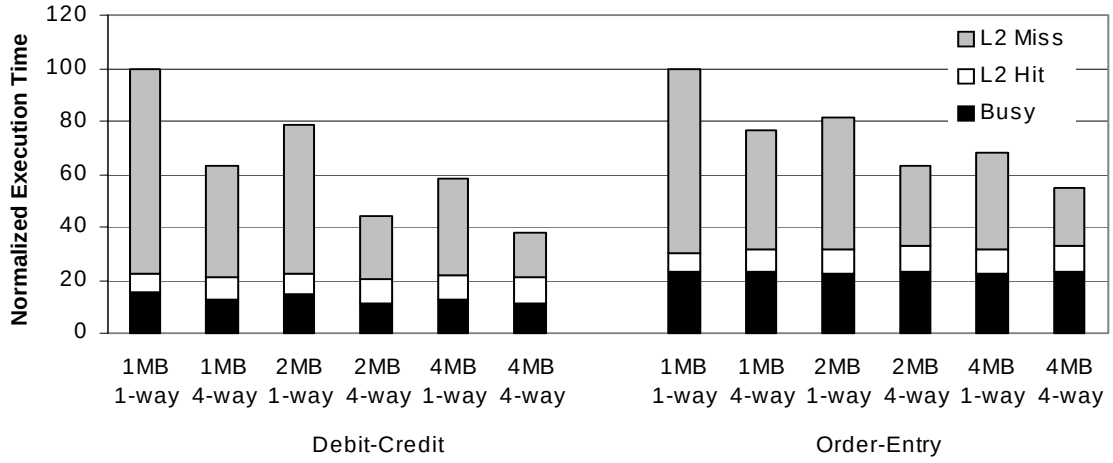

Figure 3. Effect of different L2 sizes and associativities.

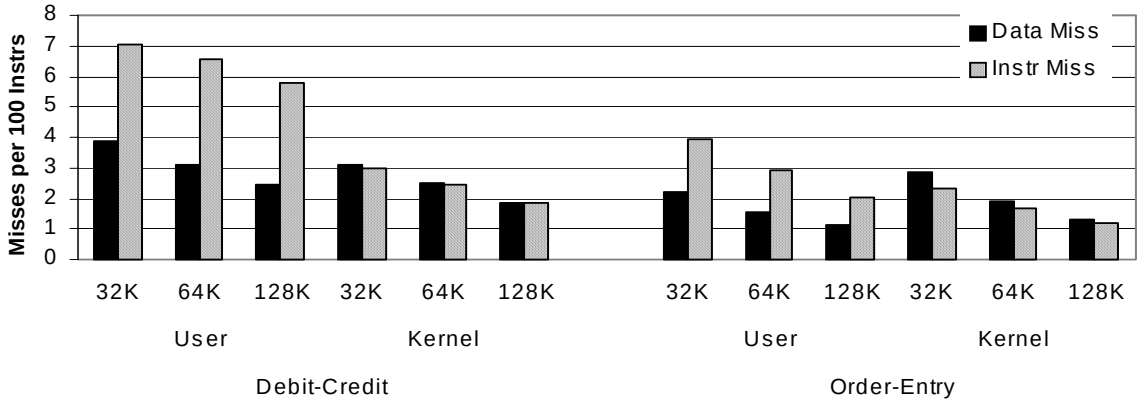

Figure 4. Miss rate for different L1 cache sizes.

issue out-of-order (OOO) processor described above (both use the base latencies in Table 2). The execution times are normalized to the base OOO configuration in each workload. As shown in the figure, the benefits of out-of-order processors are nearly identical for the two workloads (1.35x for DC and 1.47x for OE).

We also compared the performance of the two workloads using the AlphaServer 8400 (which uses the in-order Alpha 21164) and the AlphaServer DS20 (which uses the out-of-order Alpha 21264), with two processors active on each platform. The performance gains from using the AlphaServer DS20 system is again almost identical for both workloads (2.44x for DC, and 2.42x for OE). However these performance gains are only partly due to out-of-order effects, since the DS20 platform also has a more efficient memory subsystem, as discussed in Section 3.1.

4.4 On-Chip Integration of System Functionality

We next consider the impact of integrating various system modules onto the processor chip. Referring back to Figure 5, the right-most three bars show the impact of successively integrating the L2, the L2 and memory controllers (L2+MC), and finally also integrating the coherence controller and network router (All) onto the processor chip. These configurations were specified in Table 3. The trends for integrating just the L2 are similar, with DC showing slightly larger overall gains. Integrating the L2 and MC provides lower local latencies, but leads to a higher remote (two-hop) latency as shown in Table 2 (this assumption is justified in an earlier paper [1]). Both OE and DC are impacted by the increased two-hop miss latency and both experience only small benefits from the faster local latency. Both workloads exhibit slight

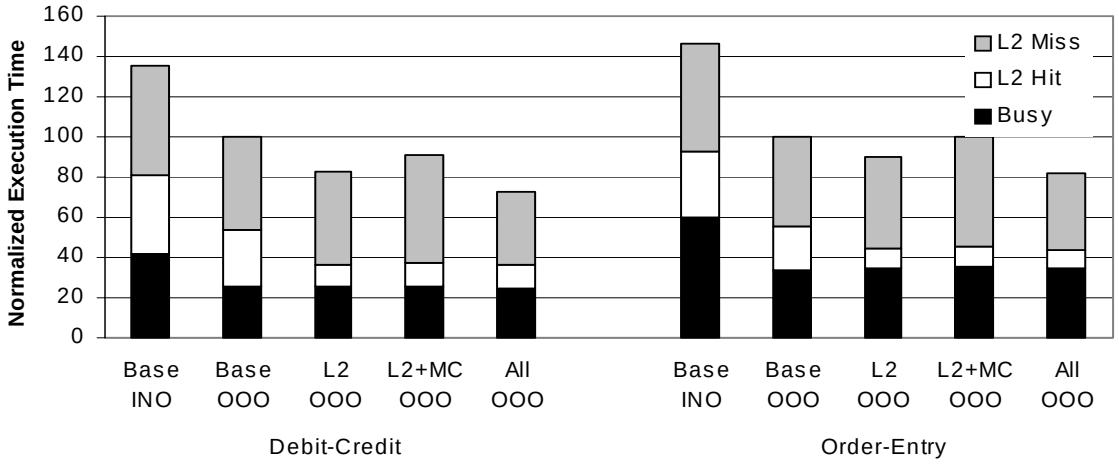

Figure 5. Performance impact of out-of-order processors and chip-level integration (8-processor configuration).

slowdowns for L2+MC vs. L2. However, the L2+MC configuration is only used to isolate performance effects, as it is not a viable design decision to break the tight coupling between the MC and CC [1]. The more interesting and aggressive design point is full chip-level integration (All OOO), and we see that this design point provides an improvement of 1.36x and 1.22x for DC and OE. DC benefits more from the lower latencies resulting from full integration, due to its higher fraction of time spent in the memory system.

4.5 Chip Multiprocessing

Figure 6 compares the performance of a single out-of-order integrated processor versus a chip multiprocessor with eight single-issue, in-order cores (parameters specified in Table 3). The parameters for the chip multiprocessor are based on the Piranha CMP [3], which was designed for an ASIC design flow. As a result, some parameters (*e.g.*, 500MHz clock speed) are conservative compared to the out-of-order processor, which assumes a full-custom design flow. Since these experiments use single-chip configurations, and correspondingly there is no chip-to-chip communication, the L2 miss component is significantly smaller compared to the results in the previous sections. Overall, the results for DC show a gain of 2.9x for the CMP. The OE results show a slightly higher gain of 3.3x. By exploiting thread-level parallelism, the CMP is able to effectively produce a relatively large number of concurrent memory requests. In contrast, the relatively low degree of instruction-level parallelism hampers the OOO processor's ability to exploit parallelism in the memory subsystem..

Given the results in the previous sections, which show that DC spends a larger fraction of time in the memory system, the gains from chip multiprocessing might be expected to be larger for DC than for OE. Surprisingly, we observe that DC and OE exhibit virtually the same gains from CMP. One of the reasons for this surprising result is the cache size assumptions for this experiment (1.5MB for OOO and 1MB for CMP, given CMP uses more area for the multiple, albeit smaller, processor cores). Given the worse locality of DC, the smaller cache in CMP hurts its performance relative to OOO more than in the case of OE. This effect offsets the higher gains from CMP that one would intuitively expect for DC, leading to approximately the same performance gains for both workloads

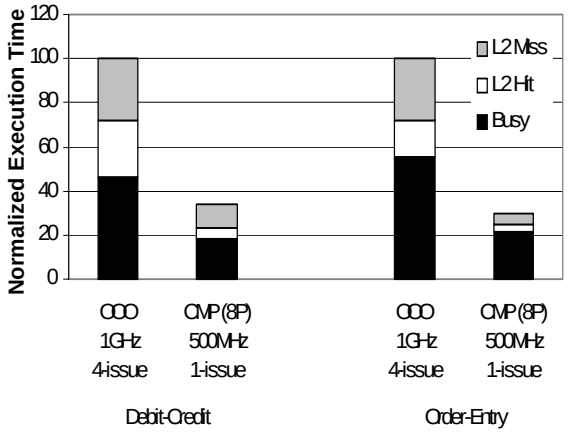

Figure 6. Chip multiprocessing performance.

4.6 Improvements in Database Management Software

In addition to understanding the characteristics of the two transaction processing workloads, it is important to realize that the evolution of the underlying database management code layer may also have a significant impact on architectural decisions. Newer major code releases from database vendors typically contain a mix of high-level algorithmic changes and lower-level performance optimizations, as well as new functionality, all of which can affect processor and memory system behavior. An awareness of the sensitivity of performance metrics to such effects is critical when extrapolating the results of earlier studies [1, 2, 3, 14] that used previous generations of DBMS software.

To evaluate such effects, we repeated the above Debit-Credit workload experiments using Oracle DBMS version 7.1.3 (Oracle7), a major code release preceding the Oracle version 8.0.4 (Oracle8) used above. Table 5summarizes the main cache hierarchy differences that we observed.

Overall, memory behavior in the higher levels of the cache hierarchy improves noticeably in Oracle8. The most dramatic improvement is seen in the 21164's 8KB Dcache, where the miss rate is halved with Oracle8. Our simulation results (not shown here) indicate that first-level cache miss rates improve by more than 20% across a range of cache sizes. Despite this reduction of misses in the higher levels of the cache hierarchy, there is a slight increase in the overall Bcache miss rate. On the positive side however, the fraction of dirty misses is also reduced by 27% in Oracle8 which can further improve scalability of the software. Considering L2 cache sizes and associativities, Oracle7 shows exactly the same trends as those depicted in Figure 3 for Oracle8. However, Oracle8 benefits more from higher associativities than Oracle7.Even though the memory system behavior is clearly different across the two versions of Oracle, we find that the architectural experiments presented in Sections 4.3-4.5 exhibit virtually identical trends across these two versions. In comparison to a single-issue in-order execution processor, a four-issue, out-of-order processor improved the performance of DC by factors of 1.32x and 1.36x on Oracle7 and Oracle8, respectively. The slight improvement in the latter can be attributed to Oracle8's lower primary cache miss rates. The only significant difference in the results of the chip-level integration experiments occurred for a chip with an integrated L2 and MC. As described, this design point leads to a reduced dirty miss latency, but an increased two-hop latency. Since the number of dirty misses decreases and the number of two-hop misses increases in Oracle8, DC performance showed less benefits from this chip configuration than with Oracle7. Finally, a CMP architecture improved performance over an integrated, out-of-order architecture by similar factors of 2.94x and 2.96x on Oracle7 and Oracle8, respectively. Again, the slight performance differences can be attributed to the better primary cache performance in Oracle8.

It is important to mention that a variety of other factors not considered in this paper can also affect the behavior of the workload. For instance, profile-based code layout optimizations have been shown to improve instruction cache performance of transaction processing workloads by up to 50%, and overall execution time by about 25% [15]. The potential of such techniques should be understood by architects before hardware solutions are evaluated.

In summary, all major architectural trends remained surprisingly consistent across the two workloads and across two generations of DBMS code, even though noticeable differences were observed in memory hierarchy parameters.

		Oracle7	**Oracle8**	**Improvement**
CPI		11.8	8.5	28.0%
Miss Rate	Icache (global)	18.4%	13.6%	26.1%
	Dcache (global)	42.1%	22.2%	47.3%
	Scache (global)	16.6%	7.5%	54.8%
	Bcache (global)	1.5%	1.6%	-6.7%
	Scache (local)	37.8%	32.3%	14.6%
	Bcache (local)	12.7%	21.4%	-68.5%
Dirty Miss Fraction		24.5%	17.8%	27.3%

Table 5: Improvements in Debit-Credit performance over one generation of DBMS software (measurements on a 21164 system).

5 Discussion and Related Work

Although many architectural studies still use scientific/engineering benchmark suites such as SPEC or SPLASH, several more recent studies have used transaction processing workloads. The majority of these studies use a simpler workload, modeled after TPC-B or earlier versions of Debit-Credit workloads [1, 2, 3, 4, 8, 13, 14, 16]. A few studies have started to use TPC-C or workloads modeled after it [5, 6, 7, 11, 12]. Overall, the studies using these two classes of workloads have led to similar conclusions: high frequency of "unpredictable" branches, large memory stall component for both instructions and data, and significant communication misses.

To the best of our knowledge, this work is first detailed side-by-side study of these two workloads across a wide range of architectural choices and with two generations of a commercial database engine. Maynard et al. [10] is the only study that we are aware of that analyzes both the Debit-Credit (TPC-A) and Order-Entry (TPC-C) workloads. Their results show nearly identical branch behavior and operating system activity for TPC-A and TPC-C, as well as identical trends for various cache parameters. They also show higher L2 miss rates for the simpler Debit-Credit compared to Order-Entry, a behavior we also observed. There are many differences between Maynard's study and our work. Maynard's study focuses on contrasting technical and commercial workloads in general, while our study provides a detailed comparison of two specific commercial workloads. Furthermore, our study uses a modern commercial database engine running on multiprocessors, while Maynard's study uses uniprocessor traces from an older generation engine. Finally, we use scaled down versions of these workloads, which allows us to use both actual hardware runs and full system simulations.

While we study these workloads in the context of a single commercial database engine, other papers that have used different commercial database engines indicate similar overall application behavior for transaction processing workloads [6, 7, 10, 16]. Our study shows clear differences in two generations of the same database engine. It would also certainly be interesting to study the effect of different database engines with respect to architectural evaluations of these workloads.

6 Concluding Remarks

With the growing dominance of commercial applications in the multiprocessor server market, a large number of recent studies have focused on characterizing and improving the performance of challenging transaction processing workloads. Many of these studies have opted to use TPC-B instead of the TPC-C workload due to TPC-B's simpler set up and tuning requirements. However, due to the Transaction Processing Performance Council's decision in the mid-90's to declare the TPC-B obsolete as a benchmark in favor of TPC-C, some have questioned the validity of the results of previous studies that are based on TPC-B. This paper sheds light on the above issue by providing a detailed comparison of these two workloads.

Our comparison study is based on full system simulations and actual runs of the debit-credit (modeled after TPC-B) and order-entry (modeled after TPC-C) workloads. We indeed observe that the two workloads exhibit significantly different behavior with respect to certain processor and cache performance metrics. For example, the debit-credit workload exhibits a CPI (cycles-per-instruction) of 8.5 on our AlphaServer 8400 compared to a CPI of 5.3 for the order-entry workload. Similarly, the communication behavior of the two workloads is somewhat different with the debit-credit workload exhibiting a dirty miss frequency of about 17.8% compared to 9.6% for order-entry. Surprisingly, these dramatic differences do not seem to surface as visibly as one would expect when we study the overall behavior of the two workloads across a range of architectural choices.

Our simulation results for the impact of out-of-order processors show an improvement of 1.35x (debit-credit) and 1.47x (order-entry) over an in-order processor for the two workloads. Similarly, the overall impact of aggressive chip-level integration is 1.36 times for the debit-credit workload, which is slightly higher than the 1.23 times for order-entry. Finally, the impact of chip multiprocessing is virtually identical for the two workloads, with a design such as Piranha providing a benefit of ~3x over an aggressive next-generation out-of-order processor. Some of these similarities, especially in the case of chip multiprocessing, arise due to subtle effects that would be difficult to predict without doing actual experiments with both workloads. Overall, these results illustrate the surprisingly similar performance behavior of the two workloads across a wide range of architectural choices. Furthermore, we observe that the above results remain fundamentally unchanged across different generations of the underlying database management software.

The reasoning behind the decision to declare TPC-B obsolete as a benchmark is not necessarily relevant to the merits of this workload for studying the processor and memory system performance of transaction processing servers. For example, one of the main reasons for this decision was that TPC-B, which was conceived as a database stress test, did not encompass the entire system (e.g., user terminals and network connections) and allowed server vendors to eschew the cost and performance issues of these other components in their benchmarks. Interestingly, these benchmarks

were developed in an era when storage and network behavior were the dominant performance factors for transaction processing, while today the most important factor is memory system behavior. Since it is widely acknowledged that actual customer database applications typically exhibit poorer memory system behavior compared to TPC-C, the more stressful TPC-B workload may actually play a positive role by pushing architects to design better performing systems for customer workloads. And as we have shown, these two workloads do not seem to lead to radically different trade-offs on general architectural choices.

Acknowledgments

We thank Ravishankar Mosur for reviewing this paper.

References

[1] L. A. Barroso, K. Gharachorloo, A. Nowatzyk, and B. Verghese. Impact of Chip-Level Integration on Performance of OLTP Workloads. *6th International Symposium on High-Performance Computer Architecture*, January 2000.

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

[3] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. *27th Annual International Symposium on Computer Architecture*, pages 282-293, June 2000.

[4] Z. Cvetanovic and D. Bhandarkar. Characterization of Alpha AXP performance using TP and SPEC workloads. *21st Annual International Symposium on Computer Architecture*, pages 60–70, April 1994.

[5] Z. Cvetanovic and D. Donaldson. AlphaServer 4100 Performance Characterization. Digital Technical Journal, 8(4), pages 3-20, 1996.

[6] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu. Evaluation of multithreaded uniprocessors for commercial application environments. *23rd Annual International Symposium on Computer Architecture*, pages 203–212, May 1996.

[7] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance Characterization of the Quad Pentium Pro SMP Using OLTP Workloads. *25th Annual International Symposium on Computer Architecture*, pages 15-26, June 1998.

[8] J. Lo, L. A. Barroso, S. Eggers, K. Gharachorloo, H. Levy, and S. Parekh. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. *25th Annual International Symposium on Computer Architecture*, June 1998.

[9] T. Lovett and R. Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. *23rd Annual International Symposium on Computer Architecture*, pages 308-317, May, 1996.

[10] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. *6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 145–156, October 1994.

[11] M. Martin, D. Sorin, A. Ailamaki, A. Alameldeen, R. Dickson, C. Mauer, K. Moore, M. Plakal, M. Hill, D. Wood. Timestamp Snooping: An Approach for Extending SMPs. *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 25-36, November 2000.

[12] A. Nanda, K. Mak, K. Sugavanam, R. Sahoo, V. Soundararajan, and T. Smith. MemorIES: A Programmable, Real-Time Hardware Emulation Tool for Multiprocessor Server Design. *9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 37-48, November 2000.

[13] S. E. Perl and R. L. Sites. Studies of windows NT performance using dynamic execution traces. *2nd Symposium on Operating System Design and Implementatio*n, pages 169–184, October 1996.

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

[15] A. Ramirez, L.A. Barroso, K. Gharachorloo, R. Cohn, J. Larriba-Pey, P.G. Lowney, and M. Valero. Code Layout Optimizations for Transaction Processing Workloads. *28th Annual International Symposium on Computer Architecture*, pages 155-166, June, 2001.

[16] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta. The impact of architectural trends on operating system performance. *15th ACM Symposium on Operating System Principles*, December 1995.

[17] M. Rosenblum, E. Bugnion, S. Herrod, and Scott Devine. Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, Vol 7, No. 1, pages 78-103, January 1997.

[18] R. L. Sites and R. T. Witek. Alpha Architecture Reference Manual (third edition). *Digital Press*, 1998.

[19] Transaction Processing Performance Council web site. http://www.tpc.org.

[20] Transaction Processing Performance Council. *TPC Benchmark B Standard Specification Revision 2.0*. June 1994.

[21] Transaction Processing Performance Council. *TPC Benchmark C Standard Specification Revision 3.5*. October 1999.