# Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads[1]

Kimberly Keeton[*], David A. Patterson[*], Yong Qiang He[+], Roger C. Raphael[+], and Walter E. Baker[#]

[*]Computer Science Division
University of California at Berkeley
387 Soda Hall #1776
Berkeley, CA 94720-1776
{kkeeton,patterson}@cs.berkeley.edu

[+]Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025
{johnq,rogerr}@informix.com

[#]Gradient Systems
Redwood City, CA
web@gradientsystems.com

*University of California at Berkeley*
*Computer Science Division*
*Technical Report UCB//CSD-98-1001*

*April 1998*

## Abstract

*Commercial applications are an important, yet often overlooked, workload with significantly different characteristics from technical workloads. The potential impact of these differences is that computers optimized for technical workloads may not provide good performance for commercial applications, and these applications may not fully exploit advances in processor design. To evaluate these issues, we use hardware counters to measure architectural features of a four-processor Pentium Pro-based server running a TPC-C-like workload on an Informix database. We examine the effectiveness of out-of-order execution, branch prediction, speculative execution, superscalar issue and retire, caching and multiprocessor scaling. We find that out-of-order execution, superscalar issue and retire, and branch prediction are not as effective for database workloads as they are for technical workloads, such as SPEC. We find that caches are effective at reducing processor traffic to memory; even larger caches would be helpful to satisfy more data requests. Multiprocessor scaling of this workload is good, but even modest memory system utilization degrades application memory latency, limiting database throughput.*

1. This research has been supported by DARPA (DABT63-C-0056), the California State Micro program, and by donations and research grants from Informix, Intel, Hitachi, LG Semiconductor, Microsoft, Silicon Graphics/Cray Research, and Sun Microsystems. Kim Keeton is supported by a Lucent Technologies doctoral fellowship.

A shorter version of this paper appeared in *Proceedings of the 25th International Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

# Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads

## Abstract

*Commercial applications are an important, yet often overlooked, workload with significantly different characteristics from technical workloads. The potential impact of these differences is that computers optimized for technical workloads may not provide good performance for commercial applications, and these applications may not fully exploit advances in processor design. To evaluate these issues, we use hardware counters to measure architectural features of a four-processor Pentium Pro-based server running a TPC-C-like workload on an Informix database. We examine the effectiveness of out-of-order execution, branch prediction, speculative execution, superscalar issue and retire, caching and multiprocessor scaling. We find that out-of-order execution, superscalar issue and retire, and branch prediction are not as effective for database workloads as they are for technical workloads, such as SPEC. We find that caches are effective at reducing processor traffic to memory; even larger caches would be helpful to satisfy more data requests. Multiprocessor scaling of this workload is good, but even modest memory system utilization degrades application memory latency, limiting database throughput.*

## 1 Introduction

Commercial applications are an important class of applications with a large installed base. According to Dataquest, commercial applications, such as online transaction processing (OLTP) and decision support (DSS) database service, file service, media and email service, print service, and custom applications, were the dominant applications run on server machines in 1995 and are projected to be the dominant server applications in 2000 [26]. Commercial applications comprised about 85% of the 1995 server market, and are projected continue this dominance as the server market grows 15 percent annually.

Database workloads alone motivate the sale of vast quantities of symmetric multiprocessing machines, and hold the dominant fraction of the massively parallel computing market [19]: databases motivated 32% of the server volume in 1995, and will motivate 39% of the 2000 server volume [26]. Despite the widespread usage of commercial applications, they are often ignored in preference to technical benchmarks, such as SPEC or LINPACK, in computer architecture performance studies. This bias is due largely to the lack of available representative multi-user traces of commercial applications, the proprietary nature of database performance information and source code, and the difficulty of properly configuring a system to run typical database benchmarks.

Commercial and technical applications have significantly different execution characteristics [16]. Commercial applications generally have a large number (e.g., 100s to 1000s) of concurrent users. As a result, they typically have high context switch rates and multiprogramming levels. They spend a substantial portion of their execution in the operating system. Commercial applications perform many I/O operations, in a random access pattern, with data spread over a wide portion of a disk. Commercial applications perform data manipulation on strings or integers, in comparison with the extensive floating point activity in technical workloads. Unlike the small instruction working sets and tight loops of technical applications, commercial applications execute fewer loop instructions, and often use non-looping branch instructions. Because of their branching behavior and data access patterns, commercial applications have been less able to effectively use the memory system of traditional workstation and server architectures.

The potential implication of these differences is profound: computers optimized for technical workloads may not provide good performance for commercial applications, and these applications may not exploit advances in processors at the same rate as SPEC. This problem is exacerbated by the fact that I/O and memory system performance improvement rates lag far behind processor performance improvements. As a result, it is important for computer architects to consider a wide range of applications when designing and evaluating architectures, especially those intended to be used in SMPs.

In this paper, we use hardware counters to measure architectural features of a four-processor Pentium Pro-based server running a commercial database executing a TPC-C-like workload. We vary several hardware and firmware configuration parameters, such as L2 cache size, the number of processors and the number of outstanding bus transactions, to evaluate hardware design trade-offs. We examine the efficiency of caching, out-of-order execution, branch prediction, speculative execution, superscalar issue and retire and multiprocessor scaling.

We find that overall (e.g., database and operating system) CPI is roughly five times higher than the theoretical minimum CPI for the architecture, and much higher than the CPI of SPEC. Resource and instruction-related stalls comprise the majority of these cycles. While out-of-order execution is somewhat effective at hiding memory hierarchy latency and other stalls, it is less effective for OLTP

database workloads than for SPEC. The branch prediction algorithms and hardware support do not work nearly as well for this database workload. Finally, increasing superscalar issue and retire width will likely be only marginally helpful for this workload.

Not surprisingly, we find that caches are effective at reducing the processor traffic to memory. We observe improvements in L2 cache miss rates for L2 cache sizes up to 1 MB, the largest available for this processor. While larger caches are effective, this benefit is not without consequences. Coherence traffic, in the form of cache misses to dirty data in other processors' caches, increases as caches get bigger, and as the number of processors increases. We find that the exclusive state of the four-state MESI cache coherence protocol is under-utilized for multiprocessor configurations, and could likely be omitted in favor of a simpler three-state protocol. Finally, multiprocessor scaling of this workload is good, but even modest memory system utilization degrades application memory latency, limiting database throughput.

Several recent studies began the examination of the architectural impacts of transaction processing database workloads for symmetric multiprocessors. Most of the studies have focused on some variation of the now-defunct DebitCredit benchmark, also known in various incarnations as TP1, TPC-A, and TPC-B [1] [5] [16] [22] [25] [27]. (This benchmark has been withdrawn by the Transaction Processing Council.) A few have examined the more complex TPC-C order-entry workload [6] [16]. Only two of these papers have explored the effectiveness of out-of-order execution for the TPC-B workload [1] [25]. None examine the effectiveness of out-of-order processors for TPC-C. In general, we observe that these papers corroborate our findings, with a few exceptions. Because the scope of the studies is vast, we defer the discussion of similarities between our work and these related studies until presenting our results in Section 3 through Section 6.

This paper is organized as follows. Section 2 describes our experimental setup, including the configuration of our server, the architecture of the Pentium Pro processor, and a discussion of the TPC-C workload. It also presents our methodology for collecting and analyzing counter data. We discuss the decomposition of CPI in Section 3, and then further explore its memory hierarchy component in Section 4 and its processor component in Section 5. Multiprocessor scaling is explored in Section 6. Future research directions are presented in Section 7, and conclusions and our recommendations to computer architects are stated in Section 8.

# 2 Experimental Setup

This section describes our experimental infrastructure, including the configuration of our server and the architecture of the Pentium Pro processor. We describe the workload used in this study, which is based on the TPC-C benchmark, and discuss our methodology for collecting and analyzing hardware counter data.

## 2.1 Measurement vs. Simulation

Hardware measurement studies are typically limited to reporting performance for today's machines. To investigate future architectural alternatives, researchers generally employ simulation techniques. Both approaches have advantages and disadvantages. Direct measurement of real hardware means that software runs at full speed, which implies that it is possible to run a fully configured OLTP database application. Real hardware also means that there is no question of validation for the hardware model, only validation of the instrumentation. Unfortunately, measurement usually implies that architectural parameters, such as the number of functional units, reorder buffer entries, and cache sizes, are fixed. In addition, performance counters may not be accurate, or may not provide the desired information.

In contrast, simulation allows researchers to explore designs of the future. There is no limitation to what can be measured, offering arbitrary levels of detail. The difficulties with this approach are that validation of complex simulators is quite difficult and simulations can run up to 10,000 times slower than real-time, making it difficult to simulate large-scale fully configured systems.

In this study, we show that there is some middle ground between these two approaches. We measure a real machine, but vary hardware parameters, including second-level cache size, the number of outstanding bus transactions, and the number of processors, to explore architectural trade-offs. The flexibility in configuration parameters afforded by our system allows us to overcome some of the traditional limitations of measurement approaches.

## 2.2 SMP Hardware Architecture

Table 1 shows the system measured in this study, a four-processor Intel-based SMP, which uses 200 MHz Pentium Pro processors. We present detailed measurements for this base system, and then modify various architectural parameters, such as L2 cache size, memory bandwidth, and the number of processors, to study their effects on performance. We choose the four-processor SMP as the base case since it is a building block for small- to mid-range database servers.

Our base system consists of the quad Pentium Pro SMP, populated with 4 GB of 4-way interleaved 60 ns main memory. In NT 4.0, a process is limited to a 2 GB of user space. Thus, only 2 GB of this memory is accessible to the database server. In addition to the on-chip first-level caches, each processor has a 1 MB L2 cache in the same multichip module. The system is configured with 90 Quantum 4.55 GB Ultra SCSI-3 disks that store the TPC-C dataset. An additional 21 disks are used for performance monitoring, development, and scratch space. The 90 data

| Characteristic | Base System |
|---|---|
| Processor speed | 200 MHz |
| Number of processors | 4 |
| L1 caches | 8 KB instr., 8 KB data |
| L2 cache | 1 MB (unified) |
| System chipset | 82450 KX/GX (Orion) |
| System bus speed | 66 MHz |
| Memory size | 4 GB |
| Memory organization | 4-way interleaved |
| Memory speed | 14-1-1-1 (bus cycles) |
| lmbench read bandwidth | 213 MB/s |
| lmbench read latency | 190 ns |

**TABLE 1. Summary of base system configuration.** The memory read bandwidth and the average memory read latency are given by uniprocessor microbenchmarks that are part of the lmbench suite [17].

disks are connected via three Adaptec dual channel SCSI-3 controllers.

### 2.3 Overview of Pentium Pro Processor Architecture

Figure 1 shows the architecture of Intel's Pentium Pro processor. The Pentium Pro implements dynamic execution using an out-of-order, speculative execution engine, which employs register renaming, non-blocking caches and multiprocessor bus support. Intel IA-32 instructions

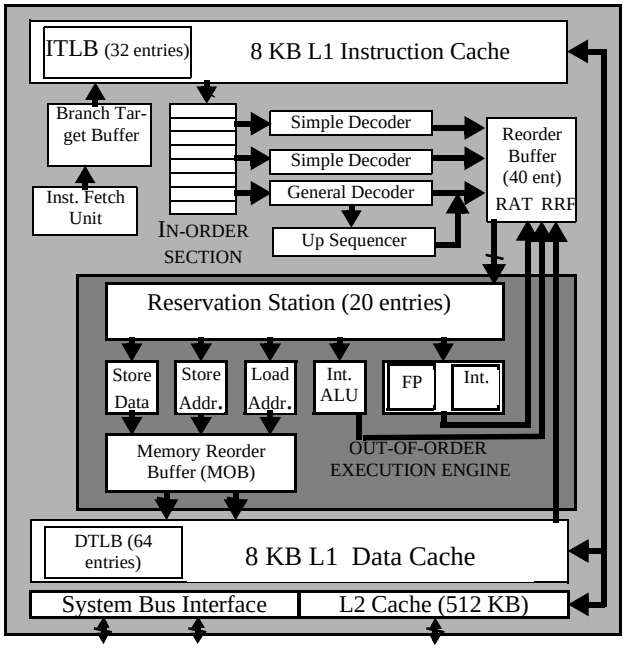


**FIGURE 1. Block diagram of Pentium Pro processor architecture.** The out-of-order execution engine is shown as the dark rectangle; the remainder of the processor is considered the in-order section.

(i.e., macro-instructions) begin and end execution in program order, in the "IN-ORDER SECTION" of Figure 1. They are then translated into a sequence of simpler RISC-like micro-operations (i.e., μops). μops are register renamed and placed into the Reservation Station, an out-of-order speculative pool of pending operations. Once their data arguments and the necessary computational resources are available, these μops are issued for execution in the "OUT-OF-ORDER EXECUTION ENGINE." After execution has completed, an instruction's μops are held in the Reorder Buffer until they can be retired, which may occur only after all previous instructions have been retired, and all of the constituent μops have completed. The Pentium Pro retires up to three μops per clock cycle, yielding a theoretical minimum cycles per μop (μCPI) of 0.33.

A more detailed description of the Pentium Pro's architectural features can be found in [2] [3] [9] [12] [20]. We will also present additional details in subsequent sections, when discussing our measurement results.

### 2.4 Software Architecture

We measured a tuned prototype version of Informix Online Dynamic Server [11], running on Windows NT 4.0 with service pack 3. Since this work began, there have been new releases of these software products, with improved performance. For this reason and due to the TPC-C reporting rules, we do not present absolute performance numbers.

The database server uses multiple processes, which exploit processor affinity to ensure that each process is run exclusively on its assigned CPU. User-level threads are then multiplexed on top of these processes. Disk I/O is done to raw disk partitions, not through the file system. I/O is done only in the kernel.

We measured a variant of the Transaction Processing Council's TPC-C benchmark [8]. TPC-C is an online transaction processing (OLTP) benchmark that simulates an order-entry environment, and includes the activities of entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

TPC-C is currently the only active OLTP benchmark supported by the TPC. It uses a mix of five transactions, rather than a single debit-credit transaction, like the now-defunct TPC-A and TPC-B benchmarks. TPC-C employs a more complex database structure and utilizes nonuniform data access patterns to simulate data hot spots, resulting in higher levels of contention for data access and update.

We used a modified version of TPC-C, where two client machines simulate thousands of remote terminal emulators (RTEs), generating requests with no think time between requests. The resulting load presented to the database server strongly resembles the full configuration with RTEs, and thus stresses the server in similar ways [28].

The performance metric for TPC-C is the number of NewOrder transactions per minute (tpmC). Since we have modified the TPC-C benchmark, and since our benchmarks

have not been audited, as per TPC-C rules, we cannot directly report tpmC ratings. Instead, we report throughput relative to the base system described above.

The TPC-C specification places several constraints on the relationship between transaction throughput and the database size. We ensure that our configurations fall within the prescribed range of 9 to 12.7 tpmC per warehouse, as required by the TPC. Configurations falling outside this range may exhibit different behavior, such as different disk I/O rates and different user-OS breakdown, which may affect the code paths and architectural behaviors measured [13] [15]. By keeping our configurations within the prescribed range, we keep the load offered to the system as consistent as possible across different configurations.

### 2.5 Methodology

We used the Pentium Pro hardware counters to measure processor-centric events. In addition to the base system described in Section 2.1, we measured numerous other system configurations obtained by varying three architectural parameters: L2 cache size, number of processors, and the number of outstanding bus transactions.[1] The values for these parameters are shown in Table 2. L2 cache size was modified by physically swapping processor/cache boards to switch between the three cache sizes. The number of processors was varied at boot time, so that the system was restricted to having only one, two, or four processors active. The buffer memory available to the database was held constant at 512 MB per processor. Finally, the number of outstanding bus transactions was varied by changing a BIOS parameter to limit the "I/O queue depth" of the controller [12].

| Parameter | Values |
|---|---|
| L2 cache size | 256 KB, 512 KB, 1 MB |
| Number of processors | 1, 2, 4 |
| Outstanding bus transactions | Uniprocessor: 1, up to 4 |

**TABLE 2. Summary of architectural parameters varied.**

To collect data on processor behavior, we used the Pentium Pro hardware counters. Each processor has two counters that can measure the number of a variety of events, such as instructions and µops retired, branch behavior, L1 and L2 cache misses, various bus transactions, and several types of stalls, for either user-level activity or system-level activity [12]. We used a total of 82 event types for the data presented in this paper. For each hardware configuration, we did at least five database runs. Each run consisted of a 15-minute warm-up period, which was sufficient to bring the database to steady state, and a 40-minute measurement period. (40 minutes is chosen to maximize the measurement time before a checkpoint must be taken.) The 40-minute measurement period is broken into 5-second fixed duration intervals, during which an event is measured for both user and system level. The same event pair is simultaneously measured across all four processors. Typically, each database run results in six to eleven observations per event type. To obtain enough data points to draw statistical conclusions, we perform at least five database runs for each hardware configuration. Some runs measured all events, and others focused on a subset of important events. We cycle through the counters in a different order for each run, to greater increase their coverage.

For each event and processor, we computed a *trimmed mean*; that is, we removed the minimum and maximum observations, and then computed the mean from the remaining observations [23]. After trimming, we have at least 30 (in some cases 40) observations for each event type. We then examined the data to determine the amount of noise due to measurement error. The standard deviation for a given event for a given processor was less than 10% of the mean for that event/processor combination, for nearly all of the event types.

Unless otherwise noted, we will present the average values across all active processors in the system, since in most cases the processors exhibit similar behavior. Any deviations from this norm will be noted.

## 3 Experimental Results: CPI

We begin by presenting the CPI for the TPC-C-like workload on the four-processor base system in Section 3.1. We then examine components of the CPI, such as memory system behavior, processor characteristics, and multiprocessor scaling more closely in Section 4, Section 5 and Section 6, respectively. In each section we pose and answer a set of questions exploring the relevant issues.

### 3.1 How does database CPI compare with the theoretical CPI possible on the Pentium Pro?

Using the Pentium Pro events that count the number of cycles and the number of instructions retired during the measurement period, we computed the cycles per micro-operation, µCPI, and the cycles per macro-instruction, CPI, for the database, the operating system, and the overall system. We begin by discussing the µCPI breakdown, shown in Figure 2, and continue by presenting the CPI breakdown, shown in Figure 3. Section 11.1 presents more detailed numerical data for these figures.

The overall measured µCPI is 1.55. Our system spends between 76% and 80% of its execution time in database code, which has a measured µCPI of 1.37. The remaining time is spent in the operating system at a measured µCPI of

1. In an earlier version of this paper, we describe the results of varying a fourth parameter, memory bandwidth. We refer the interested reader to [14] for more details.

2.87, over two times the database μCPI. The system is idle for less than 1% of the time. These values are 4X to 8.5X greater than the 0.33 theoretical minimum μCPI for the Pentium Pro. To understand this discrepancy, we further decompose the μCPI into computation and stall cycles, as shown in Figure 2.

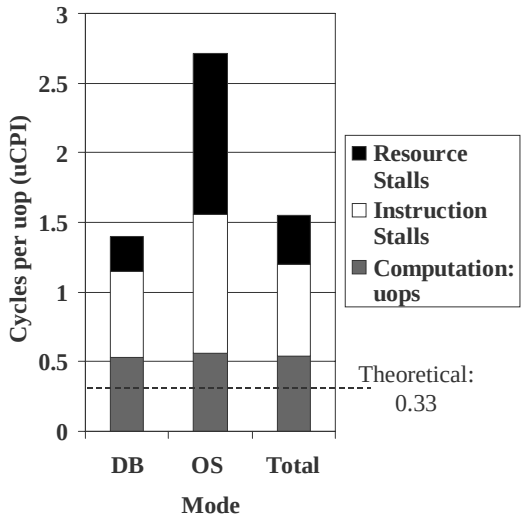


**FIGURE 2. Breakdown of cycles per micro-operation (μCPI) for base system.**

Computation μCPI is calculated based on the μop retire profile presented in Section 5.1: we assume μops retired in triple-retire cycles require 0.33 cycles in the steady state, double-retire cycle μops take 0.5 cycles, and single-retire μops need one cycle. This determines the number of cycles per μop. For example, if 53% of the μops are retired in triple-retire cycles, 22% in double-retire cycles, and 25% in single-retire cycles, the computation μCPI is 0.53*0.33 + 0.22*0.5 + 0.25*1 = 0.53 cycles per μop.

The Pentium Pro provides event types to monitor resource stalls and instruction-related stalls. Resource stalls account for cycles in which the decoder gets ahead of execution. For example, resource stalls encompass the conditions where register renaming buffer entries, reorder buffer entries, memory buffer entries, or execution units are full. In addition, serializing instructions (e.g., CPUID), interrupts, and privilege level changes may spend considerable cycles in execution, forcing the decoder to wait and incrementing the resource stalls counter. Stalls due to cache misses are not included in resource stalls. Instruction-related stalls count the number of cycles instruction fetch is stalled for any reason, including L1 instruction cache misses, ITLB misses, ITLB faults, and other minor stalls [2] [12].

Comparing computation and stall μCPI, we note that stalls dominate. Stalls comprise 65% of the overall μCPI, 63% of the database μCPI and 75% of the operating system μCPI. The bulk of these stalls are due to cache misses, which will be described in more detail in Section 4.

Figure 3 shows the breakdown of cycles per macro-instruction (CPI). The difference between these values and the values presented in Figure 2 is the ratio of μops per macro-instruction. The measured CPI for the overall system is 2.90, which can be decomposed into the measured database component, 2.52, and the measured operating system component, 6.41. In contrast, the majority of the SPEC 95 programs have a CPI between 0.5 and 1.5 on the Pentium Pro [2].

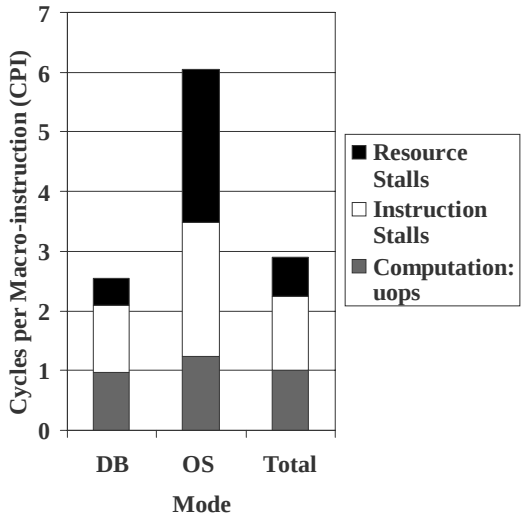


**FIGURE 3. Breakdown of cycles per macro-instruction (CPI) for base system.**

These CPI and stall percentage numbers are roughly consistent with those reported in the literature. Cvetanovic and Donaldson report a CPI of about 3.7 for Sybase running TPC-C on a four-processor in-order Alpha 21164-based server [6]. They found that roughly 80% of the time the processor was stalled. Resource (e.g., "frozen") stalls, such as data cache misses and register and floating point conflicts, comprised 49%, and "dry" stalls, where there is no instruction to issue, comprised the remaining 31%.

## 4 Memory System Behavior

We begin our more in-depth analysis of the CPI components by examining memory system behavior. Table 3 summarizes the characteristics of the Pentium Pro caches. Due to space considerations, in the following sections we present results for the overall system (i.e., database + operating system) only, without detailed discussion of the differences between database and operating system behavior. Detailed performance information for the overall system behavior, database-only behavior and the OS-only behavior, are shown in Section 11.2. In general, the characteristics of the database and OS are similar, with the OS exhibiting somewhat worse behavior. For a discussion of the database-only behavior, we refer the interested reader to [14].

| Characteristic | L1 Inst. | L1 Data | L2 (unified) |
|---|---|---|---|
| Size | 8 KB | 8 KB | 256 KB, 512 KB, 1 MB |
| Associativity | 4-way | 2-way | 4-way |
| Hit penalty | 3 cycles | 3 cycles | 4 cycles (addl.) |
| Non-blocking (e.g., hit-under-miss, miss-under-miss)? | yes | yes | yes |
| Outstanding misses | four | four | four |
| Write policy | write-back | write-back | write-back |

**TABLE 3. Pentium Pro L1 and L2 cache characteristics.**

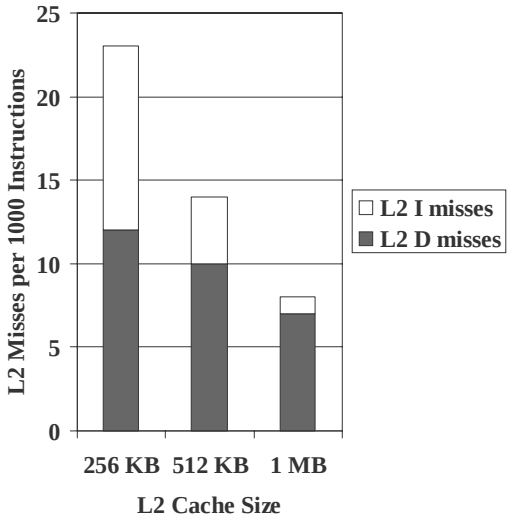


**FIGURE 4. L2 cache miss behavior as a function of L2 cache size.**

### 4.1 How do cache miss rates vary with L2 cache size?

We examined how cache miss ratios change as cache size increases by physically changing the processor boards to measure configurations with 256 KB, 512 KB and 1 MB L2 caches. Figure 4 summarizes the overall system L2 cache miss behavior. As expected, the high-order effect of increasing cache size is a decrease in L2 cache misses, and in the corresponding cache miss rates. Overall L2 miss rates decrease by 31% (from 16% to 9%) as L2 cache size increases from 256 KB to 512 KB, and by another 33% (from 9% to 6%) as the size is further doubled to 1 MB. We see that instruction-related L2 cache misses are nearly fully satisfied by the 1 MB cache; instruction-related miss rates are only 1%. Data miss ratios are still quite high, even for the 1 MB L2 cache, which suggests that even larger L2 caches could be beneficial for this workload.

L1 cache behavior remains consistent across the different L2 cache sizes. About 90 L1 instruction-cache misses are experienced per 1000 instructions retired, yielding miss rates of 5% to 6%. About 50 L1 data-cache misses are experienced per 1000 instructions retired, yielding a miss rate of 7%.

Table 11, Table 12 and Table 13 in Section 11.2 present more detailed data in tabular form for the overall system, the database and the OS, respectively. These tables present counts of cache accesses and misses per 1000 instructions retired, and the resulting cache miss ratios.

Several other recent studies indicate that both larger (e.g., up to 8 MB L2) and more associative (e.g., up to four-way) caches, with longer cache lines (e.g., 64 to 128 bytes), are beneficial to OLTP workloads, including TPC-C [1] [16] [25]. Our conversations with database experts suggest that the instruction stream can be effectively cached for all commercially available databases. Data accesses are more difficult to absorb, however, because the data footprint is much (e.g., up to an order of magnitude [16]) larger than the instruction footprint.

### 4.2 What effects do larger caches have on transaction throughput and stall cycles?

Figure 5 shows the overall system CPI breakdown for the three L2 cache sizes. Doubling the second-level cache size from 256 KB to 512 KB results in a 15% improvement in CPI, and quadrupling the cache size to 1 MB results in a 25% improvement in CPI. The bulk of this improvement is attributable to improvements in both instruction-related and resource stalls. Improvements in database transaction throughput roughly mirror the CPI improvements: transaction throughput increases 16% as L2 cache size is doubled to 512 KB, and 28% as the L2 size is quadrupled to 1 MB. More detailed numerical data is presented in Table 14 in Section 11.3.

### 4.3 Are non-blocking L2 caches successful at reducing memory stalls?

To study the effectiveness of non-blocking second-level caches, we take advantage of a BIOS parameter that allows us to limit the number of outstanding system bus transactions [12]. Ideally, we would limit each of the four processors to a single outstanding bus transaction, and compare behavior under this regime against the behavior under normal bus operation (up to 4 transactions outstanding per processor, and up to 8 transactions across all processors). However, our BIOS parameter permits only two options for the total number of outstanding bus transactions: up to 8, and one. Thus, we instead compare a uniprocessor under the normal bus operation with a uniprocessor limited to a single outstanding bus transaction.

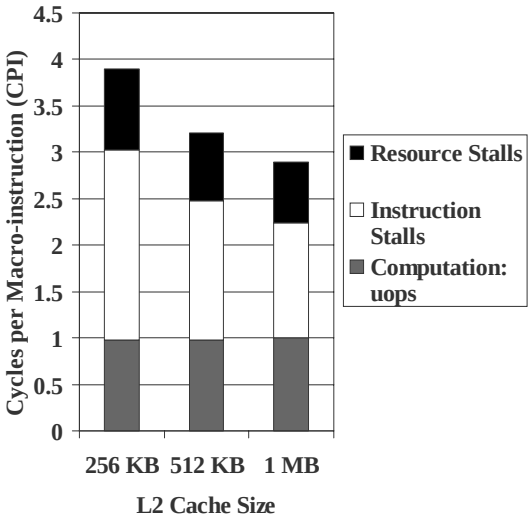


**FIGURE 5. CPI breakdown as function of L2 cache size.**

We performed this experiment for both 1 MB and 256 KB caches, and in both cases found negligible difference between the configuration where the processor was allowed 4 outstanding transactions, vs. the configuration where the processor was limited to a single outstanding transaction. Table 4 presents results for several key parameters for the 256 KB experiment.

| Characteristic | 1 bus req. | <= 4 bus req. |
|---|---|---|
| Measured CPI | 3.08 | 3.02 |
| Computation CPI | 1.00 | 1.01 |
| Resource Stall CPI | 0.52 | 0.49 |
| Instruction Stall CPI | 1.55 | 1.52 |
| Average memory latency | 60 cyc. | 58 cyc. |

**TABLE 4. Effects of non-blocking for 256 KB L2 cache.**

The latencies to memory on an L2 cache miss are too long for the out-of-order engine to cover them completely. It should be possible, though, for two or more L2 misses to overlap with each other, thus reducing memory-related stall time. This does not appear to be the case, however: improvements in stalls are negligible. This behavior leads us to believe that multiple outstanding transactions aren't used often enough to greatly improve the stall component of CPI.

These observations are consistent with conclusions from other related studies. Rosenblum, et al., also observed negligible reductions in stall time due to multiple outstanding L2 misses for TPC-B [25]. However, they demonstrated through simulation that dynamically scheduled processors can hide approximately half of the latency of L1 misses, suggesting that there is a greater potential benefit for non-blocking L1 caches.

## 5 Processor Issues

In addition to memory hierarchy-related stall CPI components, a non-negligible component of CPI (both computation and stall cycles) is due to processor features. In this section we examine some of these processor features.

### 5.1 How useful is superscalar issue and retire?

In the Pentium Pro, three parallel decoders translate x86 macro-instructions into triadic µops. Each cycle, up to three µops can be retired in the out-of-order engine, and up to three x86 instructions can be retired in the in-order engine. Figure 6 and Figure 7 present the profiles for instruction and micro-operation decode and retire behavior, broken down by cycles and instructions, respectively. Section 11.4 presents this data in tabular form, including data for database-only and OS-only behavior.

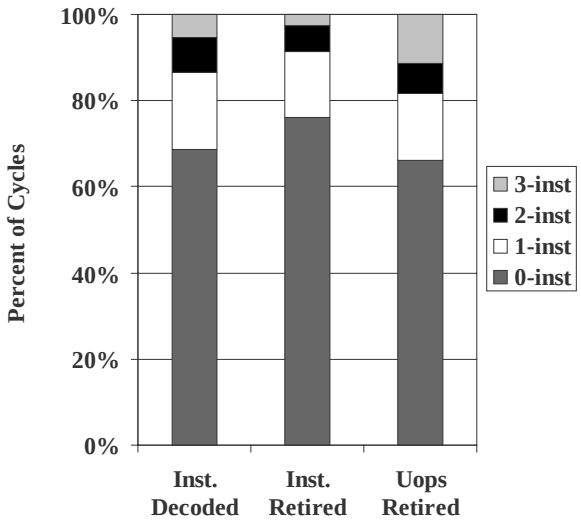


**FIGURE 6. Decomposition of instruction and micro-operation decode and retirement cycles.**

Figure 6 shows that the system experiences a high percentage (e.g., 65% to 75%) of cycles where zero instructions (or µops) are decoded or retired. In contrast, the SPEC integer programs with high L2 cache miss rates show far fewer cycles (i.e., 35% to 51%) with no instructions decoded [2]. Similar behavior is exhibited for SPEC program retirement profiles [2].

Figure 7 presents the same profiles, organized by the percent of instructions (or µops) decoded/retired in each type of cycle. At the macro-instruction level, only 32% of all instructions are decoded in triple-decode cycles, and only 22% of all instructions are retired in triple-retire cycles. Since there is only a modest benefit from macro-instruction triple-decode and -retire cycles, this workload

may not benefit from increases in macro-instruction superscalar width. The system more effectively exploits the superscalar width in the out-of-order execution of µops: 53% of all µops are retired in triple-retire cycles. Thus, increasing superscalar width for the out-of-order engine may prove to be beneficial.

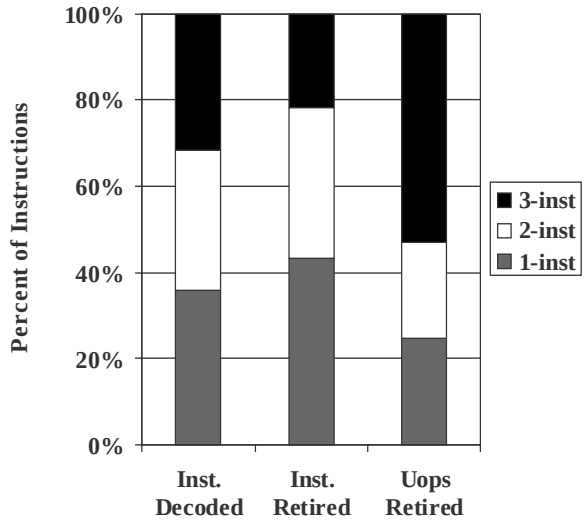


**FIGURE 7. Instruction and micro-operation decode and retirement profiles, broken down by instructions.**

Finally, we note that 1.87 µops are retired for every macro-instruction for the transaction processing workload. In contrast, the average number of µops per macro-instruction is around 1.35 for the SPEC programs [2]. This implies that the transaction processing workload utilizes more complex x86 instructions than the SPEC programs do.

### 5.2 How effective is branch prediction?

Branch behavior for the overall system, the database alone and the operating system alone is shown in Table 5. The behavior of the overall system is nearly identical to the behavior of the database alone. Branches comprise about 21% of the overall instruction mix. The branch misprediction ratio is 15%, which is quite high relative to the branch misprediction ratios of less than 10% for the SPECInt applications. In contrast to our results, Cvetanovic and Donaldson report that the frequency of branches and mispredictions for TPC-C on the Alpha 21164 is comparable to SPECInt [6].

Branch Target Buffer (BTB) miss ratios are also quite high: 56% for both the overall system and the database alone. The operating system exhibits much better BTB miss behavior, with a 33% miss rate. The overall ratio is in contrast to the SPEC workloads, where all programs except one integer program exhibit a BTB miss ratio of less than about 30%. Most SPEC BTB miss ratios are well below 15% [2].

One reason for this branch and BTB behavior is that the compilation process used for the database application employed only traditional compiler optimization techniques, but did not employ more advanced optimization techniques, such as profile-based optimization. This technique, which can move infrequently executed basic blocks out of line and lay out more frequently interacting basic blocks contiguously, could improve branch misprediction and BTB miss behavior, as well as L1 instruction cache miss behavior [24]. However, there is a limit to the savings this technique can offer. Furthermore, processors must be able to efficiently execute code, even if it has not been aggressively optimized. At the least, these miss ratios suggest that this workload requires a much larger BTB, and perhaps a different branch prediction method. Hilgendorf and Heim report that BTB miss rates improve for BTBs up to ~16k entries for OLTP workloads [10].

| Characteristic | Overall | DB | OS |
|---|---|---|---|
| Branch frequency | 21.5% | 20.9% | 27.3% |
| Branch misprediction ratio | 14.6% | 14.3% | 16.6% |
| BTB miss ratio | 55.7% | 55.6% | 32.9% |
| Speculative exec. factor | 1.43 | 1.40 | 1.75 |

**TABLE 5. Branch behavior.** The Pentium Pro processor implements a branch prediction scheme derived from the two-level adaptive scheme described by Yeh and Patt [30]. The branch target buffer (BTB), which contains 512 entries, maintains branch history information and the predicted branch target address. A static prediction scheme (backwards taken, forward not taken) is employed. Mispredicted branches incur a penalty of at least 11 cycles, with the average misprediction penalty being 15 cycles [9].

Finally, we note that the speculative execution factor, or the number of macro-instructions decoded divided by the macro-instructions retired, is 1.4 for the overall system. The speculative execution factor for nearly all SPEC programs is between 1 and 1.3 [2].

### 5.3 Is out-of-order execution successful at hiding stalls?

The Pentium Pro implements dynamic execution using an out-of-order, speculative execution engine, which employs register renaming and non-blocking caches. How effective is dynamic execution for database workloads?

It is difficult to answer this question precisely, without the ability to turn off out-of-order execution. Instead, our approach is to compare non-overlapped CPI vs. measured CPI, as described in [2]. If out-of-order execution is effective, the individual components of the non-overlapped CPI should be overlappable, and the measured CPI should be less than the non-overlapped CPI.

To compute the non-overlapped CPI, we treat the machine as if it were in-order, and explicitly account for computation and all potential stall cycles, such as instruction and data cache miss-related stalls, branch mispredic-

tion penalties, resource-related stalls, and other minor stalls. Cache miss and branch misprediction penalties are computed by multiplying the number of misses per instruction by the associated miss penalty. The computation component is computed from the µop retirement profile, or the percentage of single-, double- and triple-retire cycles, as described in Section 3.1. We assume a steady-state µCPI of 0.33 for triple-retire cycles, 0.5 for double-retire cycles, and 1 for single-retire cycles. The computation CPI is then computed by multiplying the µCPI by the µops per macro-instruction.

Figure 8 presents the non-overlapped and measured CPI for the three different L2 cache sizes. The stacked bars for each configuration include the resource stalls and computation components from Figure 3, and decompose the stall components further to include L1 I and D cache misses, L2 cache misses, branch misprediction stalls, ITLB misses and other minor stalls. The black line in each column marks the measured CPI for that configuration.

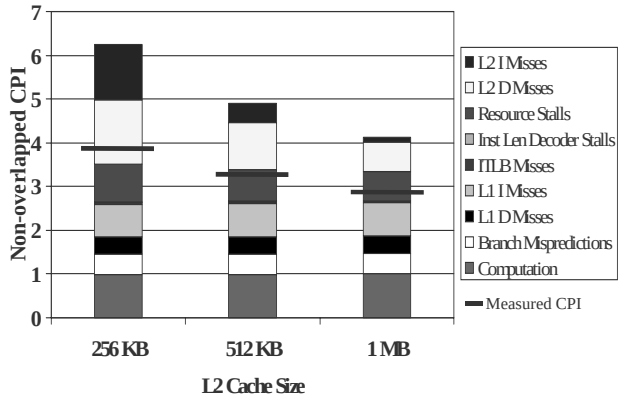


**FIGURE 8. Non-overlapped and measured CPI as a function of L2 cache size.** Latency from each of the components above is shown as a portion of the bar graph. L1 miss penalties are presented in Table 3, L2 miss penalties in the right-most column of Table 7, and branch misprediction penalties in the caption for Table 5.

For all cache sizes, the measured CPI is less than the non-overlapped total. Measured CPI is 62% of its total bar height for the 256 KB L2 cache, 67% for the 512 KB L2 and 70% for the 1 MB L2. Thus, out-of-order execution is somewhat effectively overlapping the CPI components to achieve a lower actual CPI. In contrast, the measured CPI of SPECInt programs is 45% to 65% (averaging 54%) of the non-overlapped total [2]. Since fewer of these components are being overlapped for the database workload, we conclude that out-of-order and speculative execution are somewhat less effective for database workloads than for SPEC workloads.

Other researchers have demonstrated some execution time improvements from out-of-order and increasing issue width for TPC-B [1] [25]. [25] reports a modest 25% speedup from out-of-order, due predominantly to the high percentage of I/O-induced idle time due to their underconfigured simulated disk system.

## 6 Multiprocessor Scaling Issues

In this section, we explore the scalability of the Pentium Pro architecture for running database workloads as the number of processors is varied from one to two to four. Specifically, we examine the memory system utilization, the increase in L2 coherence misses, and the MESI profile for L2 caches as the number of processors grows.

### 6.1 How well does database performance scale as the number of processors increases?

Figure 9 shows the scalability of relative database transaction throughput as a function of the number of processors. For these experiments, L2 cache size is held constant at 1 MB, memory per processor is held constant at 512 MB. As stated in Section 2.4, the ratio of TPC-C throughput to database size (i.e., number of warehouses) must fall within the range from 9 to 12.7. To maintain the correct ratio values, we scaled the size of the database for the two-processor and uniprocessor cases. As demonstrated in the figure, transaction throughput scales reasonably well as the number of processors is increased from one to four.

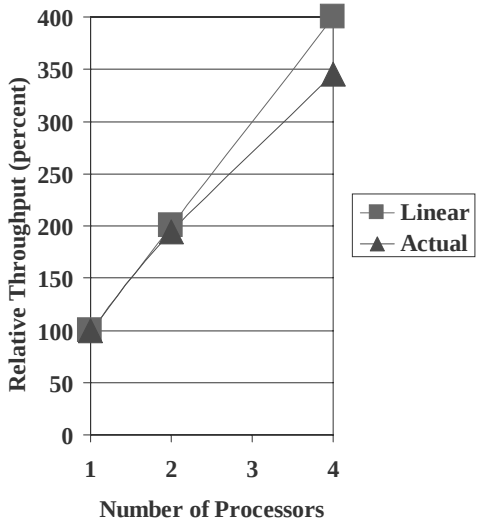


**FIGURE 9. Database throughput scalability.**

### 6.2 How does memory system performance scale with increasing cache sizes and increasing processor count?

Table 6 presents the memory system utilization across all of the processors in the system for different cache sizes. By memory system utilization, we mean the time the bus is busy transferring data (or control information), or the time when the bus is not busy but unavailable. (This latter case

may happen when some bus agent, for example the memory controller, falls behind in its internal processing.) Since this phenomenon has been observed experimentally, but cannot be fully accounted for by the existing counters, we apply a 2X multiplier to the values measured by the counters to properly estimate memory system utilization [29].

As expected, memory system utilization decreases with increasing cache size, and increases with more processors. Utilization grows more slowly for the 1 MB cache size as the number of processors grows, reaching a maximum of nearly 40% for four processors. However, even at these relatively low utilizations, bus activity has an impact on the memory latency perceived by the database and operating system.

Table 7 presents the application memory latency as a function of the L2 size and the number of processors. These cycle counts are calculated from counter values that measure the number of cycles where data read transactions are outstanding on the bus, and the total number of data read transactions. We see that application reads take roughly 60 cycles for the uniprocessor case, but nearly 100 cycles for the four-processor configuration for the 1 MB cache size. Because memory system utilization and application memory latency continue to grow as processors are added, even for larger cache sizes, it is not clear to what degree this memory system will scale to support additional processors.

| L2 Cache Size | 1 P | 2 P | 4 P |
|---|---|---|---|
| 256 KB | 24.2% | 43.5% | 73.9% |
| 512 KB | 17.9% | 33.8% | 56.0% |
| 1 MB | 14.6% | 24.9% | 39.7% |

**TABLE 6. Overall memory system utilization as a function of L2 cache size and number of processors.**

| L2 Cache Size | 1 P | 2 P | 4 P |
|---|---|---|---|
| 256 KB | 58 | 72 | 118 |
| 512 KB | 58 | 72 | 111 |
| 1 MB | 58 | 74 | 97 |

**TABLE 7. Application memory latency as a function of L2 cache size and number of processors.** All values are in processor cycles.

### 6.3 How prevalent are cache misses to dirty data in other processors' cache?

Increasing the L2 cache size also has the potential to increase the number of coherence cache misses. Particularly problematic are cache misses to dirty data in other processors' caches [1] [18]. This difficulty arises in some architectures because many RISC processors have been optimized to give the CPU priority to the L2 cache. In addition, they are typically optimized for providing high bandwidth to the L2 cache for the CPU and incoming requests. These optimizations often have the side effect of increasing the latency for dirty misses [7].

Unfortunately, the Pentium Pro counters provide no events for determining the latency of dirty misses. It is still useful, however, to quantify the frequency of this operation, and determine how it is affected by L2 cache size.

Table 8 presents the percentage of L2 cache misses to dirty data in other processors' caches for one-, two-, and four-processor servers and different L2 cache sizes. It appears that doubling the size of the cache nearly doubles the percentage of L2 misses to dirty data. Likewise, doubling the number of processors roughly doubles the percentage of L2 misses to dirty data. As the cache size increases, this percentage could be quite large. Indeed, Barroso and Gharachorloo report that roughly 60% of misses require a cache-to-cache transfer for dirty data for an 8 MB cache [1].

| L2 Cache Size | 1 P | 2 P | 4 P |
|---|---|---|---|
| 256 KB | 0.4% (0.03) | 3.4% (0.49) | 5.7% (1.38) |
| 512 KB | 0.7% (0.04) | 6.1% (0.62) | 11.8% (1.96) |
| 1 MB | 1.1% (0.04) | 11.2% (0.80) | 22.0% (2.45) |

**TABLE 8. Percentage of L2 cache misses to dirty data in another processor's cache as a function of L2 cache size and number of processors.** The absolute number of dirty misses across all active processors for the five-second measurement window is given in millions by the number in parentheses. We hypothesize that the count for the uniprocessor case is non-zero because the signal used to indicate a hit to dirty data is also raised, in conjunction with another signal, to indicate that the processor wishes to stall during the snoop bus cycle.

### 6.4 Is the four-state (MESI) invalidation-based cache coherence protocol worthwhile for OLTP?

Cache lines may be in one of four states in this protocol: modified (M), exclusive (E), shared (S), or invalid (I). Some coherence protocols don't distinguish between exclusive (exclusive clean) and modified (exclusive dirty). In this section, we investigate whether all four states are used by this workload, and use this analysis to confirm the effectiveness of the cache write policy.

The Pentium Pro counters allow us to monitor the MESI state of an L2 cache line on an access to the L2 cache (e.g., instruction fetch, load or store.) Accesses to "invalid" lines correspond to cache misses, while accesses to lines in other states correspond to hits to an L2 line found in that state, before any modifications due to that access are made.

Table 9 shows the percentages of L2 cache instruction fetches, loads and stores, broken down by MESI state.

| Configuration and L2 Access Type | M | E | S | (Miss) I |
|---|---|---|---|---|
| INST. FETCH | | | | |
| 1 processor | 0.0% | 0.8% | 97.8% | 1.4% |
| 2 processors | 0.0% | 0.0% | 98.8% | 1.2% |
| 4 processors | 0.0% | 0.0% | 98.8% | 1.2% |
| LOAD | | | | |
| 1 processor | 25.4% | 55.5% | 1.1% | 18.0% |
| 2 processors | 21.7% | 17.6% | 43.9% | 16.9% |
| 4 processors | 24.7% | 15.1% | 46.0% | 14.2% |
| STORE | | | | |
| 1 processor | 78.5% | **1.5%** | 0.1% | 19.9% |
| 2 processors | 78.0% | **1.0%** | 4.4% | 16.6% |
| 4 processors | 81.2% | **0.6%** | 6.7% | 10.5% |

**TABLE 9. State of L2 line on L2 hit.** Table shows percentage of L2 accesses.

As expected, nearly all of the instruction fetches that hit in the L2 cache are to shared cache lines. The exclusive state is heavily utilized for loads in the uniprocessor case, as might be expected. In the dual- and quad-processor configurations, hits resulting from loads are distributed across the different states, going predominantly to shared lines, followed by modified lines and finally exclusive lines. The high percentage of load hits to modified lines indicates that the processor reads data in the same line as it has recently written. In addition, over 90% of the database store hits are to modified lines, with few write accesses to shared or exclusive lines. These measurements provide quantitative evidence of the temporal locality present in the workload, and validate the usefulness of the writeback write policy employed by the Pentium Pro caches.

A primary advantage of the exclusive state is that it allows the processor to avoid the invalidation bus transaction on a store to a shared line: if the line is already in the exclusive state, it is the only copy currently cached. However, we see that store hits to exclusive lines rarely occur. Thus, it isn't clear whether the benefits of the E state are worth the cost of implementing the fourth state for this workload. A three-state protocol would be sufficient, and not result in significantly increased bus traffic.

## 7 Future Work

We see many interesting avenues of future research worth pursuing. First, we plan to characterize decision support database workloads, such as the TPC-D benchmark, using a methodology similar to the one used in this study. Another fertile research area is to study the transaction throughput and I/O rates as a function of database buffer size (i.e., memory capacity), including an examination of support for very large memory configurations. Some initial work has been done in this area [28] [6], and more investigation is warranted. Finally, we plan to investigate the importance of system configuration in determining observed performance. Many researchers scale back problem sizes or underconfigure the hardware in their systems when measuring database workloads, violating the TPC guidelines. We want to understand how these concessions impact the measured behavior [15].

## 8 Conclusions

Commercial applications have very different characteristics from technical applications, which are commonly used as benchmarks in the design of computer architectures. For better or for worse, benchmarks help to shape a field. We need to give this important class of applications a chance to help shape the field of computer architecture.

We used the Pentium Pro's built-in hardware counters to monitor numerous architectural features of a four-processor SMP running a properly configured commercial database executing a TPC-C-like transaction processing workload. In addition, we varied several hardware and firmware parameters, including the number of processors, L2 cache sizes and the number of outstanding bus transactions. We investigated the effectiveness of out-of-order and speculative execution, superscalar design, branch prediction, multiprocessor scaling and several cache parameters.

We found that out-of-order execution is only somewhat effective for this database workload. The overall CPI was 2.90, which can be decomposed into a 2.52-cycle database component, which applies for 80% of the execution time, and a 6.41-cycle operating system component, which applies for the remaining 20% of execution time. Stall cycles comprise roughly 65% of the overall CPI. To improve this situation, computer architects could provide more detailed performance counters that allow performance analysts to decompose resource stalls and instruction-related stalls further.

This workload does not fully exploit the existing macro-instruction superscalar issue and retire width. Zero macro-instructions are decoded and retired in more than 65% of the overall cycles. Only 22% of macro-instructions are decoded in triple-decode cycles, and only 32% of macro-instructions are retired in triple-retire cycles. Thus, it is not clear that increasing the macro-instruction decode and retire width further will be beneficial for this workload. However, at the micro-operation level, 53% of all µops are retired in triple-retire cycles, implying that increased superscalar width for the out-of-order engine may be helpful.

The Pentium Pro's branch prediction scheme is not nearly as effective for the TPC-C workload as it is for

SPEC workloads. Furthermore, the branch target buffer's 512 entries are insufficient for this workload. While some performance benefit may come from compilation and binary editing tools, there is room for innovation in branch prediction algorithms and hardware structures to better support database workloads.

We found that caches are effective at reducing the processor traffic to memory: Only 0.4% of all instruction and data accesses reach memory. We found that the nonblocking nature of the L2 cache does not aid in reducing memory stalls. We speculate that the workload does not have multiple outstanding cache misses frequently enough to take advantage of this feature.

Examining the four-state MESI cache coherence policy, we saw that the exclusive state is not often utilized for store operations. Hence, architects could employ a three-state (MSI) cache coherence protocol without significant increase in bus traffic due to writebacks of potentially dirty data.

As expected, the amount of time when the memory system is unavailable decreases with larger caches, and increases as more processors are added. However, even low to medium memory system utilization can increase application memory latency, which affects the scalability of transaction throughput.

## 9 Acknowledgments

We thank Seckin Unlu of Intel for his help in deciphering the Pentium Pro hardware counters and interpreting experimental results. We also thank Aaron Brown, Eric Anderson, Jim Gray, Christoforos Kozyrakis, Megan Thomas, Seckin Unlu, Catharine van Ingen, Wen-Hann Wang, and the anonymous ISCA referees for their comments on earlier drafts of this paper. This research has been supported by DARPA (DABT63-C-0056), the California State Micro program, and by donations and research grants from Informix, Intel, Hitachi, LG Semiconductor, Microsoft, Silicon Graphics/Cray Research, and Sun Microsystems. Kim Keeton is supported by a Lucent Technologies doctoral fellowship.

## 10 References

[1] L. Barroso and K. Gharachorloo. "System design considerations for a commercial application environment," presented at the *First Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW '98)*, in conjunction with the *Fourth High Performance Computer Architecture Conference (HPCA-4)*, February 1998.

[2] D. Bhandarkar and J. Ding. "Performance characterization of the Pentium Pro processor." In *Proc. of HPCA-3*, February, 1997.

[3] R. P. Colwell and R. L. Steck. "A 0.6um BiCMOS processor with dynamic execution," In *International Solid State Circuits Conference (ISSCC) Digest of Technical Papers*, February 1995, pages 176-177.

[4] D. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Inc., 1998.

[5] Z. Cvetanovic and D. Bhandarkar. "Performance characterization of the alpha 21164 microprocessor using tp and spec workloads." In *Proc. of HPCA-2*, pages 270–280, February 1996.

[6] Z. Cvetanovic and D. D. Donaldson. "AlphaServer 4100 performance characterization." *Digital Technical Journal*. 8(4):3-20, 1996,

[7] K. Gharachorloo. Personal communication, April 1998.

[8] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1993.

[9] L. Gwennap. "Intel's P6 uses decoupled superscalar design." *Microprocessor Report*, 9(2):9-15, 1995.

[10] R. B. Hilgendorf and G. J. Heim. "Evaluating branch prediction methods for an S390 processor using traces from commercial application workloads," presented at *CAECW '98*, in conjunction with *HPCA-4*, February 1998.

[11] *Informix OnLine Dynamic Server Administrator's Guide, Vol. 1 and Vol. 2*., Informix Corporation.

[12] Intel Corporation.*Pentium Pro family developer's manual, volume 3: Operating system writer's manual.* Intel Corporation, 1996, Order number 242692.

[13] Intel ISV Performance Labs. "Scaling," white paper, March 1998.

[14] K. Keeton, et al. "Performance Characterization of the quad Pentium Pro SMP using OLTP workloads." In *Proc. of the Intl. Symp. on Computer Architecture*, Barcelona, Spain, June 1998.

[15] K. Keeton and D. A. Patterson. "The impact of hardware and software configuration on computer architecture performance evaluation," presented at *CAECW '98*, in conjunction with *HPCA-4*, February 1998.

[16] A. Maynard, et al. "Contrasting characteristics and cache performance of technical and multi-user commercial workloads." In *Proc. of the 6th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 145–156, October 1994.

[17] L. McVoy and C. Staelin. "lmbench: Portable tools for performance analysis." In *Proc. of the USENIX 1996 Annual Technical Conference*, January 1996.

[18] G. Papadopoulos. "How I learned to stop worrying and love shared memory?" Keynote speech at *HPCA-3*, Feb., 1997.

[19] G. Papadopoulos. "Mainstream parallelism: Taking sides on the smp/mpp/cluster debate." Seminar presented at UC Berkeley Computer Science Division, November 1995.

[20] D. Papworth. "Tuning the Pentium Pro microarchitecture." *IEEE Micro*, pages 8-15, April, 1996.

[21] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufman Publishers, Inc., 1998, 2nd Edition.

[22] S. E. Perl and R. L. Sites. "Studies of windows NT performance using dynamic execution traces," In *Proc. of the Second USENIX Symposium on Operating Systems Design and Implementation*, pages 169-184, 1996.

[23] J. A. Rice. *Mathematical statistics and data analysis*. Duxbury Press, 1995. 2nd edition.

[24] T. Romer, et al. “Instrumentation and optimization of Win32/Intel executables using Etch,” In *Proc. of the USENIX Windows NT Workshop*, pages 1-7, August 1997.

[25] M. Rosenblum, et al. “The impact of architectural trends on operating system performance.” In *Proc. of the 15th ACM SOSP*, pages 285–298, December 1995.

[26] P. Stenstrom, et al. “Trends in shared memory multiprocessing.” *IEEE Computer*, pages 44-50, December, 1997.

[27] J. Torrellas, et al. “Characterizing the cache performance and synchronization behavior of a multiprocessing operating system.” In *Proceedings of the 5th ASPLOS*, pages 162–174, October 1992.

[28] T. Tsuei, et al. “Database buffer size investigation for OLTP workloads.” In *Proc. of the ACM-SIGMOD Conference on Management of Data*,. pages 112-122, May 1997.

[29] S. Unlu. Personal communication, February 1998.

[30] T. Yeh and Y. Patt. “Two-level adaptive training branch prediction.” In *Proc. IEEE Micro-24*, pages 51-61, November 1991.

# 11 Appendix

| μCPI/CPI Component | μCPI: Overall | μCPI: DB | μCPI: OS | CPI: Overall | CPI: DB | CPI: OS |
|---|---|---|---|---|---|---|
| Resource stalls | 0.35 | 0.25 | 1.15 | 0.66 | 0.45 | 2.56 |
| Instruction-related stalls | 0.66 | 0.62 | 1.00 | 1.24 | 1.13 | 2.24 |
| Computation: μops | 0.54 | 0.53 | 0.56 | 1.00 | 0.97 | 1.24 |
| Computed μCPI/CPI | 1.55 | 1.39 | 2.71 | 2.89 | 2.55 | 6.04 |
| Measured μCPI/CPI | 1.55 | 1.37 | 2.87 | 2.90 | 2.52 | 6.41 |

**TABLE 10. Breakdown of cycles per micro-operation (μCPI) and cycles per macro-instruction (CPI) components for base system.**

## 11.1 Overall CPI Behavior

Table 10 shows the breakdown of cycles per micro-operation and cycles per macro-instruction components for the base system (e.g., four processors with a 1 MB L2 cache). (This data is shown graphically in Figure 2 and Figure 3.)

Comparing the measured CPIs against the those calculated from the computation, resource stall and instruction-related stall components, we find that the computed values are within 6% of the measured values in all cases.

The CPI for the operating system alone is more than 2X the CPI of the database alone. The CPI for the overall system more closely resembles the database CPI, since roughly 80% of the execution time is spent in the database.

A final difference between database and operating system CPI behavior is the relative importance of the CPI components. For the operating system, resource stalls are the biggest CPI component, followed by instruction-related stalls and computation. We hypothesize that the importance of resource stalls is due to the prevalence of long-lasting operations in the OS that cause execution to lag behind the decoder (e.g., serializing instructions, interrupt handling, and privilege level changes). In contrast, the order of importance for the database alone is instruction-related stalls, followed by computation and finally resource stalls.

## 11.2 Cache Behavior

Table 11, Table 12 and Table 13 present the cache access and miss behavior as a function of L2 cache size for the overall system, the database alone and the operating system alone, respectively. (The L2 miss data is shown graphically in Figure 4.) We note that DTLB misses are not included in these tables, because they could not be measured reliably.

L1 and L2 cache miss rates for the database resemble those for the overall system. The operating system's L1 I-cache miss rates are somewhat lower than those of the overall system, and the L1 D-cache miss rates are some-

| Characteristic | 256 KB | 512 KB | 1 MB |
|---|---|---|---|
| Instruction fetches | 1738 | 1759 | 1586 |
| Data references | 389 | 388 | 437 |
| L1 I-cache misses | 92 (5%) | 93 (5%) | 93 (6%) |
| L1 D-cache misses | 48 (7%) | 51 (7%) | 51 (7%) |
| ITLB misses | 3 | 4 | 4 |
| L2 Inst.-related misses | 11 (12%) | 4 (4%) | 1 (1%) |
| L2 Data-related misses | 12 (26%) | 10 (19%) | 7 (14%) |
| Overall L2 misses | 23 (16%) | 11 (9%) | 8 (6%) |

**TABLE 11. Overall cache access and miss behavior as a function of L2 cache size.** The values in this table represent the number of events occurring per 1000 instructions retired. Miss ratios for each cache are shown as a percentage in parentheses.

| Characteristic | 256 KB | 512 KB | 1 MB |
|---|---|---|---|
| Instruction fetches | 1557 | 1496 | 1388 |
| Data references | 426 | 452 | 492 |
| L1 I-cache misses | 88 (7%) | 89 (6%) | 89 (7%) |
| L1 D-cache misses | 46 (6%) | 48 (7%) | 48 (7%) |
| ITLB misses | 3 | 4 | 4 |
| L2 Inst.-related misses | 9 (10%) | 3 (4%) | 1 (1%) |
| L2 Data-related misses | 11 (24%) | 8 (18%) | 6 (12%) |
| Overall L2 misses | 20 (15%) | 11 (9%) | 7 (5%) |

**TABLE 12. Database-only cache access and miss behavior as a function of L2 cache size.** The values in this table represent the number of events occurring per 1000 instructions retired. Miss ratios for each cache are shown as a percentage in parentheses.

| Characteristic | 256 KB | 512 KB | 1 MB |
|---|---|---|---|
| Instruction fetches | 3587 | 4041 | 3409 |
| Data references | 224 | 182 | 231 |
| L1 I-cache misses | 130 (4%) | 122 (3%) | 132 (4%) |
| L1 D-cache misses | 70 (9%) | 70 (10%) | 84 (11%) |
| ITLB misses | 2 | 6 | 6 |
| L2 Inst.-related misses | 26 (20%) | 8 (7%) | 2 (2%) |
| L2 Data-related misses | 27 (39%) | 20 (28%) | 17 (22%) |
| Overall L2 misses | 53 (26%) | 28 (14%) | 19 (9%) |

**TABLE 13. Operating system-only cache access and miss behavior as a function of L2 cache size.** The values in this table represent the number of events occurring per 1000 instructions retired. Miss ratios for each cache are shown as a percentage in parentheses.

what higher. L2 cache miss rates for the operating system alone are at least 1.6X that for the database alone.

### 11.3 Impact of L2 Cache Size on CPI and Database Throughput

Table 14 shows the breakdown of CPI for the overall system, the database alone and the operating system alone as a function of L2 cache size. (The overall system data is presented graphically in Figure 5.)

As discussed in previous sections, the OS-only CPI is more than 2X that of the DB-only CPI. For both the overall CPI and the database-only CPI, the relative importance of the components is instruction-related stalls, followed by computation and finally resource stalls. The relative importance of operating system CPI components is somewhat different. For the smaller L2 cache sizes, the instruction-related and resource stalls are most important. For the 1 MB L2 cache, resource stalls are most important, followed by instruction-related stalls and computation.

### 11.4 Effectiveness of Superscalar Issue and Retire

Table 15 and Table 16 present the macro-instruction decode and retirement profiles for the overall system, the database alone, and the operating system alone. Table 17 presents the micro-operation retirement profile, similarly decomposed. The overall system data from these tables was presented graphically in Figure 6 and Figure 7.

Similar behavior is exhibited by the database and the operating system, with a few exceptions. The OS alone experiences a higher percentage of zero-instruction decode and retire cycles, and a higher percentage of zero-µop retire cycles than the database alone. A higher percentage of the OS macro-instructions are decoded and retired in single-decode and -retire cycles, implying that wider macro-instruction superscalar issue and retire would be

| Characteristic | 256 KB | 512 KB | 1 MB |
|---|---|---|---|
| Relative transaction throughput | 100% | 116% | 128% |
| *Overall System* | | | |
| Measured CPI | 3.86 | 3.27 | 2.90 |
| Resource stalls | 0.87 | 0.73 | 0.66 |
| Instruction-related stalls | 2.05 | 1.50 | 1.24 |
| Computation: µops | 0.98 | 0.98 | 1.00 |
| *Database-only* | | | |
| Measured CPI | 3.41 | 2.89 | 2.52 |
| Resource stalls | 0.66 | 0.56 | 0.45 |
| Instruction-related stalls | 1.83 | 1.36 | 1.13 |
| Computation: µops | 0.96 | 0.96 | 0.97 |
| *Operating system-only* | | | |
| Measured CPI | 8.44 | 6.53 | 6.41 |
| Resource stalls | 3.04 | 2.10 | 2.56 |
| Instruction-related stalls | 4.22 | 2.76 | 2.24 |
| Computation: µops | 1.23 | 1.22 | 1.24 |

**TABLE 14. Relative database throughput and CPI breakdown as function of L2 cache size.**

even less useful for operating system code than for database code. Both the database and the operating system retire about half of all µops in triple-µop retire cycles. Thus, micro-operation superscalar retire width appears to be equally as useful for the operating system as it is for the database.

### 11.5 Effectiveness of MESI Cache Coherence Protocol

Table 18 and Table 19 decompose accesses to the L2 cache among the modified (M), exclusive (E), shared (S) and invalid (I) states for the database alone and the operating system alone, respectively. The overall system data is presented in Table 9.

These tables illustrate several differences in behavior between user level and system level. Load operations for the dual- and quad-processor configurations miss the L2 cache nearly 2X as often for the operating system as for the database. Also, the OS experiences a higher percentage of store hits to shared lines for dual- and quad-processor configurations: 20 to 30%, in comparison with less than 5% for the database alone. We hypothesize that there is an increased incidence of read-modify-write operations on operating system data shared between the processors: a load operation misses in the originating processor's cache, causing the line to be loaded from another cache in shared

| Characteristic | % cycles | | | % inst. | | |
|---|---|---|---|---|---|---|
| | Overall | Database | OS | Overall | Database | OS |
| 0-instruction decode | 68.7% | 65.5% | 80.8% | n/a | n/a | n/a |
| 1-instruction decode | 17.9% | 19.4% | 12.1% | 35.7% | 34.7% | 43.2% |
| 2-instruction decode | 8.1% | 8.9% | 5.4% | 32.6% | 31.7% | 38.9% |
| 3-instruction decode | 5.3% | 6.3% | 1.7% | 31.7% | 33.6% | 17.9% |

**TABLE 15. Instruction decode profile.** In the Pentium Pro, three parallel decoders translate IA-32 macro-instructions (e.g., instructions) into triadic micro-operations (e.g., μops). Most instructions are converted to a single μop, some are converted into two to four μops, and complex instructions require microcode, which is a longer sequence of μops. Up to five μops can be issued each clock cycle.

| Characteristic | % cycles | | | % inst. | | |
|---|---|---|---|---|---|---|
| | Overall | Database | OS | Overall | Database | OS |
| 0-instruction retire | 76.2% | 72.7% | 88.2% | n/a | n/a | n/a |
| 1-instruction retire | 15.2% | 17.0% | 8.6% | 43.3% | 41.9% | 55.2% |
| 2-instruction retire | 6.1% | 7.2% | 2.5% | 34.9% | 35.3% | 31.4% |
| 3-instruction retire | 2.6% | 3.1% | 0.7% | 21.8% | 22.8% | 13.4% |

**TABLE 16. Macro-instruction retirement profile.** In the Pentium Pro, up to three x86 instructions can be retired in a single cycle.

| Characteristic | % cycles | | | % inst. | | |
|---|---|---|---|---|---|---|
| | Overall | Database | OS | Overall | Database | OS |
| μops per macro-instruction | 1.87 | 1.84 | 2.23 | | | |
| 0-μop retire | 66.0% | 62.8% | 78.6% | n/a | n/a | n/a |
| 1-μop retire | 15.6% | 16.7% | 10.9% | 24.6% | 23.7% | 28.3% |
| 2-μop retire | 7.0% | 7.9% | 3.9% | 22.2% | 22.5% | 20.3% |
| 3-μop retire | 11.3% | 12.6% | 6.6% | 53.2% | 53.7% | 51.3% |

**TABLE 17. Micro-operation retirement profile.** In the Pentium Pro, up to three μops can be retired in a single cycle.

mode. A subsequent store operation then updates the shared line.

| Configuration and L2 Access Type | M | E | S | (Miss) I |
|---|---|---|---|---|
| INST. FETCH | | | | |
| 1 processor | 0.0% | 0.8% | 98.0% | 1.2% |
| 2 processors | 0.0% | 0.0% | 98.9% | 1.1% |
| 4 processors | 0.0% | 0.0% | 98.9% | 1.1% |
| LOAD | | | | |
| 1 processor | 23.7% | 56.8% | 1.2% | 18.3% |
| 2 processors | 21.2% | 17.7% | 45.2% | 16.0% |
| 4 processors | 25.7% | 15.6% | 46.4% | 12.2% |
| STORE | | | | |
| 1 processor | 76.3% | **1.6%** | 0.0% | 22.1% |
| 2 processors | 79.9% | **1.1%** | 1.9% | 17.0% |
| 4 processors | 85.8% | **0.6%** | 2.9% | 10.5% |

**TABLE 18. State of L2 line on L2 hit for the database.**
Table shows percentage of L2 accesses.

| Configuration and L2 Access Type | M | E | S | (Miss) I |
|---|---|---|---|---|
| INST. FETCH | | | | |
| 1 processor | 0.0% | 0.7% | 96.8% | 2.5% |
| 2 processors | 0.0% | 0.0% | 97.5% | 2.5% |
| 4 processors | 0.0% | 0.0% | 98.0% | 2.0% |
| LOAD | | | | |
| 1 processor | 37.1% | 46.7% | 0.4% | 15.8% |
| 2 processors | 25.3% | 17.0% | 34.0% | 23.6% |
| 4 processors | 18.5% | 12.1% | 42.7% | 26.7% |
| STORE | | | | |
| 1 processor | 89.2% | **0.9%** | 0.8% | 9.1% |
| 2 processors | 67.4% | **0.5%** | 17.7% | 14.4% |
| 4 processors | 57.0% | **0.3%** | 25.8% | 16.8% |

**TABLE 19. State of L2 line on L2 hit for the operating system.** Table shows percentage of L2 accesses.