Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads

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,web}@informix.com

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 bus 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 transaction processing and decision support 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 [25]. 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 [18]: databases motivated 32% of the server volume in 1995, and will motivate 39% of the 2000 server volume [25]. 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 [15]. 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. As a result, much of their execution time is spent waiting for I/O completions. 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 applica-

1063-6897/98 $10.00 © 1998 IEEE

tions, 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, main memory bandwidth, 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 database workloads than for SPEC. The branch prediction algorithms and hardware support do not work nearly as well for database workloads. Superscalar issue and retire is only marginally helpful for this workload.

Not surprisingly, we found that caches are effective at reducing the processor traffic to memory. Our data support the rule of thumb that doubling the L2 cache size gives about half the benefit seen from the previous doubling. 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, and could likely be omitted in favor of a simpler three-state protocol. Finally, multiprocessor scaling of this workload is good, but even modest bus 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] [15] [21] [24] [26]. (This benchmark has been withdrawn by the Transaction Processing Council.) A few have examined the more complex TPC-C order-entry workload [6] [15]. Only two of these papers have explored the effectiveness of out-of-order execution for the TPC-B workload [1] [24]. 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 database OLTP 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, memory bandwidth, 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 over-

come 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.

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.

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 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 (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

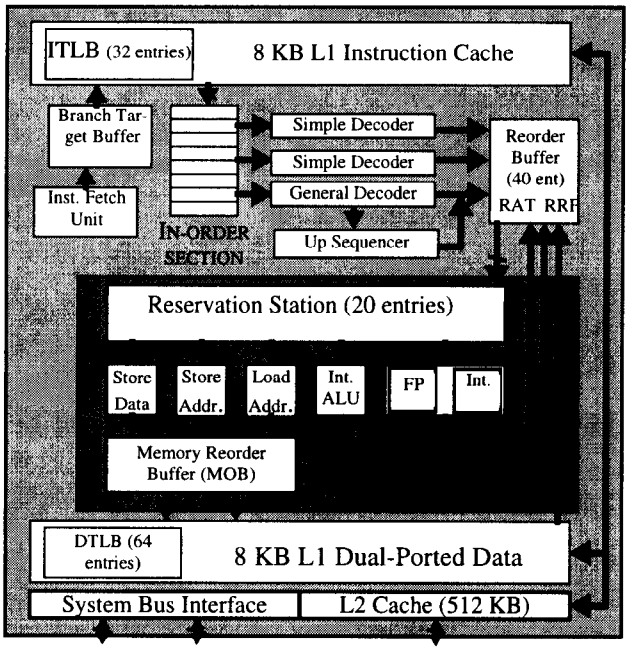

FIGURE 1. Block diagram of Pentium Pro processor architecture. The out-of-order execution engine is shown as the dark rectangle; the remainder of the processor is considered the in-order section.

-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 (CPμ) of 0.33.

A more detailed description of the Pentium Pro's architectural features can be found in [2] [3] [8] [11] [19]. 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 [10], 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 [7]. 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 [27].

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 [12] [14]. 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 memory bandwidth. 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. When the number of processors was halved, the buffer memory available to the database was also halved.

Memory bandwidth was varied by changing parameters in the memory controller to modify the memory refresh rate and page open mode. Our goal was to obtain the widest range in memory bandwidth by modifying the fewest parameters. We started with the base system, and toggled the DRAM page-open mode from on to off. When page-open mode is on, the DRAM page on chip is left "open," resulting in a lower latency access to the same page. When this mode is off, the page is "closed" between subsequent accesses, resulting in no performance optimization for same-page accesses. The remaining configurations were obtained by increasing the refresh frequency from once every 15.625 µs to once per 3 µs, 1.5 µs, and 1 µs, respectively. (We chose not to vary the DRAM RAS/CAS (row and column access times), which would degrade the latency of all requests and hence degrade bandwidth, because it resulted in only a 20% degradation in average memory latency.)

Parameter	Values
L2 cache size	256 KB, 512 KB, 1 MB
Number of processors	1, 2, 4
lmbench read bandwidth	213, 175.5, 145, 113 (MB/s)
lmbench read latency	190, 266, 332, 470 (ns)

TABLE 2. Summary of architectural parameters varied. The memory read bandwidth and the average memory read latency are given by uniprocessor microbenchmarks that are part of the lmbench suite [16]. Each of the memory latencies shown corresponds to one of the memory bandwidths given (e.g., 190 ns corresponds to 213 MB/s). Although lmbench calculates latency as well as bandwidth, since we are actually increasing the refresh bandwidth, it is better to think of our experiment as reducing the bandwidth available to the processor, rather than increasing memory latency. In Section 5.3 and Section 6.2 we will use counter values to compute the average memory latency observed by the database application.

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 [11]. We used a total of 82 event types for the data presented in this paper. For each hardware configuration, we did 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 40-minute 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

CPμ /CPI Component	CPμ: DB	CPμ: OS	CPμ: Overall	CPI: DB	CPI: OS	CPI: Overall
Computation: μops	0.53	0.56	0.54	0.97	1.24	1.03
Resource stalls	0.25	1.15	0.45	0.45	2.56	0.91
Instruction-related stalls	0.62	1.00	0.70	1.13	2.24	1.37
Computed CPI/CPμ	1.39	2.71	1.68	2.55	6.04	3.32
Measured CPI/CPμ	1.37	2.87	1.70	2.52	6.41	3.38

TABLE 3. Breakdown of cycles per micro-operation (CPμ) and cycles per macro-instruction (CPI) components for base system.

in a different order for each run, to greater increase the coverage of the counters.

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 [22]. 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. For the database, the standard deviation for a given event for a given processor was less than 5% 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 summarized in Table 3. 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. 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 CPI for the database, the operating system, and the overall system. The database CPI is 2.52, while the OS CPI is over two times that value, at 6.41. Our system spends between 76% and 80% of the time executing database code in user mode, across the four processors. The remaining time is spent in system mode, with less than 1% idle time. Considering this ~78%/~22% breakdown, we can compute an overall CPI for the system of 3.39. In contrast, the majority of the SPEC 95 programs have a CPI between 0.5 and 1.5 on the Pentium Pro [2].

Table 3 presents the cycles per macro-instruction, CPI, and the cycles per micro-instruction, CPμ, for the database and the operating system. We can decompose the monolithic CPI numbers by determining computation and stall cycles. Computation CPI is calculated based on the μop retire profile presented in Section 5.1: we assume μops retired in single-retire cycles require one cycle in the steady state, double-retire cycle μops take 0.5 cycles, and triple-retire μops need 0.33 cycles. This determines the number of cycles per μop, as shown in Table 3. (Recall that the theoretical minimum CPμ for this machine is 0.33.) To obtain the average CPI, we then multiply the CPμ by the number of μops per macro-instruction. These values are shown in Table 3.

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, except for cache misses. 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. 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] [11].

Putting these components together, we compute a CPI value that very closely approximates the measured value. For the database, our estimates are within 3% to 5% of the measured value; our operating system estimates are typically within 10%.

Comparing computation and stall CPI, we note that stalls dominate the overall CPI. Stalls comprise 62% of the database CPI and 73% of the operating system CPI. The bulk of these stalls are due to cache misses, which will be described in more detail in Section 4.

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. Due to space considerations, in the following sections we present results for the database only, without detailed discussion of operating system performance. The characteristics of the OS are similar to those of the database, so our conclusions are not affected by this decision to focus on the database results. For the complete set of data, we refer the interested reader to [13].

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. Table 4 presents counts of cache accesses and misses per 1000 instructions retired, and the resulting cache miss ratios. As expected, the high-order effect of increasing cache size is an decrease in L2 cache misses, and in the corresponding cache miss rates.

Overall L2 miss rates decrease by 45% as L2 cache size increases from 256 KB to 512 KB, and by another 36% 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 the database workload.

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] [15] [24]. 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 [15]) larger than the instruction footprint.

A rule of thumb that has been suggested for predicting the benefits of increasing L2 cache size is the following: each doubling of the cache size gives about half the benefit seen with the previous doubling [28]. (For example, if we observe a 10% improvement from 512 KB to 1 MB, we should see 5% from 1 MB to 2 MB.) The data shown in Table 4 roughly support this assertion. For the database, the number of L2 misses per 1000 instructions decreases by 45% (e.g., from 20 to 11) as the L2 cache is increased from 256 KB to 512 KB, and by 20% relative to the original (e.g., from 11 to 7), as the cache size is increased from 512 KB to 1 MB. Our data support this rule of thumb; more data must be collected to further evaluate its validity.

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 (24%)	11 (9%)	7 (5%)

TABLE 4. 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. DTLB misses are not included in this table, because they could not be measured reliably. The Pentium Pro has an 8 KB on-chip L1 instruction cache and an 8 KB on-chip L1 data cache. The data cache is 2-way associative, while the instruction cache is 4-way associative. Hits have a 3-cycle latency, and misses that hit in the L2 cache have an additional 4-cycle latency. Both are nonblocking, and support both hit-under-miss and miss-under-miss; at most four cache misses outstanding to the L2 are permitted. The L2 cache is a unified, 4-way associative, cache located off of the processor chip, in the same multi-chip module. In this base architecture, the L2 size is 1 MB. The L2 is also nonblocking: at most four bus transactions to memory/I/O are permitted. All caches employ a writeback policy, using a writeback buffer to reduce the miss penalty when a miss requires replacing a dirty block.

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

Table 5 shows the database throughput and CPI breakdown for the three cache sizes. Halving the second-level cache size results in nearly a 10% degradation in transaction throughput, while quartering the cache size results in roughly a 20% degradation in transaction throughput. Overall CPI increases by 15% when the cache is halved, and by 35% when the size is quartered, with most of the change allocated to instruction-related stalls. The database experiences a degradation in instruction-related stalls and resource stalls with the smaller cache size.

4.3 Are non-blocking L2 caches successful at hiding memory latency?

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 [11]. We compare a uniprocessor under the normal bus operation (up to 4 transactions outstanding) with a uniprocessor limited to a single outstanding bus transaction.

We performed this experiment for both 1 MB and 256 KB caches, and in both cases found negligible difference

Characteristic	256 KB	512 KB	1 MB
Relative transaction throughput	78.2%	90.7%	100%
Measured CPI	3.41	2.89	2.52
Computation: µops	0.96	0.96	0.97
Resource stalls	0.66	0.56	0.45
Instruction-related stalls	1.83	1.36	1.13

TABLE 5. Relative database throughput and CPI breakdown as function of L2 cache size.

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 6 presents results for several key parameters for the 256 KB experiment.

Characteristic	1 bus req.	<= 4 bus req.
Measured CPI	2.79	2.74
Computation CPI	2.81	2.77
Resource Stall CPI	0.40	0.38
Instruction Stall CPI	1.43	1.40
Average memory latency	60 cyc.	58 cyc.

TABLE 6. 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.

Although we offer no quantitative evidence to support the claim, we speculate that non-blocking first-level caches will permit processors to hide the shorter latencies of L2 misses. Rosenblum, et al., demonstrate that dynamically scheduled processors can hide approximately half of the latency of L1 misses for TPC-B [24]. They also reported that they didn't observe multiple outstanding L2 misses frequently enough to significantly reduce stall time.

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. Table 7 presents the macro-instruction decode behavior for the database for the base system. Table 8 presents the macro-instruction retirement profile, and Table 9 presents the micro-operation retirement profile. In each table the "% cycles" column refers to the percent of total cycles where that operation occurs. "% inst." refers to the percent of instructions that are decoded/retired in that type of cycle.

Characteristic	% cycles	% inst.
0-instruction decode	65.5%	n/a
1-instruction decode	19.4%	34.7%
2-instruction decode	8.9%	31.7%
3-instruction decode	6.3%	33.6%

TABLE 7. 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.

In all three cases, the database experiences a high percentage (e.g., 60 - 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 many fewer cycles (i.e., 35% to 51%) with no instructions decoded [2]. Similar behavior is exhibited for SPEC program retirement profiles.

Characteristic	% cycles	% inst.
0-instruction retire	72.7%	n/a
1-instruction retire	17.0%	41.9%
2-instruction retire	7.2%	35.3%
3-instruction retire	3.1%	22.8%

TABLE 8. Macro-instruction retirement profile. In the Pentium Pro, up to three x86 instructions can be retired in a single cycle.

Since there is only a modest benefit from the triple-instruction-decode and -retire cycles, this workload may not need a machine with such wide superscalar macroinstruction decode and retire capabilities. The database is better able to exploit the three-way retire in the out-of-order execution engine.

Finally, we see that roughly 1.8 µops are retired for every macro-instruction for the database. In contrast, the

Characteristic	% cycles	% inst.
µops per macro-instruction	1.84	
0-µop retire	62.8%	n/a
1-µop retire	16.7%	23.7%
2-µop retire	7.9%	22.5%
3-µop retire	12.6%	53.7%

TABLE 9. Micro-operation retirement profile. In the Pentium Pro, up to three µops can be retired in a single cycle.

average number of µops per macro-instruction is around 1.35 for the SPEC programs [2]. This implies that the database utilizes more complex x86 instructions than the SPEC programs do.

5.2 How effective is branch prediction?

Branch behavior for the database and operating system is shown in Table 10. Branches comprise about 21% of database instructions. The branch misprediction ratio for the database is 14%, which is quite high relative to the branch misprediction ratios of less than 10% for the SPEC-Int applications. 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: roughly 55% for both the database and the operating system. This 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 [23]. 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 the database and OS require 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 [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 database. The speculative execution factor for nearly all SPEC programs is between 1 and 1.3 [2].

Characteristic	
Branch frequency	20.9%
Branch misprediction ratio	14.3%
BTB miss ratio	55.6%
Speculative execution factor	1.40

TABLE 10. Branch behavior. The Pentium Pro processor implements a branch prediction scheme derived from the two-level adaptive scheme described by Yeh and Patt [29]. 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 [8].

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?

In Figure 2 and Figure 3, we decompose the stall components of CPI further to include L1 I and D cache misses, ITLB misses, L2 cache misses, branch misprediction stalls, and other minor stalls. Each graph depicts the CPI for one L2 cache size, for a variety of memory configurations, as described in Table 2. Due to space constraints, we present results for only 256 KB and 1 MB L2 caches. If out-of-order execution is doing its job properly, some of these components should be overlapped with each other during execution. In the figure we also retain the resource stalls and computation components from Table 3. The black line in each column marks the measured CPI for that configuration.

For all cache sizes and memory system configurations, we see that the measured CPI is less than the non-overlapped total. Measured database CPI is 60% of its total bar height for the 256 KB L2 cache and 67% for the 1 MB cache. Thus, out-of-order execution is somewhat effectively overlapping the CPI components to achieve a lower actual CPI. In contrast, the CPI of SPEC programs is about 20% to 50% lower than the individual components due to overlapped execution [2]. Since fewer of these components are being overlapped for the database workload, we conclude that out-of-order and speculative execution provide less value for database workloads than for SPEC workloads.

Out-of-order execution is somewhat more effective for the 256 KB cache size, as evidenced by measured CPI being a smaller percentage of the total non-overlapped CPI than in the 1 MB case. As memory latencies rise, nearly doubling, this percentage remains relatively constant. This

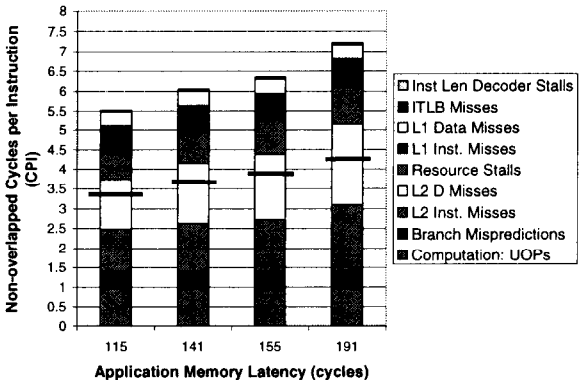

FIGURE 2. Non-overlapped CPI for the 4-processor, 256 KB configuration. Latency from each of the components above is shown as a portion of the bar graph. Application memory latency for each configuration is computed from counter values that measure the number of cycles where data read transactions are outstanding on the bus, and the number of data read transactions. These values are used as L2 cache miss penalties. L1 and L2 hit times are as described in the caption for Table 4, and branch misprediction penalties are as described in the caption for Table 10.

is somewhat disappointing, since it indicates that the out-of-order engine is unable to hide memory latency.

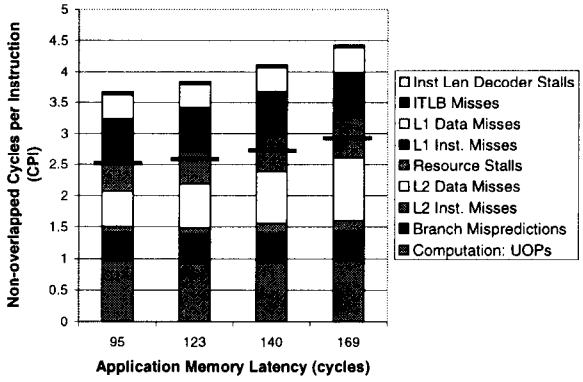

FIGURE 3. Non-overlapped CPI for the 4-processor, 1 MB configuration. Latency from each of the components above is shown as a portion of the bar graph.

Other researchers have demonstrated some execution time improvements from out-of-order and increasing issue width for TPC-B [1] [24]. [24] reports only 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 system bus utilization, the increase in bus invalidation traffic, 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?

Table 11 shows a summary of the relative transaction throughput and system and database configuration parameters. As seen in the table, transaction throughput scales reasonably well as we move from one to four processors.

Configuration Parameter	1 P	2 P	4 P
Relative Txn. Throughput	29%	56%	100%
Warehouses (W)	29%	58%	100%
Database Buffer Pool Size	25%	50%	100%

TABLE 11. Summary of database configurations and throughputs. For these comparisons, we hold the L2 cache size constant at 1 MB, and hold the memory bandwidth constant at 213 MB/s. As stated in Section 2.4, the ratio of TPC-C throughput to warehouses (i.e., database size) must fall within the range from 9 to 12.7. To maintain the correct ratio values, we changed the size of the database for the two-processor and uniprocessor cases. In addition, we modified the number of database buffers, as shown above. All values are shown relative to the four-processor base case configuration.

6.2 How does bus performance scale with increasing cache sizes and increasing processor count?

Table 12 presents the system bus utilization across all of the processors in the system for different cache sizes. By bus utilization, we mean the time the bus is busy transferring data (or control information), or the time it 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 bus utilization [28].

Interestingly, even though transaction performance scales well, we find that system bus utilization grows more slowly as the number of processors increases from two to four. However, even at these low utilizations, bus activity has an impact on the memory latency perceived by the database and operating system. For instance, database memory reads appears to take roughly 60 cycles for the uniprocessor case, but roughly 100 cycles for the four-processor case, as shown in Table 13.

Thus, it is not clear to what degree this bus will scale to support additional processors.

L2 Cache Size	1 P	2 P	4 P
256 KB	23.4%	42.7%	73.9%
512 KB	18.0%	33.8%	55.1%
1 MB	15.3%	24.6%	38.7%

TABLE 12. Overall bus utilization as a function of L2 cache size and number of processors.

L2 Cache Size	1 P	2 P	4 P
256 KB	58	71	115
512 KB	58	72	110
1 MB	58	74	95

TABLE 13. 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] [17]. This difficulty arises in some architectures because the dirty data must be implicitly written back to memory before being read into the requesting processor's cache. The Pentium Pro optimizes this dirty miss case by allowing the processor that owns the dirty line to directly transfer the line to the requesting processor, while memory snoops the bus to update its copy. Although this case may be less of an issue for the Pentium Pro architecture, it is still useful to quantify the frequency of this operation, and determine how it is affected by L2 cache size.

Table 14 presents the percentage of L2 cache misses to dirty data in other processors' caches for one-, two-, and four-processor servers and a range of L2 cache sizes. It appears that doubling the size of the cache 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 to up to 8 MB, the 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].

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). We want to investigate whether all four states are used by this workload, and use this analysis to confirm the effectiveness of the cache write policy.

L2 Cache Size	1 P	2 P	4 P
256 KB	1.1% (0.01)	3.1% (0.36)	5.3% (0.35)
512 KB	0.4% (0.02)	5.7% (0.48)	11.1% (1.45)
1 MB	0.6% (0.02)	10.0% (0.60)	21.5% (1.84)

TABLE 14. 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.

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 15 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%	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 15. State of L2 line on L2 hit. Table shows percentage of L2 accesses.

As expected, we see that 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 uni-

processor case for the database, as might be expected. In the dual- and quad-processor servers, 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 95% 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 paper. 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 [27] [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 [14].

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, memory system bandwidth, and 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 3.39, which can be decomposed into a 2.52-cycle database component, which applies for 80% of the execution time, and a 6.48-cycle operating system component, which applies for the remaining 20% of execution time. Stall cycles comprise 62% of the database CPI and 73% of the OS 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.

The database sees only modest benefit from superscalar issue and retire. Zero instructions are decoded and retired in more than 65% of the database cycles. Furthermore, zero μops are retired in 63% of the cycles for the database. Architects could potentially reduce the macroinstruction decode and retire width without adverse impact on this workload.

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 0.33% of database access and 0.47% of OS accesses reach memory. Our data support the rule of thumb that doubling the cache size gives about half the benefit seen with the previous doubling. We found that the nonblocking nature of the L2 cache did not aid in hiding memory latency. 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 bus is unavailable decreases with larger caches, and increases as more processors are added. However, even low to medium bus 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 Tho-

mas, Seckin Unlu, Catharine van Ingen, Wen-Hann Wang, and the anonymous 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] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1993.

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

[9] 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.

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

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

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

[13] K. Keeton, et al. "Performance Characterization of the quad Pentium Pro SMP using OLTP workloads," extended version, UC Berkeley Computer Science Division Technical Report UCB//CSD-98-1001, April 1998.

[14] 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.

[15] 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.

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

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

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

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

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

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

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