Profiling I/O Interrupts in Modern Architectures

Lambert Schaelicke, Al Davis, and Sally A. McKee

UUCS-99-022

Department of Computer Science
University of Utah
Salt Lake City, UT 84112

November 1999.

Abstract

As applications grow increasingly communication-oriented, interrupt performance quickly becomes a crucial component of high performance I/O system design. At the same time, accurately measuring interrupt handler performance is difficult with the traditional simulation, instrumentation, or statistical sampling approaches.

One of the most important components of interrupt performance is cache behavior. This paper presents a portable method for measuring the cache effects of I/O interrupt handling using native hardware performance counters. To provide a portability stress test, the method is demonstrated on two commercial platforms with different architectures, the SGI Origin 200 and the Sun Ultra-1. This case study uses the methodology to measure the overhead of the two most common forms of interrupt traffic: disk and network interrupts. The study demonstrates that the method works well and is reasonably robust. In addition, the results show that disk interrupts behave similar on both platforms, while differences in OS organization cause network interrupts to behave very differently. Furthermore, network interrupts exhibit significantly larger cache footprints.

1 Introduction

Understanding and modeling the complex interactions among applications, modern operating systems, and current, high-ILP microprocessors is difficult. Simulation is slow, and incapable of modeling large workloads or the nondeterminism of real system behavior. Application source instrumentation is similarly limited in its ability to capture a program's dynamic behavior. Instrumentation tools like pixie/prof [18], gprof [9], or the more recent ATOM [20], Eel [12], and MTOOL [8] perturb the target application's behavior, potentially distorting the effect to be measured. Statistical sampling tools like Compaq's CPI [1] usually do not provide the resolution required to measure short events such as interrupt handlers.

The upshot is that non-intrusive methods must be used to measure modern computer systems. Simulating with simplified models or instrumenting programs may be sufficient to identify some application bottlenecks, but those methods are generally inadequate for gaining insight into whole system performance.

Fortunately, several microprocessors (e.g., the Intel Pentium-Pro [2], MIPS R10000/R12000 [14], IBM Power2 [23], Compaq Alpha [3], Sun UltraSPARC [21], and HP PA-8000 [11]) now include hardware support for performance monitoring: hardware counters track events such as cache misses, TLB misses, branch mispredictions, coherence operations, and instructions issued or graduated. Although the features and application interfaces of these performance counters vary greatly among platforms, a common subset that includes cache misses at various levels of the cache hierarchy is sufficient to measure one of the most important aspects of system performance.

This paper presents a case study using hardware event counters to measure the overhead costs of I/O interrupts on two commercial platforms, the SGI Origin 200 and the Sun Ultra-1. We measure the cache hierarchy effects for each system under three scenarios: in the idle state, and when processing either a disk interrupt or a network interrupt.

We study I/O interrupt performance for two reasons. First, interrupts affect all aspects of OS performance, and they represent an increasingly important bottleneck in modern systems. Indeed, interrupt performance becomes crucial for gigabit networking, or very parallel or pipelined I/O. For instance, Chase et al. [7] find that interrupt handling accounts for between 8% and 25% of receiver overhead in their measurements of TCP/IP performance on an Alpha 21164 workstation. The trend towards multithreaded and modular operating systems increases the interrupt handling cost further. Second, interrupts are particularly difficult to measure accurately; one of the main contributions of this work is thus a portable methodology for isolating and measuring fine-grained system effects.

The two platforms used were chosen because they represent different architectural approaches. The SGI Origin is a four-way multiprocessor based on the dynamically-scheduled MIPS R10000 processor, while the Sun Ultra-1 is a single-processor workstation with a superscalar in-order CPU.

We find that for disk interrupts the two systems behave very similarly, with an interrupt handling footprint of about 3-4 kilobytes of data and 10-13 kilobytes of instructions. Network interrupts are more costly in both systems, with IRIX showing a four-fold increase over the disk interrupt cost with respect to instruction cache misses. Furthermore, we find that interrupt handlers do not exhibit sufficient spatial locality to make use of the larger cache lines of many second level caches. Finally, the interrupt handling cost is increased by the highly modular structure of modern operating systems. This is evident on the SGI platform in the high instruction cache miss rate observed by an application whenever a network interrupt is serviced.

In the remainder of this paper we discuss this methodology in greater detail, describe the two platforms used for the experiments, and present results for L1 data and instruction cache behavior and unified L2 cache behavior for disk and network interrupts.

2 Methodology

This section first outlines a straightforward approach to measuring the interrupt handlers' cache effects, and then presents an improved approach that measures the desired effects with greater acuity.

Basic Approach

To measure the cache effects of interrupts from an application's perspective, our initial approach employs an application with perfect cache behavior that repeatedly touches all cache lines. An I/O interrupt disturbs the application by replacing a certain number of cache lines in each level of the cache hierarchy. In addition, the interrupt handler itself incurs cache misses. To measure these effects, the experimental application first performs an operation that will lead to an I/O interrupt at a later point in time (phase 1 in Figure 1). It then starts the event counters and fills the cache with application data (phase

2). After a fixed time period, during which the I/O interrupt has been handled (phase 3), the application touches every cache line again, and stops the event counters (phase 4). The number of observed cache misses corresponds to the number of cache lines that have been replaced by the interrupt handler.

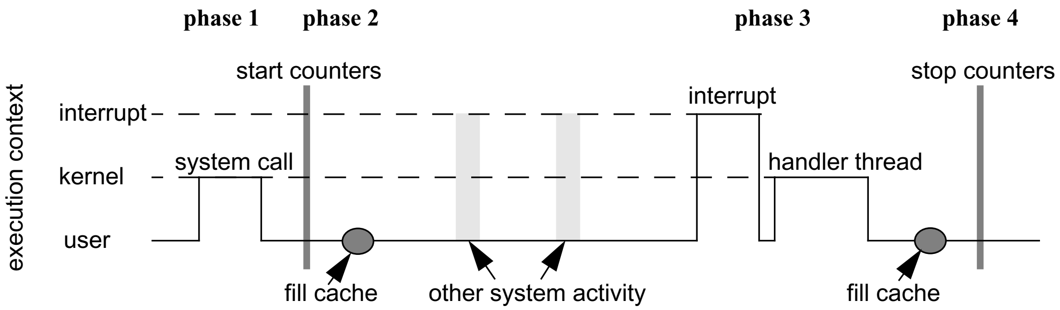

Figure 1a: Basic Experiment

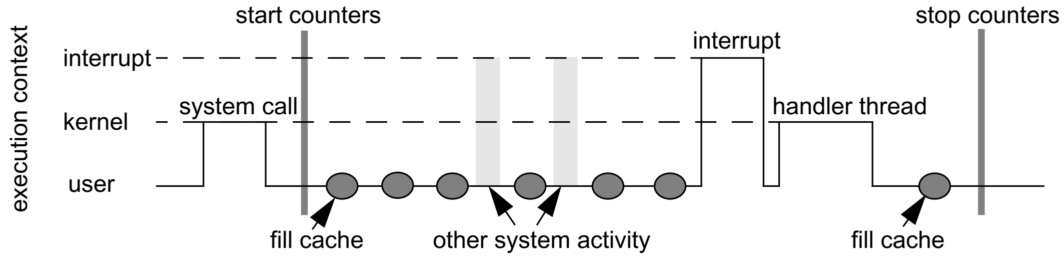

Figure 1b: Refined Experiment to Eliminate Effects of other System Activity

Figure 1: Structure of Experiments

Figure 1 (a) above illustrates this process. Note that the performance counters are started after the interrupt has been scheduled but before the cache has been filled with application data. This ensures that the interrupt handler incurs the maximum number of cache misses.

Counting cache and TLB misses in different *counter modes* allows one to observe a variety of effects. In user mode, the number of cache misses indicates how many cache lines have been replaced by the interrupt handler. Since the experimental application touches the entire cache, this represents the worst-case cost of interrupt handling, and can be used to estimate the cache footprint of the interrupt handler. When counting in kernel or exception mode, the experiments measure the number of cache misses incurred by the interrupt handler itself.

Filling a cache with application data is accomplished by reading an array with a stride equal to the cache block size and a size that exactly matches the cache size. The application does not actually consume the array data, so to reliably defeat compiler optimizations, this array scan is implemented via pointer chasing, and the last pointer is the return value of the routine. By varying the size of this array, one can measure L1, L2, or TLB effects. Since L2 caches are normally physically indexed and tagged, it is important that the array used to fill the L2 cache be contiguous in physical memory. IRIX provides a tool that gives a hint to the operating system about where to create superpages (which are contiguous in physical memory). Unfortunately, Solaris lacks a similar feature. However, on a system with sufficient free pages, memory allocation will often return contiguous physical pages, which can then be locked down.

Filling the instruction cache can be similarly accomplished with a routine that repeatedly branches forward, touching every instruction cache block exactly once.

2.1 Refined Measurements

Initial experiments using this methodology revealed that normal periodic system activity introduces a significant number of cache misses in an otherwise idle system. For instance, waiting for a disk event takes on the order of 10 to 20 ms. If the application does not touch the cache during this time, many of the application's L1 data cache lines are evicted by system threads before the I/O interrupt handler runs. This system activity is caused by various periodic clock interrupts and related handler threads (e.g., in IRIX, the process scheduling clock interrupt occurs every 10 ms, and a 1 ms clock is used for thread scheduling) and network broadcast messages.

Figure 1 (b) illustrates a refined approach that isolates the effects of the disk or network interrupt. After scheduling the interrupt, the application waits for a period of time long enough for the scheduled interrupt to occur. For the disk experiments, this has been experimentally determined to be 25 ms. For network events, the latency is set to 2 ms. During this period, the application continuously touches all cache lines, forcing them into the cache. This guarantees that the interrupt handler being measured incurs close to the maximum number of cache misses.

The baseline numbers for an idle system are obtained by omitting the interrupt scheduling system call from the process described above. This measures the number of cache misses in an idle system over a fixed period of time.

2.2 Generating Interrupts

Two different classes of I/O interrupt handlers are measured: disk and network. They represent the most common type of I/O related interrupts in today's workstations and servers. To generate a disk interrupt, the application issues an asynchronous read system call to a 16-byte file residing on a local disk. This file size is smaller then or equal to the smallest cache block size of the studied systems. Consequently, the interrupt handler overhead will not be dominated by the data transfer from the buffer cache to the application buffer. The asynchronous read allows the application to continue executing while the data is transferred from disk. Optionally, the system notifies the application of the transfer completion by sending a signal. To guarantee that the file is read from disk, the file cache is flushed by reading a large file (on the order of the main memory size of the experimental machine). Furthermore, to defeat the filesystem prefetching mechanisms, the application randomly selects a test file from a set of 16 files.

Network events are generated by sending a short message through a TCP socket to a remote machine that then returns the message to the sender. Observed round-trip times for this message exchange are on the order of 350 to 500 μs on both platforms, sufficient time to fill the cache and start the performance counters before the reply message arrives. For the L2 experiments the message round-trip time is increased to 3 ms by delaying the response message to allow the application to touch every line of the L2 cache at least once before the network interrupt happens.

2.3 Statistics

All experiments are repeated at least 16 times, until the 95% confidence interval was less than 10% of the arithmetic mean of the samples (5%). Before calculating the mean, high and low outliers were removed. High outliers are a result of context switches caused by other system activity such as broadcast network packets, in which case the result will not measure the number of cache misses due to the I/O interrupt alone. Low outliers arise when the interrupt does not occur within the fixed time period, in which case the result corresponds to the baseline for an idle system. The results presented here are the arithmetic mean of the remaining data points.

3 Experimental Setup

The interrupt overhead measurements are performed on an SGI Origin 200 [13] and a SUN Ultra-1 workstation, both running commercial UNIX variants. Table 1 summarizes the relevant features of these platforms.

	SGI Origin 200	Sun Ultra-1
OS	IRIX 6.5	Solaris 2.6
CPU	4 x 225 MHz R10000	143 MHz UltraSPARC-1
L1 D-Cache	32 Kbyte 32 byte blocks 2-way set-assoc. write-back virtulaly indexed physically tagged	16 Kbyte 32 byte blocks direct-mapped write-through virtually index virtually tagged
L1 I-Cache	32 Kbyte 64 byte blocks 2-way set-associative virtually index physically tagged	16 Kbyte 32 byte blocks 2-way set-associative virtually index virtually tagged
L2 Cache	2 Mbyte 128 byte blocks 2-way set-assoc. phys. index & tagged	0.5 Mbyte 64 byte blocks direct-mapped phys. indexed & tagged
TLB	64 entries unified with micro I-TLB	64 entry D-TLB 64 entry I-TLB
Main Memory	1024 Mbyte	256 Mbyte

Table 1: Experimental Platforms

3.1 SGI Origin 200

The SGI Origin 200 used in this study is a four-processor shared memory multiprocessor, consisting of two nodes connected through a CrayLink network. Each node contains two MIPS R10000 processors, one half of the total physical memory, two PCI buses and a variety of I/O adapters.

The MIPS R10000 [14] is a four-way superscalar, out-of-order CPU. Instructions are reordered in three independent queues of 16 entries each, from which they are issued to two integer units, two floating point units, or a load-store unit. For precise exception handling, original instruction order is maintained in a 32-entry reorder buffer. Up to four instructions can be decoded and retired per cycle, and a maximum of five instructions can be issued to the execution units each cycle.

The R10000 provides two performance counters that support a rich set of events, ranging from issued and retired instructions (including total instructions, loads, and stores) to total cache misses per cache and TLB misses. Events can be counted in any combination of user, kernel, and exception modes. Access to the counters is privileged, but IRIX provides an ioctl() interface through the /dev/proc filesystem. A system call is used to configure and start the performance counters and to retrieve event counts. The system binds the performance counters to the application, thus restricting the counted events to the desired process context.

3.2 Sun Ultra-1

The Sun Ultra-1 is a uniprocessor workstation based on the UltraSPARC-1 processor [21], which is a superscalar, in-order design. It features two integer ALUs, three floating point units, a specialized graphics unit, and a load/store unit. It can issue two integer instructions per cycle and two floating-point instructions per cycle. Unlike the R10000, instructions issue strictly in order. An interesting feature is that the L1 data cache blocks are split into two 16-byte subblocks that share a cache tag. Upon a cache miss, only 16 bytes are read from the L2 cache. For this study, the L1 data cache is treated as if it consisted of 16-byte blocks.

The UltraSPARC chip also provides two performance counter registers that can track the total cache accesses and cache hits for each cache level. L1 data cache events are categorized into read and write accesses. Events can be counted in user mode, kernel mode, or both. Unlike the MIPS processor, the SPARC does not distinguish between kernel and exception mode.

The freely available kernel module *perfmon* [4] acts as a device driver that gives user programs access to the performance counters. An ioctl() system call is used to configure the counters, letting the application then read and write the counter registers directly through a set of assembler routines. An unfortunate side-effect of this simple interface is that the counters count all events, regardless of the current process context.

3.3 Platform Comparison

While the hardware organizations of these two platforms are significantly different, the operating systems share many characteristics. Both Solaris 2.6 [6] and IRIX 6.5 [18] are modern UNIX variants.

The kernels are internally multithreaded to efficiently support multiprocessor platforms, multithreaded applications, and real-time processing [6]. In these kernels, interrupts are handled by threads. The interrupt handler itself determines the cause of the interrupt and schedules the appropriate kernel thread to run. The interrupt handler thread is then scheduled like any other kernel thread, and it synchronizes with other threads via normal synchronization primitives. Advantages of this approach include the potentially higher degree of concurrency in the kernel, the ability to block interrupt handling threads without the danger of deadlocks, and finer scheduling control. However, invoking the kernel thread scheduler for every interrupt is also likely to increase the interrupt handling overhead.

The entire IRIX kernel is mapped through a single 32-megabyte page table entry, which is pinned in the TLB. Interrupt handling thus incurs no TLB misses and replaces no application TLB entries. The UltraSPARC performance counters do not support counting TLB misses, hence it was impossible to determine whether Solaris uses a similar superpage approach to reduce or avoid kernel TLB misses.

IRIX has the ability to selectively isolate processors from the system, which prevents the kernel from scheduling any processes on that CPU. To further reduce the system activity, one can disable the "fast" 1 KHz clock interrupts for selected processors. However, other I/O interrupts are always handled by the CPU that is "closest" to the I/O device. In contrast, the Sun Ultra-1 is a uniprocessor workstation, thus it is not possible to reduce idle system activity by isolating a processor.

4 Results

To illustrate how the cache effects of interrupt handling can be computed from our experiments, we first present detailed results for one representative experiment.

Table 2 below summarizes the cost of the performance counter system calls. These numbers represent the number of cache lines replaced from the perspective of an application. It is interesting to note that the performance counter interface in Solaris is significantly less costly, particularly when considering that the L1 data cache consists effectively of 16 byte cache lines.

Cache	O-200	Ultra-1
L1 Data Cache Misses	388	100
L1 Instruction Cache Misses	360	195
L2 Cache Misses	695	158

Table 2: Performance Counter Overhead

Table 3 below lists the observed cache misses for disk interrupts both the basic and the refined experiments on the SGI Origin. The first row represents baseline results for an idle system, where the application touches all cache lines only at the end of the measurement period. More than half the L1 data cache lines have been replaced by the 10 ms clock interrupt handler and the performance counter system calls. A disk interrupt increases the number of replacements by approximately 17-24 cache lines. This demonstrates that the basic experiment is unable to measure the cache effects of a disk interrupt, because most of the I/O interrupt-induced cache misses are hidden by the clock interrupt and the performance counter overhead.

All results presented in the following sections are thus obtained with the refined methodology (Figure (b)), in which the application repeatedly touches all cache lines. Baseline results for this approach appear in the second group of measurements (on the right) in 3. Over a period of 25 ms, the application incurs over 4000 cache misses, even though the system is otherwise idle. The disk interrupt introduces an additional 100 to 130 cache misses.

Experiment	Cache Misses	Experiment	Cache Misses
idle system over a 25 ms period	657	idle system over a 25 ms period, application repeatedly touches all cache lines (refined experiment)	4587
disk read, no signal delivered upon completion	674	disk read, repeated cache touch, no signal delivered upon completion	4691
disk read, signal delivered to application	681	disk read, repeated cache touch, signal delivered to application	4717

Table 3: L1 D-Cache User-Level Results for Disk Interrupt on the Origin 200

4.1 Disk Interrupt Effects

Tables 4 - 6 summarize the results for both platforms for the L1 data and instruction caches and the L2 cache. As expected, when the kernel delivers a signal to the application at the end of the disk transfer the number of cache misses increases slightly.

Mode	Description	O-200	Ultra-1
User	no signal delivered	104	215
	signal delivered	130	216
Kernel	no signal delivered	52	273
	signal delivered	63	228
Except.	no signal delivered	48	n/a
	signal delivered	53	n/a

Table 4: L1 D-Cache Misses for Disk Interrupts

Mode	Description	O-200	Ultra-1
User	no signal delivered	169	402
	signal delivered	156	444
Kernel	no signal delivered	58	100
	signal delivered	61	110
Exception	no signal delivered	80	n/a
	signal delivered	86	n/a

Table 5: L1 I-Cache Misses for Disk Interrupts

The L1 data cache effects are similar for both platforms. The data cache footprint of the interrupt handler is approximately 3-4 kilobytes (e.g., 298 lines × 32 bytes = 9536 bytes on the Origin). Note that since the Ultra-1 cache line size is 16 bytes, the number of cache misses incurred by the application is twice that of the Origin 200.

Since the Ultra-1 performance counters do not distinguish between kernel and exception mode, the Solaris kernel mode cache misses should correspond to the sum of the kernel and exception cache misses in IRIX. However, since Solaris events are counted regardless of the process context, the result is higher then the corresponding IRIX results. In IRIX, on the other hand, the cache misses incurred by the interrupt handler thread are not included in these measurements, hence the sum of kernel and exception misses is less then the total number of replaced cache lines.

The L1 instruction cache results follow the same trend as for the data cache. Both platforms show an instruction cache footprint of about 10-13 kilobytes. The Solaris interrupt handler replaces about 15-18% more instruction cache lines than the IRIX handler. However, the Solaris results show a large variation (especially in kernel mode) and occasionally did not reach the 95% confidence interval of 5%of the mean. This is mostly due to the fact that the Origin 200 is a four-processor system, where clock interrupt and network packet processing can be moved to other processors. On the single-CPU Ultra-1, all normal system activity is handled by one processor. This introduces more noise into the experiments, especially when measuring over the relatively long period of 25 ms.

It is also important to note that due to the inclusion property of caches, instruction cache lines may be evicted when the corresponding L2 cache line is replaced, regardless whether it is replaced by data or instructions. This explains why the number of kernel instruction cache misses is lower then the total number of cache lines being replaced by the interrupt handlerr..

Mode	Description	O-200	Ultra-1
User	no signal delivered	298	253
	signal delivered	300	243
Kernel	no signal delivered	70	245
	signal delivered	65	264
Exception	no signal delivered	64	n/a
	signal delivered	74	n/a

Table 6: L2 Cache Misses for Disk Interrupts

The number of L2 cache lines that are replaced by the interrupt handler is approximately the same for both platforms, with the Sun results being slightly lower. Since the L2 cache line size on the UltraSPARC processor is half that of the MIPS R10000, this indicates that the interrupt handler does not exhibit sufficient spatial locality to benefit from a larger cache line size. This is confirmed by the observation that the sum of the number of L1 instruction and data cache lines replaced is almost equal to the number of replaced L2 cache lines on the SGI platform.

On the Sun, on the other hand, the sum of L1 instruction and data cache misses in user mode is higher than the number of L2 misses, indicating either that: in the smaller L2 cache of the UltraSPARC, instructions and data overlap and conflict with each other, creating a smaller footprint.

4.2 Network Interrupt Effects

Tables 7 - 9 summarize the cache misses introduced by the network interrupt handler. For both platforms, network interrupts incur more cache misses then disk interrupts. In particular, on the Solaris platform, the number of data cache misses doubles, while the instruction cache misses remain the same as for the disk interrupt case. On the IRIX platform, on the other hand, the number of cache misses (both instruction and data) more than quadruples. These results highlight a significant difference in how IRIX handles disk and network interrupts: while disk interrupts are handled by a general-purpose interrupt thread, network interrupt handling is split between the interrupt handler thread and a separate network process. This results in about four times as many user-level cache misses, since the application experiences one additional context switch. Figure 2 shows an IRIXview [19] context switch trace for network interrupts.

The IRIX scheduler returns control to the application for a few microseconds before switching to the network process. During this time the application touches every cache line, incurring many cache misses. This effect is particularly pronounced for the instruction cache measurements, since one "iteration" through the instruction cache takes only 5.2 µs, whereas a data cache "iteration" takes 64 µs.

Mode	Origin 200	Ultra-1
User	493	361
Kernel	201	425
Exception	186	n/a

Table 7: L1 D-Cache Misses for Network Interrupts

Mode	Origin 200	Ultra-1
User	1621	427
Kernel	105	429
Exception	142	n/a

Table 8: L1 I-Cache Misses for Network Interrupts

Mode	Origin 200	Ultra-1
User	1114	465
Kernel	164	523
Exception	193	n/a

Table 9: L2 Cache Misses for Network Interrupts

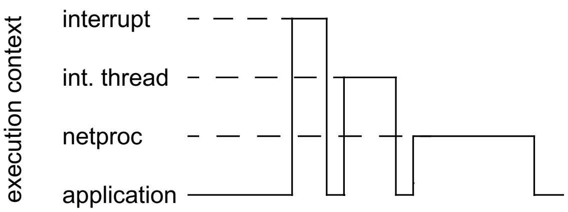

Figure 2: Network Interrupt Handling in IRIX

The SGI performance counters do not capture cache misses incurred by the network process. Consequently, the sum of kernel and exception cache misses is lower than the number of observed user mode cache misses. For the same reason, the number of kernel cache misses in Solaris is higher then the sum of kernel and exception level misses in IRIX.

The lower number of cache misses for Solaris suggests that either the network process is scheduled immediately after the interrupt, or Solaris does not use a separate process to handle network interrupts.

5 Related Work

Profiling tools can be evaluated according to the *overhead* they impose on the programs or systems being measured, *scope* (whether they can be used to profile the entire system, or only application programs), *acuity* (the granularity of program units for which they can assign costs), *accuracy* (the ability to attribute stall cycles accurately), and *portability*. Which approach or tool is best for a given profiling task depends on what is being profiled and how the results will be used. In general, the easiest tools to use tend to provide the least useful data.

Gprof [9] instruments an application at compile time to generate execution call graph information at run-time. Tools like pixie [18] and jprof [15] rewrite executables to collect basic execution frequencies, generate traces, or extract a control-flow graph. All impose high overhead costs, and gprof and jprof only operate on a procedure granularity. Eel [12] and Etch [16] also edit executable images, providing the similar data collection facilities along with the ability to perform simulation based on dynamic information (e.g., simulating the cache effects of the memory reference stream). Pixie and EEL have finer resolution, but like the others, they can only be used to profile application performance.

Sampling systems like Morph [25] and Etch apportion execution time by statistically sampling the program-counter. These tools can be used to analyze both application and OS performance, but they rely on timer interrupts to generate their samples, and thus cannot be used to profile the interrupt handler itself. Intel's Vtune performance analyzer [10] is a collection of tools to extract and analyze architecture-specific software performance data; vtune performs statistical sampling, call graph extraction, and (like Eel and Etch) dynamic assembly and simulation of small portions of code, and can be used to profile the whole Pentium III system.

Platforms like SimOS [17] can be used to model both operating system and application performance with high accuracy. Unfortunately, simulation is extremely slow, and performance is much more deterministic than in a physical system, where complex interactions inject a high degree of non-determinism. For instance, physical page layout and interrupt timing cause interactions with the interrupt handler with respect to conflicts in the cache, TLB, and hardware speculation resources.

Morph and Compaq's CPI (Continuous Profiling Infrastructure) [1] both provide transparent, low-overhead, continuous profiling on modern processors. The Morph monitor generates statistical profiles by sampling the program counter once per hardware clock interrupt. Since the monitor's profile information concerns only the PC, Morph can not attribute stall times to particular instructions, but the system performs its profiling transparently at the overhead cost of only 0.1% of the application's execution time. CPI provides instruction-level information with high accuracy, attributing dynamic stalls to the instructions that actually incur those stalls. Such systems can deliver useful profiles of complex applications that are too large to simulate reasonably. For in-order processors, CPI relies on statistical profiling, but for out-of-order architectures, it leverages hardware support for extracting instruction-level information.

Hardware performance counters greatly enhance the quantity and accuracy of profiling data, expanding the set of events that can be measured in isolation or in relation to each other. Bhandarkar and Ding [2] use the Pentium Pro performance counters to characterize the performance of that architecture according to CPI, cache misses, TLB misses, branch prediction, speculation, resource stalls, and micro-operations. They present their measurements for the SPEC benchmarks as averages per instruction over the application’s entire execution (e.g., for memory data references) or as averages per thousand cycles executed (e.g., for cache and TLB misses).

Without the event counters provided by today's high-performance architectures (most of which are superscalar, speculative, and out-of-order) [3][2][23][14], it would be impossible to study many of the complex interactions that affect system performance. These mechanisms not only speed the measurement process, but also provide a means of validating simulation models. The tools that use these counters provide software interfaces and data visualization support to make the hardware profiling mechanisms easier to use, but analyzing system performance with them can still be labor intensive.

Regardless of the system being measured or the tools chosen, deriving accurate results with any profiling approach requires great care: quantization errors are a problem with the coarser-grained approaches, and statistical approaches lack accuracy and can potentially exhibit correlations between other system activity and the sampling itself. Using any profiler to isolate fine-grain system effects can be tricky, especially in a multithreaded OS. In addition to the technical means to extract the raw system performance data, one also needs a methodology for determining what to measure, and how best to measure it. If comparisons are to be made among different systems, this methodology must be portable. This paper presents one such methodology that has proved useful in measuring the I/O interrupt performance on two different commercial systems.

Others have endeavored to analyze details of operating system performance. Torrellas et al. [22] study the caching and synchronization performance of a multiprocessor OS, whereas previous work had focused on the cache activity of applications alone. They use a hardware monitor on an SGI POWER 4D/430, an SMP with four 33MHz MIPS R3000 nodes, capturing complete behavior without perturbing the system. They find that OS-induced cache misses account for up to one-fourth of the stall time for the parallel compilation, multiprogrammed workload, and commercial database that they measure.

Endo et al. [5] present measurement techniques and benchmark methodologies for analyzing the performance of interactive workloads on Windows NT and Windows 95. Rather than rely on throughput measurements to study behavior of latency-critical events (such as user input or network packet arrival), they measure actual event-handling latency in the context of realistic, interactive, desktop applications. As in the study presented here, they strive to measure short-duration events without access to the application or OS source code, and they rely on hardware counters to provide the raw data to be analyzed in constructing the profile.

6 Conclusion

This case study introduces a portable methodology for measuring relatively small events in the operating system, without requiring access to the OS source code. This method has been used to measure the cache effects of disk and network interrupts on two commercial platforms with diverse architectures. The cache effects can be categorized as 1) effects that are observed by the application, namely cache lines that have been replaced by the interrupt handler, and 2) cache misses incurred by the actual interrupt handler. In addition, this study presents measured I/O interrupt overheads in two commercial systems — numbers that to the authors' knowledge have not been published elsewhere.

The disk interrupt effects are very similar for the two platforms. Both disk interrupt handlers have data cache footprints of about 3-4 kilobytes and instruction cache footprints of 10-13 kilobytes. For the UltraSPARC-1 processor, this means that about two thirds of the instruction cache is polluted by the interrupt handler. The number of L2 cache lines that are replaced by the disk interrupt is approximately the same on both systems. Since the two platforms use different L2 cache line sizes, this shows that interrupt handlers do not have sufficient spatial locality to benefit from longer cache lines.

On each platform, the network interrupt handler has a bigger footprint than the disk interrupt handler, but this gives rise to different effects on each platform. On Solaris, the network interrupt replaces about twice as many data cache lines, but the instruction footprint increases only slightly. In IRIX, on the other hand, the number of instruction cache misses increases by more than a factor of four. The poor instruction cache behavior results from IRIX's organization: part of the interrupt processing is delegated to a separate process that is scheduled a few microseconds after the primary interrupt handler completes. In addition, the IRIX primary network interrupt handler has a larger cache footprint than the disk interrupt handler.

These observations confirm that from the application's perspective interrupts in multithreaded operating systems have a higher cost than in traditional operating systems. The additional thread scheduling activity and context switches cause many more application cache misses.

While measuring OS behavior from the application level is a very portable approach, it requires that the observed system be as quiet as possible. This presents an especially tricky problem on a single-CPU workstation, where periodic system activity in an otherwise idle system can introduce a large variation in the measured results. Multiprocessors (such as the SGI Origin 200) that allow clock interrupt processing to be off-loaded to other CPUs produce more reliable results.

In most architectures, the hardware performance counters can be configured such that they are directly accessible by user-level software. For protection reasons the two platforms used in this study did not support this but instead required a system call for configuring the counters. This approach has the benefit that the experimental application does not necessarily have to run with root privileges when measuring the OS behavior's effects on the application. In addition, the system call can perform consistency checks on the arguments and reduce the possibility of measurement error due to incorrect use of the hardware counters. This method of accessing the counters introduces additional cache misses, but the resulting overhead can easily be measured and accounted for in the experiments.

The diversity of the performance counter capabilities and interface semantics can pose a problem for studies like this one. For instance, the fact that some platforms bind the event counters to the calling process while others count all events regardless of the current process context makes it more difficult to compare results among these platforms. A more standardized interface would be beneficial for making fair comparisons. Nonetheless, the methodology used herein, the comparative performance numbers presented, and the pitfalls exposed should prove useful to other practitioners in the design and analysis of efficient systems.

References

[1] J.M. Anderson, L.M. Berc, J. Dean, S. Ghemawat, M.R. Henzinger, S.-T.A. Leung, R.L. Sites, M.T. Vandevoorde, C.A. Waldspurger, and W.E. Weihl. Continuous Profiling: Where Have All the Cycles Gone? *Proceedings of the 16th ACM Symposium on Operating Systems Principles,* October 1997.

[2] D. Bhandarkar and J. Ding. Performance Characterization of the Pentium Pro Processor. *Proceedings of Third IEEE Symposium on High-Performance Computer Architecture (HPCA'97)*, 1997.

[3] J.H. Edmondson, P.I. Rubinfeld, P.J. Bannon, B.J. Benschneider, D. Bernstein, R.W. Castelino, E.M. Cooper, D.E. Dever, D. R. Donchin, T.C. Fischer, A.K. Jain, S. Mehta, J.E. Meyer, R.P. Preston, V. Rajagopalan, C. Somanathan, S.A. Taylor, G.M. Wolrich. Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor. *Digital Technical Journal,* Vol 7, No. 1, July 1995.

[4] R. Enbody. *Perfmon User's Guide*. Available at: http://www.cse.msu.edu/~enbody/perfmon/index.html

[5] Y. Endo, C. Wang, J.B. Chen, and M. Seltzer. Using Latency to Evaluate Interactive System Performance. *Proc. Second Symposium on Operating System Design and Implementation (OSDI'96)*, October 1996.

[6] J.R. Eykholt, S.R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, D. Stein, J. Voll, M. Weeks, and D. Williams. Beyond Multiprocessing … Multithreading the SunOS Kernel. *Proceedings of Summer'92 USENIX.* June 1992.

[7] A. Gallatin, J. Chase, and K. Yocum. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. *Proceedings of the 1999 USENIX Technical Conference*. June 1999.

[8] A.J. Goldberg and J.L. Hennessy. MTOOL: An Integrated System for Performance Debugging Shared Memory Multiprocessor Applications. *IEEE Trans. on Parallel and Distributed Systems,* pages 28-40, January 1993.

[9] S. Graham, P. Kessler, and M. McKusick. gprof: A Call Graph Execution Profiler. *SIGPLAN Notice*s, 17(6): 120--126, June 1982.

[10] Intel Corp. *Visual Tuning Environment*. Available at: http://developer.intel.com/design/perftool/vtune

[11] A. Kumar. The HP PA-8000 RISC CPU. *IEEE Micro,* Volume 17, No. 2, March 1997.

[12] J.R. Larus and E. Schnarr. Eel: Machine-Independent Executable Editing. *Proceedings of the SIGP{LAN'95 Conference on Programming Language Design and Implementation,* pages 291-300, 1995.

[13] J. Laudon and D. Lenoski. *System Overview of the SGI Origin 200/2000 Product Line.* White Paper, Silicon Graphics, Inc., 1997.

[14] MIPS Technologies. *MIPS R10000 Microprocessor User's Manual*, Version 2.0, 1996.

[15] J.F. Reiser and J.P. Skudlarek. Program Profiling Problems, and a Polution via Machine Language Rewriting. SIGPLAN Notices, 29(1):37-45, January 1994.

[16] T. Romer, G. Voelker, D. Lee, A. Wolman, W. Wong, H. Levy, B. Bershad, and B. Chen. Instrumentation and Optimization of Win32/Intel Executables Using Etch. *Proceedings of the USENIX Windows NT Workshop.* pages 1-7. August 1997.

[17] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod. Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM TOMACS Special Issue on Computer Simulation*, 1997.

[18] Silicon Graphics Inc. *IRIX man pages*. Available at: http://techpubs.sgi.com/library/.

[19] Silicon Graphics Inc. *IRIXview User's Guide.* 1998. Available at: http://techpubs.sgi.com

[20] A. Srivastava and A. Eustace. ATOM: A System for Building Customized Program Analysis Tools. Compaq Western Research Laboratory Technical Report 94.2, March 1994.

[21] Sun Microsystems. *UltraSPARC User's Manual.* 1997.

[22] J. Torrellas, A. Gupta, and J. Hennessy. Characterizing the Caching and Synchronization Performance of a Multiprocessor Operating System. *Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 162-174, October 1992.

[23] E.H. Welbon, C.C. Chan-Nui, D.J. Shippy, and D.A. Hicks. POWER2 Performance Monitor. *PowerPC and POWER2: Technical Aspects of the New IBM RISC System/6000*, IBM Corp. SA23-2737, pp. 55-63

[24] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz. Performance Analysis Using the MIPS R10000 Performance Counters. *Proceedings Supercomputing'96,* November 1996.

[25] X. Zhang, Z. Wang, N. Gloy, J. B. Chen, and M. Smith. System Support for Automatic Profiling and Optimization. *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997.