To appear in The Proceedings of the USENIX Winter 1994 Technical Conference.

Memory Behavior of an X11 Window System

J. Bradley Chen
School of Computer Science
Carnegie Mellon University

Abstract

We used memory reference traces from a DEC Ultrix system running the X11 window system from MIT Project Athena and several freely available X11 applications to measure different aspects of memory system behavior and performance. Our measurements show that memory behavior for X11 workloads differs in several important ways from workloads more traditionally used in cache performance studies. User instruction cache behavior is a major component in overall memory system delays, with significant competition within and between address spaces. User TLB miss rates are up to a factor of two higher than other ill-behaved integer workloads. Write-buffer stalls, data cache behavior, and uncached memory reads can be problematic for microbenchmarks, but they are not an issue for the realistic applications we tested.

1. Introduction

We have used memory reference traces from DEC Ultrix, the X11 window system from MIT Project Athena, and freely available X11 applications to explore several aspects of memory system behavior and performance for X11 workloads. We measured behavior within the system, server and client, as well as interaction between address spaces.

Our analysis shows that memory behavior for X11 workloads differs substantially from that of traditional workloads, particularly in the instruction cache and TLB. Competition within and between contexts in the instruction cache has significant performance impact. This cache competition appears difficult to avoid in a direct mapped cache, suggesting that higher associativity may be required. TLB designs that do not accommodate the demands of large interactive systems may also become performance problems.

X11 workloads, as compared to the SPECmarks [23] and other more traditional workloads for behavioral studies of memory systems, differ in several fundamental ways:

- **Large program text.** Even the largest SPECmarks are small compared to X11. At 688 KBytes, *gcc* stands out among the SPECmarks for its large text segment[1]. X11 servers commonly have as much as 1.8 megabytes of text, more than twice that of *gcc*. X11 clients also tend to have large code. The two real-world X11 clients used in this study, *gs* and *splot*, have text sizes of 946 Kbytes and 278 Kbytes respectively. User text size for *gs* with the X11 server is over four times that of *gcc*.

[1]Text sizes are given for Ultrix DECstation executables. *gcc* is as built from the SPECmark distribution. The X11 server size is for `/usr/bin/Xws` on an Ultrix workstation, which includes a number of DEC extensions. The tracing experiments used a smaller server (958 KBytes). See Section 3 for details on the server used in the tracing experiments.

This research was sponsored in by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597, and by an equipment grant from Digital Equipment Corporation.

- **Three interacting contexts.** Typically, batch-oriented workloads involve two contexts: the user application and the kernel. For many of these workloads (scientific workloads are the most common examples) kernel activity is negligible. In contrast, activity in most X11 workloads is split among three contexts: the client, the X11 server, and the operating system, with significant activity occurring in all three contexts. The result is additional resource competition that does not happen in the two-context and single-context case.

- **Mandatory and potentially frequent context switches.** When multi-task workloads are used in memory system studies, they are usually created by taking unrelated batch-oriented workloads and running them simultaneously. Context switches for these multi-task workloads can often be scheduled arbitrarily. An intelligent scheduler may try to make switches infrequent, as a strategy for minimizing cache competition. In contrast, scheduler policy is irrelevant in client-server systems. Context switches are largely determined by client behavior and inter-process communication implementations. Depending on the client, context switches may be frequent.

- **A large community of users.** The X11 server and clients are used daily and repeatedly by a large contingent of the workstation computing community. Many of these users make rare use of programs such as those in the SPECmarks.

Performance for benchmarks is typically measured in terms of *throughput*, with program execution times reduced to units such as MIPS or MFLOPS. A key distinction between interactive workloads and more traditional benchmarks is their sensitivity to *latency*, which is the time required for the system to respond to a given input event. Analysis of memory system components such as caches and write buffers is common practice for throughput benchmarks [7, 8, 10]. However, interactive programs and client-server systems have received relatively little attention in recent research. [3, 20]. This is unfortunate in that, for many computer users, quick response time for latency-critical interactive applications is more important than the throughput of batch jobs. Because of the size and complexity of server-based systems such as X11, few detailed measurements of their behavior have been made. We think the problem deserves more attention, as memory system delays can have a significant impact on latency for interactive workloads.

1.1. Related Work

This research focused primarily on measuring the behavior and performance of realistic X11 client workloads from the perspective of the memory system. Several prior studies measure X11 behavior, although they differ substantially in that they consider behavior at higher levels of abstraction. Researchers at the Microelectronics Computation and Technology Corporation built a tool called XSCOPE to measure X11 performance and localize performance problems [22]. XSCOPE provides information about X11 request, reply, error, and event packets. Their experience in designing XSCOPE indicated some problems with the syntax of the X11 protocol.

Simple measures of performance, such as operations per second, are often used when characterizing new graphics hardware. Researchers at DEC WRL have done significant work in achieving good X11 performance, both with simple bit-mapped framebuffers [16] and more complicated hardware [17]. They also demonstrate software algorithms that permit effective use of the hardware. They consider memory reference behavior, but strictly as related to frame buffer references; application performance is beyond the scope their work. Researchers at Hewlett Packard used a technique called Direct Hardware Accesses (DHA) in their Starbase/X11 Merge system to enable high performance when Starbase applications access the display. [6, 5]

Several other projects consider memory system performance in a more general context, independent of X11 applications. MTOOL [11] compares execution time of program segments to predicted time for a perfect memory system. A large difference between the predicted and the measured times suggests a possible memory system performance problem. MTOOL has been applied primarily to detecting memory bottlenecks in FORTRAN programs, and is not appropriate for measuring operating system behavior. Thus, MTOOL is not appropriate for measuring X11 workloads. MTOOL has been adapted to work with shared-memory multiprocessor programs [12]. Another project, MemSpy [15], is based on the Tango [9] simulation and tracing system. To date, Tango is designed for use with parallel applications and multiprocessor systems, and has not been applied to multiprogrammed uniprocessor workloads or measurements of operating system activity. The measurements for this paper concern aggregate memory system behavior. In contrast, both MTOOL and MemSpy identify performance problems as specific segments of code (also data for MemSpy) within a workload.

The remainder of the paper is organized as follows. The next two sections describe the experiments, first giving details on the tracing and simulation systems, then a qualitative and quantitative characterization of the workloads. Next, in the analysis section, we analyze memory delays for X11 workload from three points of view: memory penalties by subsystem, cache effects, and TLB behavior. The paper closes with a brief review of our major conclusions.

2. Tracing and Simulation

The experiments for this study ran on a DECstation 5000/200, using an address tracing system developed at Carnegie Mellon University and DEC WRL [4, 7]. The tracing system uses object code rewriting [4, 24], in which original object code is augmented with instrumentation instructions such that an address trace is generated as a side effect of program execution. Traces are accurately interleaved both within a single context and across user and system contexts. Traced addresses are corrected to reflect those of the original and not the traced instruction stream. The Ultrix kernel, the X11 server, and X11 clients were all instrumented and traced.

The DECstation 5000/200 uses a 25 MHz MIPS R3000 CPU with MIPS R3010 floating point unit and MIPS R3220 memory buffer. The DS5000/200 uses a simple eight-bit frame buffer interface, in which the frame buffer appears as memory and is written directly by the processor. No special-purpose graphics hardware is used. The X11 server runs as a user process. Communication between X11 server and clients occurs through the socket interface provided by Ultrix. The frame buffer is mapped directly into the address space of the X11 server, and accesses to the frame buffer bypass the cache. This could potentially induce penalties for frame buffer reads. Fortunately such reads are relatively rare. Frame buffer writes pass through the write-buffer so their performance is unaffected. A discussion of effective software support of the DECstation 5000/200 frame buffer can be found in [16].

instruction cache: 64 KB, direct-mapped, physical, 16 byte line, 15 cycle miss penalty.

data cache: 64 KB, direct-mapped, physical, 4 byte line, write allocate, 15 cycle read miss penalty, read miss fetches 16 aligned bytes.

write buffer: six entries, page-mode writes complete in one cycle, non page-mode writes complete in five cycles; CPU reads have priority for memory access, but wait for writes that have already started. 4 KB page size for page-mode writes.

translation buffer: 64 entries, 56 random/8 wired entries, trap to software on TLB miss. Each TLB entry maps a 4 KB page.

page mapping: Deterministic. The physical page used to back a given virtual page is determined by the virtual page number and its address space identifier. The deterministic strategy prevents conflicts within any 64 KB (cache size) range of virtual addresses.

kernel memory: All kernel text and most kernel data is in unmapped, cached physical memory.

Table 2-1: Memory system simulation parameters.

We used a DECstation 5000/200 memory system simulator, along with several other simple tools, to process trace generated by the test workloads. The parameters for the memory system simulation are given in Table 2-1.

We omit several million instructions from the start and end of each simulator experiment to eliminate startup and shutdown effects. The tracing system extracts page table information from the running kernel to provide virtual to physical page mappings in the simulator. The Ultrix page allocation code attempts to assign physical pages such that virtual page orderings are preserved in the physical cache. Kernel text is not mapped.

3. Workloads

We used the standard X11R5 distribution from MIT Athena (available by anonymous ftp from `export.lcs.mit.edu`). The PEX extension[2] was omitted from the X11 server. Otherwise, we used the default server configuration. The Ultrix system was version 4.2 revision 96. Table 3-1 describes the X11 clients used in this study. All programs are written in C, and compiled with version 2.1 of the DEC/MIPS C compiler.

workload	Description	time
micro benchmarks		
destroy	window destruction, using *x11perf -repeat 5 -reps 10 -subs 10 100 -destroy*	2.6
resize	window resize, using *x11perf -repeat 2 -reps 5 -subs 10 100 -resize*	2.5
circulate	window circulate operations, using *x11perf -repeat 2 reps 5 subs 10 100 -circulate*	2.8
ftext	text painting, using *x11perf -repeat 5 reps 500 -ftext*	2.4
copy	bitmap copy, using *x11perf -repeat 5 reps 250 -copywinwin100*	11.4
scroll	window scrolling, using *x11perf -repeat 2 reps 250 -scroll500*	23.3
X11 clients		
splot	Splot is run four times on four different input files. Total size of splot input is 94 Kbytes.	12.4
gs	Ghostscript is used to preview a twenty page conference paper. Input file size is 251 Kbytes.	25.9
Other workloads		
gcc	The GNU C compiler converts a 17K (preprocessed) source file into optimized Sun-3 assembly code. Not an X11 client.	3.7
compress	Data compression using Lempel-Ziv encoding. A 100K file is compressed then uncompressed.	1.3

Table 3-1: Experimental workloads. Times are in seconds.

X11perf is a client in the X11R5 distribution. It measures the time to repeat a given server operation some number of times, and is commonly used as a gauge of X11 server performance. All the microbenchmarks for this paper are runs of *x11perf* with different input parameters. *Splot* is a program for generating plots for PostScript and X11. It is available by anonymous ftp from `labrea.stanford.edu`. *Ghostscript*, by

[2]PEX is the PHIGS extension to X11, used for three-dimensional graphics.

Aladdin Enterprises, is an X11 previewer for Adobe Systems' PostScript language. It is distributed with the GNU General Public license, available by anonymous ftp from `athena-dist.mit.edu`. Version 2.6.1 of *gs* was used.

	instruction reads					data reads				data writes			
workload	idle	non-idle	%sys	%Xs	%Xc	non-idle	%sys	%Xs	%Xc	non-idle	%sys	%Xs	%Xc
destroy	0	33999	5.2	91.7	3.1	7128	5.5	91.1	3.4	4529	7.1	87.9	5.1
resize	0	35999	2.6	96.9	0.5	7382	2.2	97.2	0.6	2759	2.8	96.1	1.1
circulate	0	45999	2.9	96.7	0.4	9958	2.2	97.4	0.4	4284	3.4	95.9	0.6
ftext	0	62000	2.3	96.1	1.7	7921	4.2	92.5	3.3	4827	4.9	92.0	3.1
copy	0	92000	3.8	95.6	0.6	15912	3.4	95.5	1.1	14457	1.4	97.7	1.0
scroll	0	105999	2.0	97.9	0.1	32528	1.2	98.7	0.0	31740	0.5	99.5	0.0
splot	6910	148619	33.4	36.0	30.6	30852	32.2	34.9	32.9	16346	41.6	32.7	25.7
gs	3992	448018	10.0	30.8	59.2	93584	11.0	24.6	64.3	50994	14.6	33.4	52.0
gcc	63684	28899	21.0	0.0	79.0	5716	20.5	0.0	79.5	3810	28.6	0.0	71.4
compress	5555	16934	19.2	0.0	80.8	3249	18.9	0.0	81.1	2225	29.2	0.0	70.8

Table 3-2: Instruction and Data Reference Counts

This table shows event counts for each workload, along with the percentage contribution from the system (%sys), X11 server (%Xs), and X11 client (%Xc). The first column for each workload shows the number of idle instructions executed during that workload. All other counts and percentages are for non-idle events. All counts are in thousands.

Two workloads which are not X11 clients, *gcc* and *compress*, are included for comparison between X11 clients and other workloads. Among common integer benchmarks, *compress* presents an above average demand on the memory system, while *gcc*'s demands are extreme. More details on the memory system behavior of these workloads can be found in [7].

Table 3-2 gives reference counts for the experimental workloads. The low percentage of kernel instructions for the microbenchmarks demonstrates that many types of X11 operations require relatively little kernel activity. Higher levels of kernel activity in *splot* and *gs* are attributed (at least partially) to the file I/O that these workloads require.

4. Experiments and Analysis

4.1. Memory Cycles Per Instruction

We use data from our simulator to calculate *memory cycles per instruction*, (*MCPI*), which is the total number of CPU stall cycles due to the memory system divided by the total number of instructions executed. *MCPI* is one of several components of cycles per instruction (*CPI*), a metric commonly used to evaluate computer systems [14]. Other components of *CPI* that are not reflected in *MCPI* include one cycle per instruction for instruction execution by the processor, cycles during interlocked multiply, divide, and floating point operations, and no-ops inserted by the compiler for load and branch delays. The other components of *CPI* remain relatively constant as processor cycle time decreases. In contrast, *MCPI* is a function of the ratio of memory speed to processor speed, and is less dependent of processor architecture. *MCPI* will dominate overall *CPI* if current trends in processor and memory speed continue. *MCPI*+1 is a lower bound and a good estimate of overall *CPI* for workloads (such as operating systems) in which multiplies, divides and floating-point operations are rare.

Figure 4-1 illustrates *MCPI* for the experimental workloads. For comparison, *MCPI* measurements for *gcc* and *compress* are also shown. Each bar is shaded to denote different *MCPI* components. For visual clarity, the system and user contributions are separated by a vertical bar. Data and instruction cache misses in user and system mode are only partially responsible for the total *MCPI*. Other components include CPU write-stalls and uncached memory reads. CPU write-stalls are reflected in the *wbuffer* component which show the average per-instruction penalty from writes to a full write-buffer, and reads during the completion of a five-cycle write.

TLB misses are not included explicitly in the baseline *MCPI*. Their cost appears as additional instructions executed and additional data references, included in the total counts.

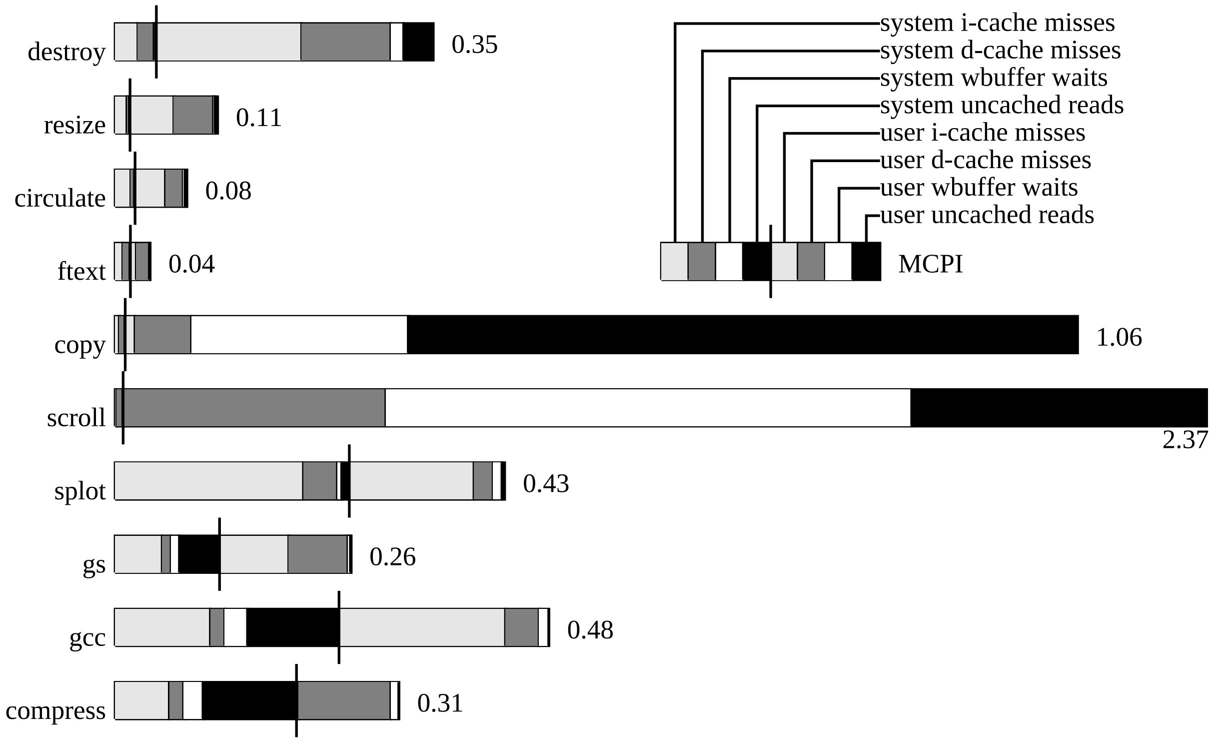

Figure 4-1: Baseline *MCPI*.

Each horizontal bar represents total *MCPI* for a given experimental workload, broken down between system/user and various components of the memory system. Contribution from system and user activity are separated by a vertical line. User activity includes X11 server and X11 client. The number at the right of each bar is the *MCPI* for that workload. Startup and shutdown effects were excluded by omitting several million instructions at the beginning and end of each simulation experiment. *MCPI* for *gcc* and *compress* are included for comparison with workloads that are not X11 clients.

As can be seen from Table 3-2, operating system overhead is low for the six microbenchmarks. This is reflected in Figure 4-1 as low system *MCPI*. With the X11 server accessing the frame buffer directly, kernel activity during the microbenchmarks is dominated by TLB faults and socket communication, both of which are relatively inexpensive as compared to the activity of disk I/O intensive workloads. In contrast, user-level *MCPI* varies significantly across the microbenchmarks, and is dependent on server activity. The worst behavior by far occurs for *copy* and *scroll*, which incur substantial write-buffer and uncached-read penalties due to a high density of frame-buffer references.

Comparing system behavior for the microbenchmarks to that of *splot* and *gs*, there is substantial additional system overhead for the realistic benchmarks. This reflects greater variation in system activity due to the addition of file I/O, and is consistent with behavior observed for system-intensive and I/O intensive applications such as *gcc*.

Turning to user-level overhead, Figure 4-1 shows that many X11 clients have significant user instruction cache *MCPI* contributions, sometimes higher than system i-cache *MCPI* contributions. This is unusual for integer workloads[3] [7]. In the next section we discuss how competition within and between address spaces contributes to poor instruction cache behavior, for both system and user.

[3] *gcc* is exceptional in this respect.

Penalties from the write-buffer and uncached memory reads appear problematic in microbenchmarks but aren't significant in more realistic workloads. For frame-buffer writes, the X11 server benefits from the combination of write-buffer and writes through the uncached segment. Together they permit frame buffer writes to proceed at top speed without disturbing the contents of the data cache.

4.2. Cache Effects

We consider two types of cache misses:

- *Inter-Context Competition* occurs when references from two or more active address spaces displace each other in the cache. Client-server systems such as X11 introduce a user level server context, in addition to the application and system context of a workload such as *gcc*. The additional server context could induce more competition in the cache.
- *Self-Interference* misses occur when two active instructions in the same address space collide in the cache. The Ultrix page-mapping algorithm ensures that self-interference misses will not occur within an address space if a program's text is smaller than the cache size. Many of the SPECmarks have text smaller than the cache, but the text of X11 server, *gs* and *splot* are all much larger. With localities spread throughout large text, self-interference misses are more likely to occur.

We discuss inter-context competition first.

4.2.1. Inter-Context Competition

An X11 workload is composed of three contexts: X11 client, X11 server, and the operating system. Inter-context competition occurs when two or more contexts compete for memory resources (such as cache lines), causing degraded performance. To simulate a system without competition, we ran experiments with memory reference trace from only one source (eg. use trace from the kernel only), then summed events over all three runs. The sum represents the behavior of a hypothetical machine where each context has its own private memory system, hence a system where all competition has been eliminated. Figure 4-2 compares *MCPI* for several workloads with and without competition.

Figure 4-2 shows that, for realistic X11 workloads, inter-context competition has a large impact on overall *MCPI*. For both *splot* and *gs*, competition is responsible for a large proportion of system i-cache misses. Additionally, eliminating competition from *splot* causes a drastic reduction in user i-cache miss rates. The instruction cache requirements of *destroy* microbenchmark are modest, so eliminating competition has little effect. Similarly, compress shows little benefit when competition is eliminated.

We compared missing instruction addresses in the X11 server to counts of kernel instruction references for a run of *splot* to identify probable inter-context conflicts. An example of such a conflict was between the Ultrix general exception handler, `exception()`, and the X11 server memory allocation routine, `malloc()`. Both routines are called frequently, and are located in such a way that they overlap on a 4 Kbyte memory page. Kernel text pages are not mapped, so `exception()` will always be located in the same place in the cache. User text is mapped, so the location of `malloc()` in the cache will depend on the virtual-to-physical page assignment. In Ultrix the page assignment is a function of virtual page number and process ID. Given the page-mapping policy and cache parameters for this memory system, the probability that the above conflict will occur is 1/16. Many such conflicts were identified for the *splot* run. The large working sets of X11 workloads, combined with frequent mandatory context switches, makes competition misses more likely.

These results confirm earlier research on cache competition, which demonstrated significant penalties for warming up the cache after a context switch [19]. The earlier study found competition to be important in multitasking and compute-bound workloads, but a non-issue in the client-server workload they tested. However, the earlier study did not include system behavior, and it used a synthetic client-server workload dominated by communication, a workload more similar to the microbenchmarks used for this study than the realistic workloads. Our work complements the prior study by including system effects in the memory reference stream, and by demonstrating that inter-context competition can also have impact for client-server workloads.

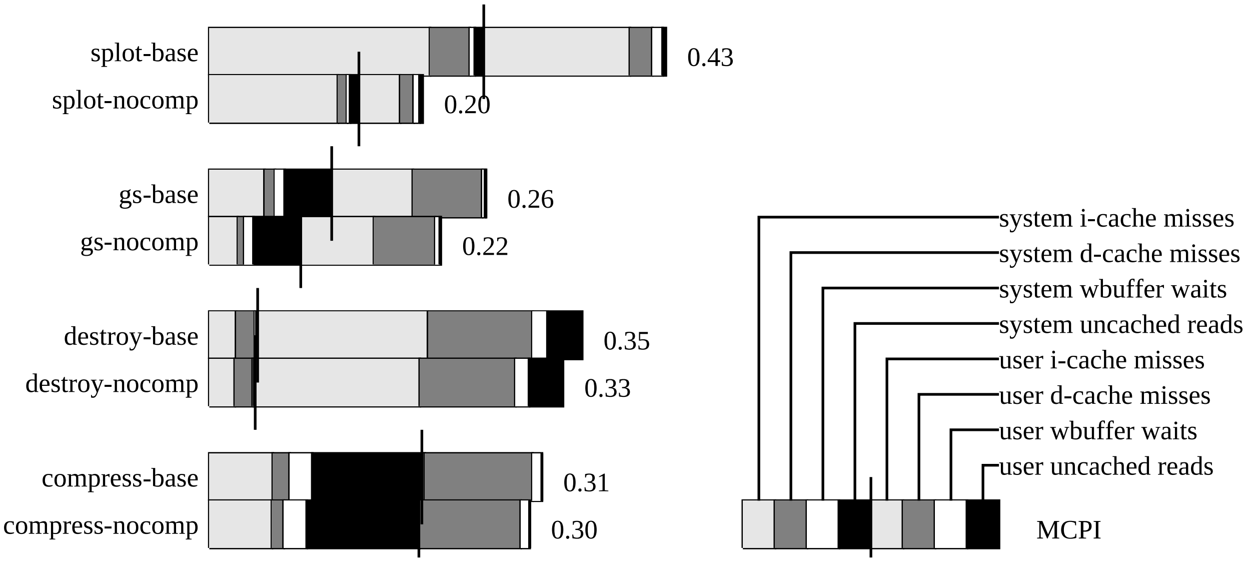

Figure 4-2: *MCPI* with and without inter-context competition.

This figure shows the effect of inter-context competition on memory system performance. Each horizontal bar represents total *MCPI* for a given experimental run, as in Figure 4-1. We show two bars for each workload, the upper corresponding to the base system, and the lower representing a system in which competition between address spaces has been eliminated by giving each address space a private memory system. This figure shows that inter-context competition has major impact on instruction cache behavior for realistic X11 workloads.

4.2.2. Self-Interference misses

We measured the impact of self-interference misses by replacing the direct-mapped caches in the simulator with two-way set associative caches of the same size. Figure 4-3 illustrates the combined effects of competition and associativity on *MCPI* for *splot* and *gs*. To isolate the impact of self-interference misses from that of competition misses, compare runs where all competition misses are eliminated (*bench*-nocomp and *bench*-a+nocomp). In this comparison, all misses eliminated by associativity are from conflicts within an address space. For both *gs* and *splot*, associativity eliminates a significant number of self-interference misses.

Figure 4-3 also shows the effect of associativity on inter-context competition. The two-way set associative caches eliminate some inter-context competition, but not all (compare *bench*-assoc and *bench*-a+nocomp).

4.2.3. Summary

For the two realistic X11 workloads we consider, both inter-context competition and self-interference misses have significant impact on memory system behavior, with the most significant effects occurring in the instruction cache. Strategies have been described [18] for avoiding text conflicts within an address space, but it is difficult to envision a practical software system to avoid competition between address spaces. The problem could potentially be addressed in hardware with cache associativity, although this could have an impact on machine cycle time. Many current systems rely on good luck to avoid inter-context competition. As the gap between CPU and memory speed grows, and as users demand improved performance for interactive and multi-address space systems, more aggressive cache designs may be required. The lack of client/server workloads in standard benchmark suites could lead hardware developers to believe that inter-context competition is a non-issue. Our measurements suggest it deserves more attention.

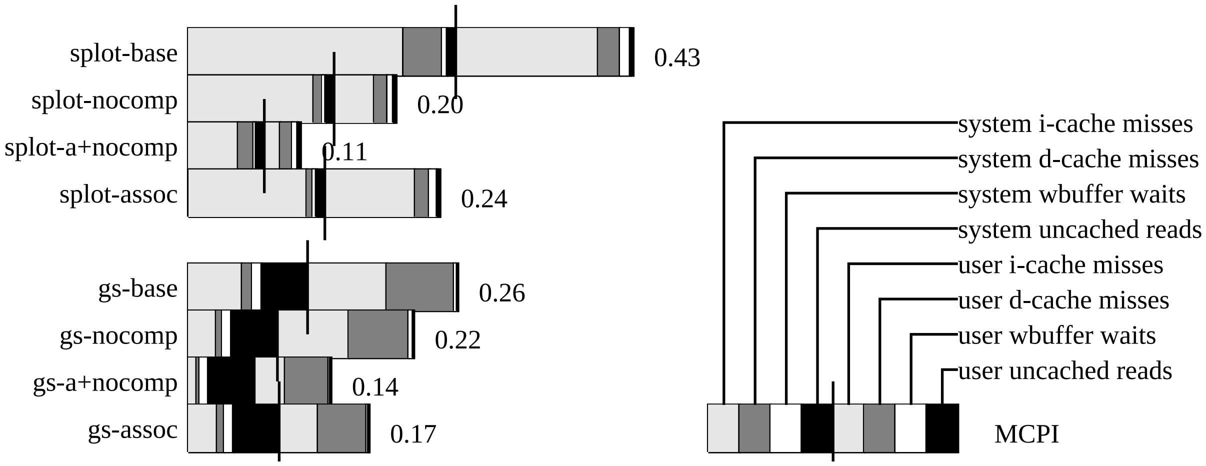

Figure 4-3: Inter-Context Competition, Associativity, and *MCPI*.

This figure shows the effect competition and self-interference misses on memory system performance. Each horizontal bar represents total *MCPI* for a given experimental run, as in Figure 4-1. We show four bars for each workload: the base system (*bench*-base), without competition (*bench*-nocomp), with associativity and without competition (*bench*-a+nocomp), and with associativity (*bench*-assoc). This figure shows that both competition and self-interference misses have significant impact on memory system behavior for realistic X11 workloads, particularly in the instruction cache.

4.3. TLB Behavior

In this section we consider the TLB behavior of the realistic X11 workloads. Table 4-1 shows TLB miss data for *splot*, *gs*, and *gcc*. Compared to other integer workloads, X11 applications have poor TLB behavior. Both *splot* and *gs* show significantly higher miss rates than *gcc*, which is relatively demanding among integer workloads [7]. Three phenomena contribute to increased TLB miss rates:

- The X11 server needs over 200 page mappings to address the entire frame buffer. Any operation that paints a significant part of the screen will tend to flush the TLB.
- The X11 server, *gs* and *splot* all have relatively large program text. This increases the likelihood that localities will be spread across multiple text pages, which in turn increases the demand on TLB resources.
- X11 applications involve two interacting user contexts, as opposed to one context for *gcc*. Multiple contexts mean more fragmentation and increased competition for limited TLB resources.

workload	user	system
splot	1.58	0.28
gs	1.49	0.14
gcc	1.11	0.07

Table 4-1: TLB Misses per 1000 instructions.

System TLB misses include misses to both user and system segments.

During the run of *splot*, 280000 user TLB misses occurred. There were about 680000 during *gs*. Estimating 20 cycles to service a TLB miss [21], the penalty for TLB faults is less than 0.04 CPI for both workloads. Thus, the impact on overall performance is not significant for the memory system we simulated.

The impact of the TLB on overall performance is dependent on the performance balance of memory system components. Processors such as the DEC Alpha 21064 [2] rely on fewer TLB entries with larger and oversized pages to achieve good TLB behavior. If software systems such as X11 don't make good use of these new features, miss rates will go up, increasing the impact of the TLB on overall performance. Also, the penalty for a single TLB miss could increase. An earlier study used 100 cycles as an estimate of the TLB miss penalty for a futuristic machine [8]. As the balance of TLB to cache resources changes, TLB performance could become an important issue. X11 workloads require more TLB resources than popular benchmarks such as *gcc*. If computer systems are designed to optimize the performance of the popular benchmarks, systems such as X11 can be expected to suffer.

Our measurements show degraded TLB behavior for X11 workloads as compared to other integer codes. Researchers at the University of Michigan [21] measured similar TLB behavior for the Mach 3.0 operating system [1, 13], another system where a user-level server contributes significantly to overall activity. The two independent studies suggest a broader conclusion, that page behavior for user-level client/server systems induces substantially elevated TLB miss rates.

As a final note, competition also affects TLB behavior. In our measurements, competition accounted for 60% of user TLB misses in *splot* and 30% of user TLB misses in *gs*. Note that additional associativity can't help here, as the DECstation 5000/200 TLB is already fully associative.

5. Conclusions

Memory system behavior for X11 workloads differs significantly from the batch-mode programs typically used in memory system studies. With large program text, a large mapped frame buffer, and multiple competing contexts, they can present a far greater load on instruction caches and TLBs than typical throughput benchmarks.

Cache associativity and TLB size are sensitive issues for hardware designers. For many machines, increasing these parameters would have direct impact on machine cycle time, the principle metric driving performance improvements in microprocessors. As machine cycle time dominates performance for many current benchmarks, there is a potential conflict between high throughput for benchmarks, and low latency for large client/server systems. Optimal throughput and optimal latency may not be possible in the same machine.

Our measurements are specific to X11 clients and server, but similar behavior can be expected in other client/server systems. Instruction cache and TLB behavior is aggravated by large user text, frequent and mandatory context switches, and multiple active contexts. Any system with these characteristic will probably have similar behavior. In systems with multiple heavy-weight servers, contention for memory system resources will be even more intense.

6. Acknowledgements

Thanks to Norm Jouppi who first suggested looking at X11. Brian Bershad and Doug Tygar made useful comments on the text. Joel McCormick answered questions about X11. David Wall provided the initial version of *epoxie*. The design of the tracing system is based on prior work by Anita Borg, and her contributions continued throughout this project. Thanks also to Digital Equipment Corporation for their generous support.

References

1. Michael J. Accetta, Robert V. Baron, William Bolosky, David B. Golub, Richard F. Rashid, Avadis Tevanian, Jr., and Michael W. Young. Mach: A New Kernel Foundation for Unix Development. Proceedings of the Summer 1986 USENIX Conference, July, 1986, pp. 93-113.

2. Digital Equipment Corporation. Digital's 21064 Microprocessor. Data sheet.

3. Brian N. Bershad, Richard P. Draves, and Alessandro Forin. Using Microbenchmarks to Evaluate System Performance. The Proceedings of the Third Workshop on Workstation Operating Systems, April, 1992, pp. 148-153.

4. Anita Borg, R.E. Kessler, Georgia Lazana, and David Wall. Long Address Traces from RISC Machines: Generation and Analysis. WRL Research Report 89/14, Digital Equipment Corporation Western Research Laboratory, 1989.

5. J.R. Boyton, S.L. Chakrabarti, S.P. Hiebert, J.J. Lang, J.R. Owen, K.A. Marchington, P.R. Robinson, M.H. Stroyan, J.A. Waitz. "Sharing access to display resources in the Starbase/X11 Merge". *Hewlett Packard Journal 40*, 6 (December 1989), 20-32.

6. Kenneth H. Bronstein, David J. Sweetser, and William R. Yoder. "System design for compatibility of a high-performance graphics library and the X Window System.". *Hewlett Packard Journal 40*, 6 (December 1989), 6-10. .

7. J. Bradley Chen and Brian N. Bershad. The Impact of Operating System Structure on Memory System Performance. Proceedings of the 14th ACM Symposium on Operating System Principles, December, 1993.

8. J. Bradley Chen, Anita Borg, and Norman P. Jouppi. A Simulation Based Study of TLB Performance. The Proceedings of the 19th Annual International Symposium on Computer Architecture, May, 1992, pp. 114-123.

9. H. Davis, S.R. Goldschmidt, and J. Hennessy. Tango: A Multiprocessor Simulation and Tracing System. Proceedings of the International Conference on Parallel Processing, August, 1991, pp. 99-107.

10. Jeffrey D. Gee, Mark D. Hill, Dionisios N. Pnevmatikatos, and Alan Jay Smith. Cache Performance of the SPEC Benchmark Suite. University of Wisconsin-Madison, 1991.

11. Aaron Goldberg and John Hennessy. MTOOL: A Method for Detecting Memory Bottlenecks. WRL Technical Note TN-17, Digital Equipment Corporation Western Research Laboratory, 1990.

12. Aaron J. Goldberg and John L. Hennessy. "MTOOL: An Integrated System for Performance Debugging Shared Memory Multiprocessor Applications". *IEEE Transactions on Parallel Processing 4*, 1 (January 1993), 28-40.

13. David Golub, Randall Dean, Alessandro Forin and Richard Rashid. UNIX as an Application Program. Proceedings of the Summer 1990 USENIX Conference, June, 1990, pp. 87-95.

14. John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, Palo Alto, CA, 1990.

15. Margaret Martonosi, Anoop Gupta, and Thomas Anderson. MemSpy, Analyzing Memory System Bottlenecks in Programs. Proceedings of the 1992 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, June, 1992, pp. 1-12.

16. Joel McCormack. Writing Fast X Servers for Dumb Color Frame Buffers. WRL Research Report 91/1, Digital Equipment Corporation Western Research Laboratory, 1991.

17. Joel McCormack and Bob McNamara. A Smart Frame Buffer. WRL Research Report 93/1, Digital Equipment Corporation Western Research Laboratory, 1993.

18. Scott McFarling. Program Optimization for Instruction Caches. The Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems, April, 1989, pp. 183-191.

19. Jeffrey C. Mogul and Anita Borg. The Effect of Context Switches on Cache Performance. The Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April, 1991, pp. 75-84.

20. Jeffrey C. Mogul. SPECmarks are leading us astray. The Third Workshop on Workstation Operating Systems, April, 1992, pp. 160-161.

21. David Nagle, Richard Uhlig, Tim Stanley, Stuart Sechrest, Trevor Mudge and Richard Brown. Design Tradeoffs for Software-Managed TLBs. Proceedings of the 20th Annual International Symposium on Computer Architecture, May, 1993, pp. 27-38.

22. J.L. Peterson. XSCOPE: a debugging and PERFORMANCE tool for X11. Proceedings of the IFIP 11th World Computer Congress, September, 1989, pp. 49-54.

23. *SPEC Benchmark Suite Release 1.0.* System Performance Evaluation Cooperative, 1989.

24. David W. Wall. Systems for Late Code Modification. In *Code Generation --- Concepts, Tools, Techniques*, Springer-Verlag, 1992, pp. 275-293.

J. Bradley Chen is presently completing a PhD in Computer Science at Carnegie Mellon University. His interests include operating systems, memory systems, and issues relating to the design and performance of large software systems. He received BS and MS degrees in 1987 from Stanford University.

Table of Contents

List of Figures

List of Tables