

The Inaccuracy of Trace-Driven Simulation Using Incomplete Multiprogramming Trace Data

J. Kelly Flanagan
Performance Evaluation Lab
Computer Science
Brigham Young University
Provo, UT 84602

Brent E. Nelson
James K Archibald
Electrical and Computer Eng.
Brigham Young University
Provo, UT 84602

Greg Thompson
Po Box 58119
Intel Corp.
Santa Clara, CA 95052

Abstract

Trace-driven simulation is commonly used to predict the performance of computer systems. However, existing tracing techniques produce traces inadequate for some studies: they do not usually record operating system references, and they produce relatively short traces. This paper explores the impact of these trace distortions on the performance estimates of uniprocessor memory hierarchies using multiprogramming workloads. We used a hardware monitor to capture traces under a variety of workloads and operating systems. Our monitor captures every reference and can record arbitrarily long traces. We quantify memory hierarchy performance using traces of the SPEC SDM1.1 benchmark suite executing on an i486 CPU. To evaluate variations due to operating systems, we compare these results under both Mach 3.0 and UNIX Sys V R4. We conclude that for current uniprocessors, long but incomplete traces result in modest errors in estimated performance, but for proposed architectures with large delays to main memory, the errors can be significant.

1 Introduction

Simulation is an indispensable tool in the design of any computer system and can take many forms. One common example is trace-driven simulation used in the design of memory hierarchies. With this method, a software model of the caches and memory modules being simulated is driven with a trace or stream of memory references to determine cache miss-rates, bus traffic levels, or effective memory access times.

The software model used in trace-driven simulation is typically under the control of the designer – given enough time it can be created with whatever level of detail is needed to obtain results with the desired precision. For example, it may accurately model the timing of the memory hierarchy, or it may simply count cache misses. In contrast, the designer often has little control over the accuracy or completeness of the input trace data. The difficulties inherent in trace collection are well known [1, 2, 3], and the development of better tracing methods has been the goal of much previous work.

For memory hierarchy studies it is desirable to use long address traces that are accurate representations

of real multiprogramming workloads. Such traces have not been generally available to the research community, and those traces that are available are usually short, lack operating system references, and contain distortions due to the trace collection method employed.

Our previous work examining the impact of trace distortions on simulation results used trace data collected from small single process benchmarks [4]. For this work we use publically available traces of the SPEC SDM benchmark suite that we collected. These benchmarks have working sets ranging in size from 4-9MBytes and contain significant multitasking behavior compared to the commonly used SPECint and SPECfp benchmarks. Under these workloads the operating system contributes nearly 70% of the cache misses. We demonstrate that miss-rates obtained using complete trace data are higher for Mach than for UNIX System V R4. In addition, we find that for small main-memory access delays, the error in effective memory access time prediction caused by using incomplete trace data is minimal. As main memory access times increase in relation to CPU cycle times, the errors become significant. Finally, these results confirm most of the findings of three significant papers [5, 6, 3].

The remainder of this paper is organized in five sections. In Section 2 we outline common trace gathering methodologies and the distortions they introduce into trace data. In addition, we describe our trace collection technique and its operation. In Section 3 we describe the workstation we collected traces from, our multiprogramming workloads, and the resulting trace data. In Section 4 we quantify the cache simulation miss-rate errors that result from the use of trace data containing only user references. We then discuss in Section 5 how effective access time predictions are affected by inaccurate trace data. Finally, we present conclusions in Section 6.

2 Trace collection techniques

This section describes various trace-gathering techniques and discusses the trace distortions they introduce. In addition, the trace gathering technique we use is briefly described.

2.1 Common trace-gathering techniques

There are five common trace-gathering techniques:

- Instruction modification or inlining [2].
- Microcode modification [1]
- Single stepping
- Processor simulation [7].
- Hardware monitors [6, 8, 9, 10, 11].

Each of these trace-gathering techniques introduces at least one kind of error or distortion into trace data. The four most common trace distortions are: missing workload components, absence of multitasking behavior, time dilation, and short traces. These errors result in traces that are not representative of the stream of references generated in a real system environment. The differences between the stream of references in a real system and those generated by previous tracing techniques introduce errors in trace-driven simulation results.

Table 1 summarizes the advantages and disadvantages of each tracing technique and illustrates the common distortions associated with them. There are noted exceptions to the general trace characteristics associated with each technique. For example, the instruction modification technique developed in [2, 3] includes operating system and multitasking behavior, and some implementations of single stepping methods include nearly all the operating system references. In addition, to acquire a complete address trace using a hardware monitor it is necessary to disable any internal CPU caches. This hinders system performance and is another form of time dilation. The trace length limitation associated with hardware monitors is due to the difficulty and expense of constructing a buffer that is both fast and deep. Although each trace-gathering technique generates useful data, all techniques produce traces with at least one type of distortion. Simulation results obtained using distorted traces may be incorrect. These inaccuracies may not change general trends, but they impact design decisions where accurate system performance predictions are necessary. In the next section we briefly describe BACH, a hardware monitor used to collect the trace data used in this study.

2.2 BACH operation and organization

In this section, the BACH mechanism is described in general terms, since it can be used to trace a variety of computer systems. BACH has been used to trace an i486, SPARC 1+, and a Motorola 68030 based UNIX workstation [10, 11]. We have developed several techniques that provide BACH with the advantages of a hardware monitoring technique, with few of the disadvantages.

A typical tracing setup consists of the machine being traced (TRACEE), BACH, and an extractor (EXTRACT). When enabled, BACH monitors TRACEE’s CPU pins, storing desired signal values in an internal buffer. The collected signals usually include the processor’s address and data pins, control pins, and the number of cycles between references. When the buffer fills, EXTRACT downloads the trace for storage or processing.

Method	Trace Characteristics				
	OS	MT	TD	TL	V/P
SCI	no ¹	no ¹	10:1	∞	V
ECI	no ¹	no ¹	10:1	∞	V
OCI	no ¹	yes	10:1	∞	V
MM	yes	yes	10:1	400 K	V/P
SS	no ¹	no ¹	100:1	∞	V
PS	no	no	1000:1	∞	V/P
HM	yes	yes	1:1 ¹	1 M ¹	V/P

SCI	Source code inlining
ECI	Executable code inlining
OCI	Object code inlining
MM	Microcode modification
SS	Single stepping
PS	Processor simulator
HM	Hardware monitors
OS	Are operating system references present?
MT	Is multitasking activity present?
TD	Time dilation
TL	Length of trace segments
V/P	Are address references physical or virtual?

Table 1: Summary of previous tracing techniques. A marked (¹) item indicates that a particular implementation of the associated technique partially or completely includes this feature.

We have devised a technique that overcomes the trace length limitation of previous hardware monitors. When BACH’s internal buffer is nearly full it sends a low priority interrupt to TRACEE. The low priority interrupt enables TRACEE to finish servicing higher priority interrupts before responding to BACH. After all other devices have been serviced, TRACEE enters the BACH interrupt routine, which disables interrupts and spins in a tight loop. While TRACEE spins in the interrupt loop, BACH’s internal buffer is emptied by EXTRACT. The buffer contents may be stored to secondary storage media or processed while being extracted. When the buffer is emptied, EXTRACT signals TRACEE to continue execution. This process may be repeated as many times as desired, producing a contiguous trace. Tracing is completed when TRACEE disables tracing. A complete description of BACH can be found in [10, 11].

In summary, trace driven simulation is important for modeling computer systems. Traces gathered using the methodologies discussed in Section 2.1 suffer from a variety of errors and dilations. These trace perturbations introduce errors in uniprocessor cache simulation results. To quantify the errors present in simulation results, we have constructed a hardware monitor that overcomes most of the limitations of previous hardware monitors. Traces from this system lack many of the trace distortions of other methods. In later sections we use our traces – complete and accurate representations of system activity – to quantify the trace-driven simulation errors which result from the use of incomplete and distorted traces similar to those generally available in the research community.

3 Hardware, benchmarks, and traces

We chose to trace hardware and software platforms that are well understood and available to other researchers. We traced an i486-based personal computer with an ISA bus, 16MB of RAM, and a 64KB second-level cache. It contained an ESDI disk controller with a 300MB hard disk. For the work described in this paper we traced UNIX SVR4 and Mach 3.0. This was done not only to obtain more than one data point for our work but to also allow comparisons to be made between the monolithic UNIX kernel and the Mach micro-kernel.

The SDET and KENBUS multiprogramming benchmarks from the SPEC SDM1.1 suite were used for this work. SDET and KENBUS are system-level benchmarks designed to measure overall system performance as well as the specific performance of system components such as the CPU, FPU, I/O bus, operating system, and compilers. They contain an internal driver, term scripts, and clone directories. The internal driver forks processes, each of which emulates a different user in a software development environment by executing term scripts of UNIX commands. The benchmark measures the total wall clock time elapsed for all simulated users to complete all tasks.

Each term script emulates a user typing commands at a terminal. SDET makes no attempt to model the typing speed of each user, while KENBUS does. Each term script is a random collection of common UNIX commands, such as cc, as, ls, cat, grep, nroff, mv, cd, and mkdir.

We instrumented both benchmarks to enable the hardware tracer at the beginning of the actual benchmark run. Initialization tasks such as the creation of temporary work directories were completed before tracing was enabled. The traces were collected

Name	Refs (billions)	%OS	Context Switches
SYSV-KENBUS	1.07	41.0%	3,623
SYSV-SDET	1.04	42.5%	2,993
MACH-KENBUS	1.04	45.1%	79,791
MACH-SDET	1.03	37.4%	78,421

Table 2: Characteristics of traces selected for this research. The percentage of operating system references in the Mach 3.0 environment reflect only the kernel references, not the combination of kernel and UNIX server. Each trace represents approximately 200 seconds of normal execution.

immediately following the completion of initialization tasks until the storage media was full, usually about one billion references. Four traces were used in this study: SYSV-KENBUS, SYSV-SDET, MACH-KENBUS, and MACH-SDET, as summarized in Table 2. It is interesting to note that the SYSV-SDET workload completed in 240 seconds during normal operation on the i486 used for this study. Our SYSV-SDET trace is 222 seconds long, or 93% of the entire execution of the benchmark.

In order to understand the variability present in the trace data, we collected traces from repeated runs of each benchmark and performed the analyses described below on each. Overall, the results from different runs were very consistent, and the data presented in this paper is a representative sample.

The user-only traces used in our study were created from our complete traces by removing all the operating system references. For Mach user-only traces, the UNIX server references were removed in addition to kernel references. As a result, the user-only traces contain all user references and user process interleavings and are similar to the user-only traces described in [2].

For this work we used all four traces from Table 2. In the interest of space, only the results from the SDET traces are presented. In all cases, the KENBUS analyses confirm the SDET results.

The caches used for this study are unified data and instruction caches ranging in size from 4-Kbytes to 256-Kbytes, with associativities from 1 to 16. The line size was held constant at 16 bytes. The cache simulation model implemented a write-back, allocate-on-write miss policy and used an LRU replacement algorithm.

4 Incomplete traces and miss-rates

In this section we use complete traces in addition to user-only and system-only components to test the validity of uniprocessor cache simulation results.

4.1 User-only vs. system-only miss-rates

Size	Cache Associativity				
	1	2	4	8	16
OS					
4K	14.89	12.70	12.37	12.14	12.03
8K	11.36	9.46	8.69	8.46	8.58
16K	8.21	6.65	5.39	4.68	4.48
32K	5.60	3.97	3.38	2.97	2.88
64K	3.92	2.48	1.94	1.75	1.66
128K	2.30	1.46	1.13	1.00	0.96
256K	1.31	0.81	0.64	0.58	0.55
user					
4K	6.32	4.59	3.76	3.52	3.46
8K	4.20	2.82	2.26	2.07	2.01
16K	2.60	1.67	1.24	1.14	1.09
32K	1.55	0.83	0.59	0.46	0.43
64K	0.95	0.41	0.28	0.24	0.23
128K	0.51	0.22	0.15	0.14	0.13
256K	0.29	0.11	0.08	0.08	0.07

Table 3: System-only and user-only cache miss-rate percentages for various cache configurations having 16 byte lines using the SYSV-SDET trace

Average miss-rate is a common cache performance metric. Tables 3 and 4 illustrate the difference in miss-rate for user code and operating system code for the SYSV-SDET and MACH-SDET traces. System-only miss-rates are two to nine times higher than the user-only miss-rates for both traces. In both cases, supervi-

sor code doesn't benefit as much from increased associativity as user code. This is surprising, because one of the findings in [3], based on the SPEC92 benchmarks, is that system instruction references benefit significantly from an increase in associativity. For the SDM benchmarks this does not seem to be the case. Note that the user-only miss-rates for both environ-

Size	Cache Associativity				
	1	2	4	8	16
OS					
4K	13.64	12.11	11.75	11.60	11.52
8K	10.82	9.72	9.62	9.52	9.39
16K	8.12	6.98	6.67	6.51	6.36
32K	5.80	4.53	4.06	3.80	3.68
64K	3.87	2.90	2.46	2.35	2.29
128K	2.41	1.70	1.36	1.22	1.13
256K	1.40	0.87	0.68	0.59	0.55
user					
4K	5.95	4.21	3.73	3.48	3.39
8K	3.85	2.54	2.08	1.94	1.90
16K	2.42	1.36	1.04	0.94	0.89
32K	1.31	0.74	0.55	0.48	0.45
64K	0.82	0.41	0.31	0.27	0.26
128K	0.46	0.24	0.19	0.18	0.17
256K	0.28	0.15	0.13	0.12	0.12

Table 4: System-only and user-only miss-rate percentages for various cache configurations having 16 byte lines using the MACH-SDET trace. For the Mach traces we define system references to be those generated by the Mach kernel or the UNIX server.

ments are similar. This is not surprising, since the applications being traced are otherwise identical. The slight variation that does occur is probably the result of different implementations of the system utilities. A second observation is that the system miss-rates for Mach are higher than those for UNIX for all but the smallest caches. This is due to the fact that the Mach kernel services interrupts or UNIX system calls using a separate UNIX server process. These transitions between the Mach kernel and UNIX server lower the locality of the system references, thus increasing the cache miss-rate. This is consistent with the findings in [3].

4.2 Comparison to other studies

Agarwal et al. [5] proposed a classification of cache misses useful in evaluating a memory hierarchy in a multiprogramming environment. They described three categories of cache misses: (i) misses caused by user references, (ii) misses caused by the operating system, and (iii) misses caused by the interference or collisions in the cache between user and operating system references. The top band in Figures 1 and 2 shows the fraction of misses in our traces due to these collisions.

Figure 1 and Table 2 show that user references contribute less than 20% of the misses directly and another 10% through user-collision misses despite mak-

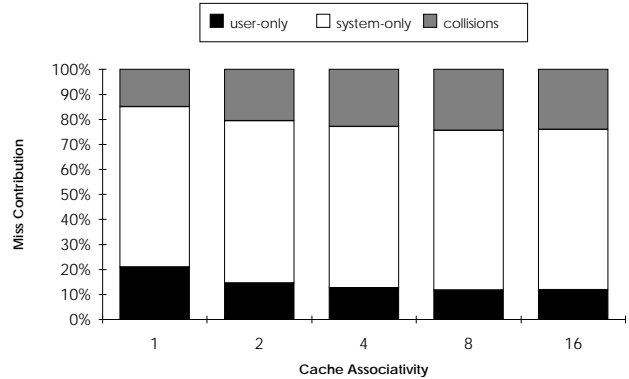


Figure 1: Percentage contribution of user, system, and collision cache misses for a variety 64-Kbyte caches with 16 byte lines using the SYSV-SDET trace

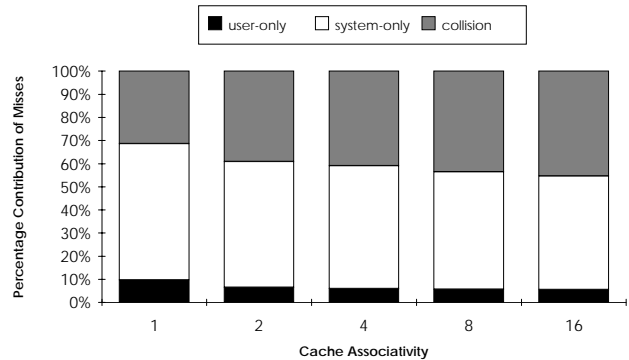


Figure 2: Percentage contribution of user, system (server+kernel), and collision cache misses for a variety of 64-Kbyte caches with 16 byte lines using the MACH-SDET trace

ing up nearly 60% of the SYSV-SDET references. In contrast, the operating system references account for 70% of the misses, while making up approximately 40% of the trace. Figure 2 shows that for the MACH-SDET trace, the operating system references (UNIX server + Mach kernel) generate approximately the same fraction of misses (70%).

These values are much higher than those reported in [6]. For the analysis in that paper a hardware monitor was used to capture traces of three benchmarks (Pmake, Multipgm, and Oracle) running on a multiprocessor. Pmake is a parallel make program, while Multipgm is a Pmake, along with a numeric application and five concurrent editor sessions. The fraction of misses due to operating system references was 52.6% and 46.3% respectively for these two workloads. The third benchmark, Oracle, is a scaled-down implementation of a transaction processing benchmark. It resulted in the operating system generating only 26.6% of the misses. All these results are considerably lower than the 70% operating system misses acquired through simulation with our SDET and KENBUS benchmarks. Even with a 256Kbyte cache similar to caches found on a SGI 4D/340, our miss-rates

are much higher than those reported by Torrellas et al. We believe our figures are higher because of fundamental differences in the benchmarks being traced. Database programs such as Oracle often bypass the operating system for I/O to increase performance. Thus many traditional operating system tasks show up in the Oracle trace as user tasks. In contrast, the SDET benchmark makes frequent use of many system utilities and is fairly I/O intensive. Additionally, the Ultrix and Mach results reported in [3] seem to agree with ours – the system misses are 70% or more of the total misses under both Ultrix and Mach for many benchmarks.

The large contribution of operating system activity in our traces cannot be attributed to the idle routine. Not only is the benchmark optimized for maximum throughput (minimal idle time), the idle loop in UNIX SVR4 on the i486 computer consists of a single HALT instruction.

Turning our attention now to collision miss differences between the two traces, we note that the most striking difference between Figure 1 and Figure 2 is the lower fraction of user misses and higher fraction collision misses for Mach. For example, under UNIX, the collision misses account for just over 20% of the misses for the 8 way set-associative cache of Figure 1. The corresponding value using the Mach trace is approximately 40%. This is because most system calls are handled by the Mach kernel calling the user-level UNIX server process. This result differs from those reported in [3], which claims that collision misses are not a major cause of performance degradation. We believe that the higher number of collision misses observed in our simulations are due to the multiprogramming workloads we use – the benchmarks traced in [3] were taken from SPEC92 and consist of CPU-bound single process workloads.

4.3 Dynamic miss-rates

A wide variation of cache miss-rates over time was illustrated in [2], disputing the idea that once the caches are warmed up, a steady state in miss rate is reached. Our results confirm this and extend it by demonstrating (1) the variability present in the complete trace and (2) the lack of correlation between the dynamic user-only and complete trace miss-rates. Figure 3 presents dynamic miss-rates – miss-rates computed for 10 million reference windows – for the SYSV-SDET trace. The curves were created by configuring the simulator to output statistics every 10 million references. The caches were not flushed after each window’s statistics were output, so only the first window contains cold start effects. The bottom curve represents our user-only traces that are similar to those used in [2] in that they are long and contain only user references. The top curve is for the complete trace collected using our hardware monitor. The figure shows that little, if any, correlation exists between the miss-rate of user-only code and the true miss-rate for the trace; the user-only miss-rates are consistently lower, but beyond that no obvious relationship exists.

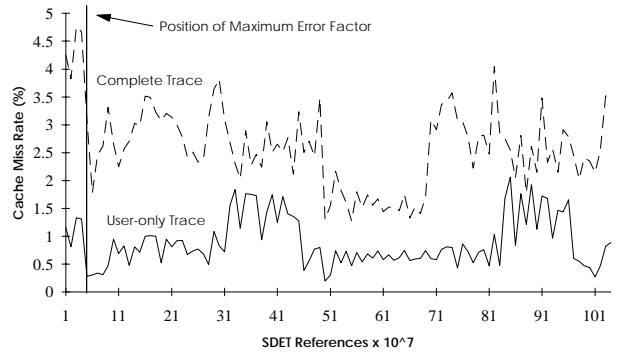


Figure 3: Dynamic miss-rates of a 64-Kbyte direct-mapped cache in ten million reference windows using the user-only and complete SYSV-SDET trace

4.4 Section summary

In summary, simulations using user-only traces may be missing as many as 70% of the cache misses. As a result, work that relies on the total number of misses per unit time, such as a bus utilization study, may greatly underestimate the load the traced workload places on the system. Similarly, a writeback policy study may underestimate the required depth of write buffers if based on a user-only trace. The transitions between the UNIX server and kernel in the MACH-SDET trace greatly increase the number of collision misses in the cache. In addition, the combined size of application plus operating system is much larger in the Mach environment, and this leads to higher cache miss rates. Finally, there is little correlation between the dynamic miss-rates of user-only and complete trace data. We demonstrate in Section 5 the effect of these higher miss-rates on effective memory access time.

5 Effective memory access time

We now look at how the miss-rate errors found in previous sections affect the prediction of effective memory access time. Effective memory access time may be approximated by

$$T_{eff} = T_{hit} + MR_{trace} * T_{miss} \quad (1)$$

where T_{eff} is the effective memory access time, T_{hit} is the time required to service a cache hit, MR_{trace} is the miss-rate of the cache for a particular workload, and T_{miss} is the time to access main memory. For this study, T_{hit} is one cycle. Using the average user-only and complete miss-rates resulting from the SYSV-SDET trace with Equation 1, the effective access times for a 64-Kbyte 8-way associative cache are plotted as a function of memory delay in Figure 4. The following points can be made:

1. Our user-only traces can be considered “best-case” having been derived from a complete trace by removing supervisor references. Other trace collection techniques likely would have produced traces with lower miss-rates due to their lack of true multiprogramming behavior or use of

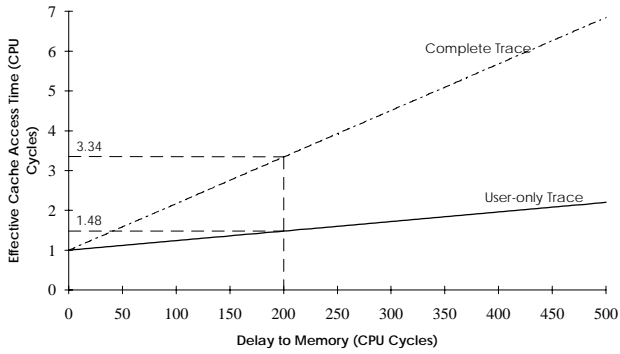


Figure 4: Effective cache access times for a 64-Kbyte 8 way associative cache for various delays to main memory using user-only and complete SYSV-SDET data

simpler/smaller benchmarks. This would have widened the gap between the lines in the figure by decreasing the slope of the user-only line.

2. The error in most previous studies is likely to be small if the memory access time was relatively short, if the traces were sufficiently long, and if the traced workload exhibited true multitasking behavior. Changes in any of these can significantly increase the magnitude of errors.
3. The accuracy of previous results depends on the delay to main memory. Processor cycle times have been decreasing at a faster rate than main memory access times, and machines with main memory delays as large as 200 [2] cycles have been discussed by researchers. Machines with these large memory delays would likely use additional levels of cache to reduce the effective memory access times. However, since the complete and user-only traces would both benefit from a second level cache, the relative position of the lines in Figure 4 would remain approximately the same – only the values on the y-axis would change. From the figure it can be seen that a 70 cycle delay to memory results in an error ratio of about 1.5, while at 200 cycles the estimate is in error by a factor of 2.3.

In Figure 4, only the SYSV-SDET data is plotted, due to space limitations, but the results for MACH-SDET data are similar. With a delay to main memory of 200, the error is 2.6 for MACH-SDET data and 2.3 for SYSV-SDET trace.

5.1 Error factor calculations

The ratio of predicted effective memory access times for complete and user-only traces is plotted in Figure 5 for the SYSV-SDET trace. For sufficiently large T_{miss} the ratio approaches some maximum value, which we define to be the error factor (EF). This may be calculated by

$$EF = \lim_{T_{miss} \rightarrow \infty} \frac{T_{hit} + MR_{complete} * T_{miss}}{T_{hit} + MR_{user-only} * T_{miss}},$$

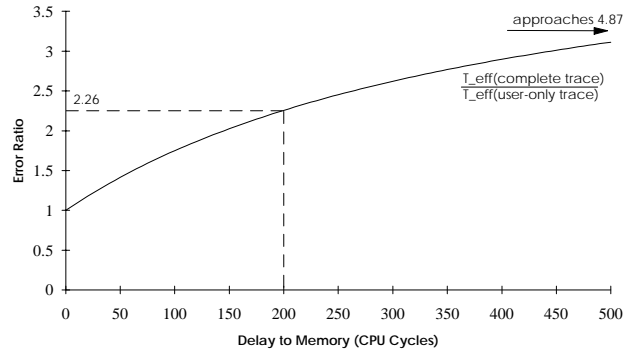


Figure 5: Error in estimating effective access times for a 64-Kbyte 8 way associative cache for various delays to main memory, $\frac{T_{eff}(complete)}{T_{eff}(user-only)}$, using SYSV-SDET data

$$EF = \frac{MR_{complete}}{MR_{user-only}}.$$

Tables 5 and 6 summarize error factors for a range

Cache Size	Cache Associativity				
	1	2	4	8	16
4-Kbyte	1.65	1.88	2.14	2.23	2.26
8-Kbyte	1.87	2.29	2.61	2.80	2.89
16-Kbyte	2.17	2.73	3.10	3.08	3.07
32-Kbyte	2.47	3.36	4.03	4.33	4.42
64-Kbyte	2.74	3.98	4.57	4.87	4.78
128-Kbyte	3.00	4.32	4.73	4.50	4.62
256-Kbyte	3.10	4.64	4.87	4.38	4.71

Table 5: Error factors for a variety of cache configurations using the SYSV-SDET trace data, $EF = \frac{MR_{complete}}{MR_{user-only}}$

of caches for SYSV-SDET and MACH-SDET. They range from 1.65 to 4.87 for SYSV-SDET and from 1.90 to 6.77 for MACH-SDET. For large main memory access times, our error factor is similar to the miss-rate ratios (ratio of complete to user-only miss-rates) used by Agarwal [5] et al. The ratios they computed using ATUM traces were about 2 – much lower than many presented here. This can be attributed to the larger benchmarks used for this study and the use they make of operating system resources.

6 Summary

Trace-driven simulation relies on accurate input data. Such accurate data has, to date, been difficult to obtain in the research community. Using a hardware monitor, we collected a number of long and accurate traces from the SPEC SDM1.1 multiprogramming suite. We then compared uniprocessor cache simulation results using these traces to those obtained using user-only data. Several observations can be made:

Size	Cache Associativity				
	1	2	4	8	16
4-Kbyte	1.90	2.32	2.52	2.67	2.72
8-Kbyte	2.30	3.07	3.65	3.87	3.89
16-Kbyte	2.74	4.10	5.08	5.54	5.73
32-Kbyte	3.58	4.86	5.80	6.17	6.29
64-Kbyte	3.83	5.59	6.13	6.67	6.77
128-Kbyte	4.26	5.63	5.74	5.44	5.47
256-Kbyte	4.14	4.60	4.15	3.83	3.58

Table 6: Error factors for a variety of cache configurations using the MACH-SDET trace, $EF =$

$$\frac{MR_{complete}}{MR_{user-only}}$$

1. We have traced the SPEC SDM benchmarks and will make the traces available on request to other researchers. The benchmarks have large working sets (4-9MB) and significant multitasking behavior compared to the commonly used SPECint and SPECfp benchmarks.
2. For both Mach and UNIX, supervisor code does not benefit as much from increased associativity as user code. This is contrary to previous results using the SPEC92 benchmarks.
3. The operating system contributes nearly 70% of all misses generated in 64 Kbyte unified caches. These figures are considerably higher than those reported in [6], and very similar to those reported in [3]. We believe that the differences are primarily the result of the workloads traced.
4. We measured the dynamic or windowed miss-rate computed for 10 million reference windows and found it to vary by as much as an order of magnitude over the trace. More importantly, we found the dynamic miss-rate graphs of user-only traces to be quite different from those generated using complete trace data; there was little correlation, if any, between the two.
5. We demonstrated that the miss-rates using complete trace data are higher for Mach than UNIX System V R4.
6. We determined that for small main memory delays, the error in effective memory access time prediction was minimal. However, as relative access times of main memory increase, the errors become significant. With a memory delay of 200 cycles, the error factor is 2.26, meaning that the actual effective memory access time is 2.26 times as great as that predicted using user-only traces.

References

- [1] A. Agarwal, R. L. Sites, and M. Horowitz, ATUM: A new technique for capturing address traces using microcode, In *Proc. of 13th Int. Symp. on Computer Architecture*, pages 119–127. IEEE, 1986.
- [2] A. Borg, R. E. Kessler, and D. W. Wall, Generation and analysis of very long address traces, In

Proc. of 17th Int. Symp. on Computer Architecture, pages 270–279. ACM, 1990.

- [3] J. B. Chen and B. Bershad, The impact of operating system structure on memory system performance, In *Proc. of 14th Symp. on Operating System Principles*. ACM, 1993.
- [4] J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud, Incomplete trace data and trace driven simulation, In *Proc. of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems MASCOTS*, pages 203–209. SCS, 1993.
- [5] Anant Agarwal, J. Hennessy, and M. Horowitz, Cache performance of operating system and multiprogramming workloads, *ACM Transactions on Computer Systems*, 6(4):394–431, November 1988.
- [6] Josep Torrellas, Anoop Gupta, and John Hennessy, Characterizing the cache performance and synchronization behavior of a multiprocessor operating system, Technical Report CSL-TR-92-512, CSL, Dept. of EECS, Stanford University, January 1992.
- [7] Gurindar S. Sohi and Manoj Franklin, High-bandwidth data memory systems for superscalar processors, In *Proc. of 4th Int. Conf. on Arch. Support for Prog. Lang. and Operating Systems*, pages 53–62. ACM, 1991.
- [8] D. Nagle, R. Uhlig, and T. Mudge, Monster: A tool for analyzing the interaction between operating systems and computer architectures, Technical report, The University of Michigan, 1992.
- [9] Douglas W. Clark and Joel S. Emer, Performance of the VAX-11/780 translation buffer: Simulation and measurement, *ACM Transactions on Computer Systems*, 3(1):31–62, February 1985.
- [10] J. Kelly Flanagan, Brent E. Nelson, James K Archibald, and Knut Grimsrud, BACH: BYU Address Collection Hardware, the collection of complete traces, In *Proc. of the 6th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 128–137, 1992.
- [11] K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, and B. Nelson, BACH: A hardware monitor for tracing microprocessor-based systems, *Microprocessors and Microsystems*, 17(6), October 1993.