Trace Driven Simulation in Research on Computer Architecture and Operating Systems*

Alan Jay SMITH
Computer Science Division
University of California
Berkeley, California 94720, USA
smith@cs.berkeley.edu

Abstract

Trace driven simulation (TDS) is a widely used and important technique in computer architecture and system research. Conventional (random number driven) discrete event simulation uses implicit or explicit models of external processes to provide the events to drive the simulation. Such models can fail to capture extremely important characteristics of the driving process. TDS drives the simulation with a trace, which is a recorded sequence of real events believed to be representative of what the simulated system could expect. To the extent that the trace is (approximately) representative of the events that would actually occur in the simulated system, TDS provides a much more realistic workload.

TDS is the standard technique for studying computer system memory hierarchies and many other aspects of computer design and performance. Traces of memory addresses are used to drive cache and main memory studies. Traces of I/O events are used to study disk and file systems. Traces of instructions are used to study CPU implementation and to estimate performance. Traces of CPU runtime intervals have been used to study CPU scheduling.

In this paper, we will discuss how research studies such as the above are conducted and will survey some of the relevant issues. We will also consider the limitations and difficulties of TDS.

1. Introduction

Simulation is widely used as a tool for system design and for research studies. It has advantages over both analytic modeling and prototype implementation. Analytic models are seldom able to incorporate sufficient realism, and often have to be recreated if a different system structure is to be considered. Implementations of any type usually require at least an order of magnitude more effort than a simulation, frequently can represent only one of a set of design alternatives, and often cannot be tested under realistic conditions or with realistic workloads. Simulations can model the target design at any level of detail, and can be modified as necessary to reflect alternatives and changes.

One class of simulations are called *discrete event simulations*; these are simulations over time in which the state of the simulation model changes at discrete epochs, when various events occur. Such a model is typically driven by an event queue; the simulation process is based on the loop: {get event, update simulation statistics, update system state and clock, update (delete from/add to) event list}. In classical simulation, some or all of the events that drive the simulation are generated using something that is essentially a random number generator, although the (uniform) random number generator is usually used to generate something ranging from a more complex distribution to a sophisticated input process model. The presumption in such models is that the inputs (workload) are reasonably well known or understood, but that the effect of the input on the output is complex and must be determined through the simulation. To phrase it mathematically, the simulation output y is determined by the simulation $f(.)$ and the input workload x as $y=f(x)$; for many simulation problems, x is fairly well understood, but $f(.)$ is not.

For many problems in computer architecture and computer operating systems performance and design, the problem is not in mapping a known input to an output, but in the fact that the relevant aspects of the input are not adequately understood.

*The author's research is supported in part by the National Science Foundation under grants MIP-9116578 and CCR-9117028, by NASA under Grant NCC 2-550, by the State of California under the MICRO program, and is or has been recently supported by Sun Microsystems, Apple Computer Corporation, Mitsubishi Electric Research Laboratories, Philips Laboratories, Digital Equipment Corporation, and the International Business Machines Corporation.

This paper was published in the Proceedings of the Conference on New Directions in Simulation for Manufacturing and Communications, ed. Morito, Sakasegawa, Yoneda, Fushimi, Nakano, Tokyo, Japan, August, 1994, pp. 43-49.

Frequently in such cases, $f(.)$ is sufficiently simple that if there were a simple characterization of x, the nature of y would be immediately evident. For such problems, *trace driven simulation (TDS)* is particularly valuable. TDS is a type of event driven simulation in which the events come from a *trace*, rather than being generated as an input to the simulator or as part of it. The *trace* is a sequence of the actual simulator input events gathered from a real functioning system sufficiently similar to the system being modeled; thus, the entire event list is available and is known at the start of the simulation.

The advantage to trace driven simulation is that since the input trace is a sequence of real events, taken from a real, functioning system, it will contain within it all the characteristics of the workload, including those relevant to the proposed studies. An additional advantage is that since the trace is known, fixed and static, a variety of simulations can be run using the same trace with some assurance that any difference in simulation results is due to changes in the model parameters, not variations in the input. This is not to say that there are no problems with trace driven simulation, many of which are described below, but it does eliminate one source of error.

Trace driven simulation has been used for research on computer systems for many years. Among the early studies, it is worth citing [Bela66] which was an early and excellent study of paging algorithms, and [Sher72], which was a very interesting study of CPU scheduling. Trace driven simulation is also frequently used for studies of I/O systems, disk caching, file migration, pipeline performance, and cache memories, each of which is considered below.

In the remainder of this paper, we first consider some of the general issues in trace driven simulation, such as trace generation, trace storage, trace length, representative trace samples, efficient algorithms, etc. We then briefly discuss the use of TDS for studies of cache and main memory, disk systems, file migration, CPU scheduling and CPU pipeline analysis. Finally, we also briefly discuss some techniques for using parallel processors for trace driven simulation.

2. Trace Driven Simulation

As explained above, trace driven simulation (TDS) uses a trace of events to drive an event driven simulation, rather than generating the events on the fly from a stochastic model. The simulation model itself may internally contain some stochastic aspects, but the principal events driving the model come from the trace. There are a number of issues that relate to the difficulty and validity of TDS; we discuss those in this section.

2.1. Obtaining A Trace

Generally, the most serious issue with TDS is that of obtaining a trace of suitable events. This problem breaks down into two simpler problems: finding a system in which the relevant events occur, and then collecting the trace. Fortunately, there are very few advances in computer systems that are so radical that there is no suitable system that can be traced; this will become clear from the examples we discuss below. Unfortunately, in many cases it is quite difficult to obtain the relevant traces, since it can be quite difficult to instrument the existing system to obtain the appropriate data; this is also discussed below.

2.2. Representative and Valid Trace

In some cases, there may be no "representative" or valid workload. For example, there may be many parallel algorithms to solve a given problem, or a number of ways to code a given parallel algorithm on a multiprocessor, and the choice of algorithm could make an enormous difference in the results obtained; which algorithm should be traced? In other cases, the most representative data (e.g. operating systems address traces [Flan93]) may be very hard to obtain. In still other cases, changes of scale (orders of magnitude increases in CPU speed, disk size, main memory size over a decade) may rapidly make a trace obsolete. Each of these issues needs to be carefully considered when relevant. An important point to note, however, is that the situation only gets worse without a trace; a trace represents a data point, and generally one or several data points are significantly better than a random number driven simulation, which really only represents a guess. To the extent that there is a recognized problem with a trace, it may be possible to adjust, scale or supplement the data to account for anticipated changes in the workload or environment.

One problem with trace driven simulation is that the trace may not be completely invariant to differences between the traced system and the simulated system. For example, changes in the way memory conflicts are resolved in a parallel processor system could cause changes in the relative speed of the various processors/processes, and thus changes in the actual sequence of memory references. Likewise, use of disk caching or changes in disk scheduling algorithms could cause changes in the timing of arrivals of disk I/O requests. Thus it may not be completely valid to use a fixed, invariant trace, if the changes in the simulated system would have caused the actual generated trace to be different. Care must be taken to consider and evaluate this problem; if it is believed to be severe, execution driven simulation (described below) may be preferable.

2.3. Trace Storage

For some types of studies, e.g. cache memories or disk I/O analysis, the trace may be very long. For example, in one study [Borg90], traces of several billion memory references were used; in another [Gee93a], many billions of memory references were processed. There are four approaches to this problem. First, with modern storage technologies (helical scan tapes, such as DAT tapes or Exabyte 8mm tapes), long traces can be stored cheaply; I/O data rates for these technologies, however, are quite low - around .25 MB/sec. A second approach is to use data compression; Samples [Samp89] obtained around 95% percent compression with some rather simple techniques; more effort might have yielded even better results. A third approach is to process the trace and discard events that have little effect on the performance metric. For example, [Smit77] shows that references to the top of the stack in a memory simulation can be discarded, with little effect on a count of the number of misses for a reasonably sized memory.

2.4. Execution Driven Simulation

The fourth approach is to use *execution driven simulation (EDS)*, which can be considered to be a variant of trace driven simulation (see e.g. [Davi91, Dwar93]). (EDS is also called "direct simulation.") With TDS, a system is traced, the trace is stored, and then the simulation is run from the trace. With execution driven simulation, the events from the traced system are fed directly into the simulated system. This avoids the data storage problems described above. EDS can be taken a step further, with a feedback loop from the simulated system to the traced one. In this case, the traced system relies on the simulated system to provide feedback or results of various types. This permits the sequence of events (the trace) to vary with changes in the simulated system. The advantage is that the trace retains its validity, but with the disadvantage of a simulation that is actually driven by a different sequence of events in each case. Generally, EDS is more difficult than simple TDS, since it combines trace collection with TDS, rather than separating the two.

2.5. Trace Length

The use of long traces has in the last few years become quite fashionable, but is frequently unnecessary; it can be a poor substitute for good experimental design. Long traces are usually collected by tracing a given system or program for an extended period of time. Since the target system is unlikely to be identical to the traced system, we believe it is generally far better to use an equal amount of trace data gathered from a variety of systems over a variety of trace periods. Such a varied trace workload is much more likely to include within its range the workload to be experienced on the system being studied, and also provides for a variety of simulation results, one for each input, which can demonstrate the variation to be expected. Further, samples taken intermittantly over a long period can be much more useful than the same aggregate trace length over a shorter continuous period. The only advantage to a long trace is that there may be an extended period required to initialize the simulation model (e.g. fill the cache). A simulation run from an "empty" state is often called *cold start*, and one from a fully initialized state is frequently referred to as *warm start*. The initialization problem can be overcome by one of the following: (a) preprocessing a continuous trace to obtain the (approximate) initial state; (b) discarding those portions of the simulation prior to warm full initialization; and/or (c) calculating (in some manner) the effect of the initialization transient on the simulation results.

Even though many computer system models can be very simple, e.g. an LRU stack, long simulation runs (billions of events) can result in very long simulation times; in [Gee93a], 17 months of CPU time were used. There are a number of approaches to this problem. Some (e.g. [Hill89, Matt70, Thom87,89]) are algorithms for generating the results for a large number of parameter combinations from one simulation run. Others (e.g. [Cmel93]) are techniques for coding the simulations very efficiently. In the experience of this author, a simple stack simulation will run 100 to 1000 times slower than would the actual system being simulated. [Cmel93] claims to run less than 10 times slower using his coding and simulation tricks.

In the next few sections, we discuss various aspects of computer systems and the use of TDS to study them.

3. Cache, Main Memory and TLB Studies

A computer system contains an ALU (arithmetic-logic unit) which does instruction fetches, loads and stores to main memory. Main memory is typically much slower than the ALU, so in modern computer systems a cache memory is used to make most memory accesses much faster. Most modern computer systems use virtual memory and paging, so a TLB (translation lookaside buffer) is needed to cache page table entries. Studies of caches, TLBs and main memory paging generally all consider the issue of hit ratio, which is the fraction of events (I-fetches, loads, stores, translations) that are satisfied by the faster element (cache relative to main memory, TLB vs. translator, main memory vs. disk). All three types of studies use as input a trace of virtual addresses (instruction fetches, loads, stores) generated by the ALU. The virtual address trace generated, for a given uniprocessor program, ignoring task switching, is invariant with respect to changes in the memory system, the contents of the page table or almost anything else, so driving a simulation of a variety of memory configurations is generally valid.

3.1. Address Trace Generation

There are a number of techniques for generating address traces; see [Laru93] for a discussion of some of these. One of the earliest techniques was to write a simulator of the architecture which the code being traced runs on. The object code is then interpretively executed by the simulator, which generates appropriate trace records. This is particularly easy to do for the IBM 360/370, which has an "execute" instruction, which executes any other instruction it "points" at [Peut76]. This is the method used to generate many of the traces used in [Peut76, Lee84]. One of the few ways to generate operating systems traces is to use a machine simulator and run it in a virtual machine environment; that was the technique used to generate OS traces at Amdahl Corporation (see [Smit85b] for the analysis of some OS traces).

Another old technique for collecting traces is to use the "trace-trap" facility. Most computers have various trace-trap facilities, by which the machine can be set to trap if various things happen, such as a memory access to a certain memory region, etc. These features are usually included for debugging purposes, but can generally be set to cause a trap on every instruction fetch. The trap handler then generates a trace record and resumes execution of the program.

Similar to the trace-trap method is to invalidate either the page tables or the TLB (if TLB misses are handled in software). There is then a trap on every memory reference, and a trace record can be generated.

A hardware monitor can be used to collect address traces from the appropriate signal pins [Grim93]. This technique has seen only very limited use for a number of reasons: (i) a hardware monitor is needed, and they are often hard to obtain; (ii) the monitor must be fast enough to collect the data - i.e. usually faster than the system being traced; (iii) the monitor must have substantial amounts of storage in which to collect the trace data; (iv) the necessary signals must be available, which is rare; (v) the experimenter must have the knowledge to instrument the system properly; (vi) the experimenter must have access to the system hardware.

In the case of a microcoded machine, the microcode can be modified to generate traces [Agar86]. The problems here are that there are few microcoded machines (any more), and the experimenter must have the knowledge and ability to modify the microcode, and access to a machine to which the modification can be made.

Probably the most popular technique currently for generating traces is to instrument the object code. A call to a tracing routine is inserted for every load, store and branch. A postprocessing step adds trace records for instruction fetches between branches. This approach is used by the *pixie* facility, available on MIPS processor based systems. This technique is reasonably efficient, and lends itself well to execution driven simulation.

3.2. Cache Studies

Memory address traces have been used for a variety of studies of CPU cache memories, many of which are illustrated in [Smit82]. These include the effect of line (block) size [Smit87], the effect of prefetching on the miss ratio [Smit78b,85b], the effect of the frequency of task switching, the effect of associativity [Hill89], etc. Most of these studies have used very simple cache models, for which the metric of interest was the miss ratio. For some issues, e.g. prefetching [Smit78b], a more complicated model with detailed timing information is needed; such a study is in progress [Tse94]. It is worth noting that a major problem with many of the studies that have been done on cache memories is that the bulk of the cache misses occur while the operating system is executing [Smit82, Gold93], and very few studies have included OS traces.

It is worth pointing out that cache memory studies clearly illustrate the need for trace driven simulation. Although there are many models in the literature for program behavior, none purports to model all of the necessary features (effect of line size, effect of associativity, effect of prefetching, etc.) of real traces. Even traces of real programs from standard benchmark sets can fail to be representative of real workloads [Gee93a]. Traces of individual programs are also of limited use for studying large caches, since a substantial fraction of the misses for large caches are due to task switching effects (and the corresponding cold starts), and traces that cross task switch boundaries are seldom available.

Recently, studies of the design of caches for multiprocessor systems have become popular. Such studies were long known to be useful and interesting, but were held up for the lack of suitable multiprocessor traces. A problem with these studies, however, is that there does not seem to be a genuinely representative workload; depending on how the code is written, performance and behavior can vary widely [Gee93b,c].

3.3. Other Memory Address Driven Studies

Trace driven simulation of memory systems was first used for studies of main memory paging (see e.g. [Bela66]). Most such studies also considered only miss ratios, although a few timing studies were also done. These studies looked at issues such as page size, page replacement algorithm, page fetch algorithm, etc. An overview of research on paging appears in [Denn80], although without much emphasis on TDS. See [Smit78c] for a bibliography on paging.

Memory address traces have also been used to study memory interference in shared memory systems [Bask76]. In this case, it was shown that representing an address stream as randomly referencing the various memory modules was a good approximation.

Branch target buffers (BTBs) are caches for branches (and their targets); they are used in the ALU to predict when branches will be taken and what their targets will be. A successful prediction means that there will be no pipeline "hole." BTBs can also be studied using instruction and memory address traces [Lee84, Perl93].

4. Disk Cache and I/O Optimization

The large ratio between the time to do an I/O and the CPU cycle time (10^6 or more) has made optimization of the I/O system crucial. Many of these optimization studies have relied on I/O traces. The use of these traces for I/O studies is similar to studies of main and cache memories, but collecting the traces is quite different. We discuss both in this section.

In collecting I/O traces, we would like several types of information, since various types of study require different data; see [Zhou85, Toua91] for a discussion of many of these issues. Ideally, we would like trace records for each open, close, read, write, rename and file delete event. In each case, we'd like to know the owner of the process, the owner of the file, the process ID, and the time. For reads and writes, we'd like both the logical (file name and address within file) and physical (disk address, cylinder, track and sector) addresses. For opens, closes, renames, and file deletes, we'd like to know the file name, size and possibly layout.

There are two general methods to collect the data described above. The first is to use an existing trace package if available; the second is to modify the operating system. In general, trace packages are not available, but sometimes there exists other types of tracing facilities which can be pressed into service. For example, in IBM mainframes running MVS and its variants, the GTF (general trace facility) feature can be used to generate trace records for all system calls. The problem is that GTF was designed for debugging and does not generate all of the necessary information, generates what it does in inconvenient formats, generates far too much unneeded information, and causes a high level of overhead. There is also SMF (system management facility) data; SMF generates data for many of the higher level events (e.g. opens, closes, etc.). SMF, however, was primarily designed for accounting purposes, and suffers from the same types of flaws when used for tracing as GTF. Finally, there is the difficult problem of merging the two types of data; we discuss the latter issue further below.

The other approach to collecting I/O trace data is to modify the operating system. There are two general problems here. First is the general difficulty involved of obtaining access to the OS source code, understanding it, modifying it, and then being able to bring it up on a machine running a real user workload. The second problem is that generally, modifications are required throughout the operating system, because the desired data is not usually available in one place. For example, by the time (in the control flow of the OS) that the physical I/O address is available, the file name, user and logical address are generally unavailable.

A third approach, of limited utility, is to use a hardware monitor to record the signals between the CPU and the I/O devices. The problem is that those signals contain only low level information (physical disk addresses and commands).

Once trace data has been generated, using either of the first two approaches described above, it needs to be placed in usable form. That generally means merging two or more types of trace data (e.g. SMF/GTF, or physical and logical I/O records). This merge is greatly complicated by the fact that there may be no easy way to tie together related records (matching physical and logical data may not be easy), and the huge volume of data makes the programming and logistics very difficult. Once the data has been merged, it helps to create unique identifiers for relevant entities (file names, user names, disk addresses, etc.), which greatly facilitates the analysis. This is all a huge amount of work. For example, man years were required for each of the studies described in [Smit85a] and [Zhou85] to make the data usable.

A problem with I/O studies is that the traces may not be invariant with respect to changes in the I/O architecture or operation. Changing the speed of I/O operations, for example, may change the process scheduling in the CPU, and thereby the sequence of I/O addresses issued.

A variety of studies are possible using I/O traces. Disk cache has been studied extensively (see e.g. [Smit85a]), as has database buffering [Smit78a]. Various I/O optimizations have been investigated (see [Smit81a] for a survey), such as disk arm scheduling, file placement, block size optimization, etc. More modern topics such as RAID disk systems [Katz89] and log-structured file systems [Rose91] have also been considered. Traces taken from database systems [Sing94] have been used to study things such as the behavior of locking algorithms.

5. File Migration

File migration has to do with the movement of files between levels of the memory hierarchy (typically between disk and tape) so as to maintain online a large volume of data while using a much smaller amount of expensive disk. The data necessary for file migration studies is much easier to collect than that for general I/O studies. At most, a trace of open events (with file sizes and time stamps) is sufficient. One can even do very interesting studies given only information, for each day, as to which files were used that day [Smit81b]. In general, one wants to trace a system with a very large online file store (disk and/or tape), such that miss ratios can be calculated for smaller physical file systems. It would seem difficult to extrapolate performance for larger file systems than are actually traced.

6. CPU Scheduling

CPU scheduling is another issue that has been studied using trace driven scheduling. In an early study [Sher72], a variety of scheduling algorithms were compared, using a trace of run time intervals. Each interval was a period of execution terminated by some sort of trap or interrupt.

There are three general difficulties with such scheduling studies. First is collecting the data, which may require modifying the operating system. The second is the question about the invariance of the trace with respect to changes in the scheduling. Third is the fact that actual operating systems schedulers make scheduling decisions based on factors other than run length intervals, such as memory allocations, resources (disks, tapes, semaphores) needed, etc. Collecting this additional information can be difficult, and a simulation to use it could be very complicated.

7. CPU Pipeline Analysis

Instruction traces are frequently used to drive simulations of CPU pipelines. Such instruction traces can be gathered using most of the same methods as were described above for gathering program address traces - e.g. the trace trap facility, the object code modification approach (pixie), the machine simulator, etc. The purposes of such simulations are twofold. First, there are studies relating to performance [Peut76], to measure various types of pipeline stalls, number of instructions per second, etc. The second is that of debugging, whereby a detailed logic simulator for the CPU is driven by an instruction trace in order to determine if there are any errors in the logic design.

For performance studies of most machines, it is only necessary to consider instruction sequences between taken branches [Koba84]. This is because when a branch occurs, the pipeline is reloaded, and so the branch is effectively a regeneration point in the simulation. Note that if branch prediction is used, the set of sequences that needs to be considered changes with any change in the branch prediction methodology.

8. Parallel Simulation

The availability of parallel processors has led to a number of techniques for running parallel simulations of various aspects of computer systems. Many of these techniques are specialized, and do not represent the generic type of simulation available in the literature [Chan81, Kona91, Misr86].

For a set associative memory design, separate processors can be assigned to each simulate a subset of the sets in the memory [Puza85]. For multiprocessor studies, each processor of the simulation engine can be mapped to one or more processors of the simulated system [Rein93, Qin94]. In some cases, it is possible to divide a trace into a number of sequential segments, with each segment simulated by a separate processor. The issue here is that of initial conditions, but those can be provided by a second pass; i.e. the processor using the k'th segment of the trace generates results contingent on some unknown initial conditions. Those conditions are provided when the simulation of the k-1'st segment is complete (and so on, recursively, back to segment 1) [Harp93].

There are two general problems with parallel simulation techniques. First is to find sufficient parallelism in the system being simulated, or alternatively, to find a way (as with the last technique above) to work around a lack of parallelism. The second problem is the overhead of communication between the processors performing the simulation; this overhead can be the limiting factor in the speed of the simulation.

9. Conclusions

Trace driven simulation is a very powerful technique for obtaining valid simulations of systems in which the workload (the input) is not well understood and fully characterized. It is the overwhelmingly dominant technique used for studies of computer systems, particularly memory hierarchies. It is also widely used for debugging digital logic. In this paper, we have reviewed the principal subjects of such simulations and have discussed the issues of trace generation, management and validity.

Bibliography

[Agar86] Anant Agarwal, Richard L. Sites and Mark Horowitz, "ATNUM: A New Technique for Capturing Address Traces Using Microcode", Technical Report, Digital Equipment Corporation, DEC-TR-415, December, 1985. republished in Proc. 13'th Ann. Int. Symp. on Computer Architecture, June 2-5, 1986, Tokyo, Japan, pp. 119-127.

[Bask76] Forest Baskett and Alan Jay Smith, "Interference in Multiprocessor Computer Systems with Interleaved Memory", Communications of the ACM, 19, 6, June, 1976, pp. 327-334.

[Bela66] L. A. Belady, A Study of Replacement Algorithms for A Virtual Storage Computer, IBM Sys. J., 5, 2, 1966, pp. 78-101.

[Borg90] Anita Borg, R.E. Kessler, and D.W. Wall, "Generation of Very Long Address Traces", Proc. 17'th ISCA, May, 1990, pp. 270-281.

[Chan81] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", CACM, 24, 11, April, 1981, pp. 198-206.

[Cmel93] Robert Cmelik and David Keppel, "Shade: A Fast Instruction Set Simulator for Execution Profiling", Technical Report UWCSE 93-06-06, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA.

[Davi91] Helen Davis, Stephen Goldschmidt, John Hennessy, "Multiprocessor Simulation and Tracing Using Tango", Proc.

1991 Int. Conf. on Parallel Proc., August, 1991, Penn. State University, pp. II-99 - II-107.

[Denn80] Peter Denning, "Working Sets Past and Present", IEEETSE, SE-6, 1, January, 1980, pp. 64-84.

[Doug89] Fred Douglis and John Ousterhout, "Log Structured File Systems", Proc. Compcon, Spring, 1989, February, 1989, San Francisco, CA, pp. 124-129.

[Dwar93] S. Dwarkadas, J.R. Jump, R. Mukherjee and J.B. Sinclair, "Execution Driven Simulation of Shared Memory Multiprocessors", Proc. MASCOTS'93, pp. 83-86.

[Flan93] J.K. Flanagan, B. E. Nelson, J.K. Archibald, K. Grimsrud, "Incomplete Trace Data and Trace Driven Simulation", Proc. MASCOTS'93, January, 1993, San Diego, CA, pp. 203-209.

[Gee93a] Jeffrey Gee, Mark Hill, Dionisios Penvmatikatos and Alan Jay Smith, "Cache Performance of the SPEC Benchmark Suite", IEEE MICRO, 13, 4, August, 1993, pp. 17-27,

[Gee93b] Jeffrey Gee and Alan Jay Smith, "Analysis of Multiprocessor Memory Reference Behavior", Technical Report UCB/CSD-93-754, June, 1993, to appear, ICCD, 1994.

[Gee93c] Jeffrey Gee and Alan Jay Smith, "Absolute and Comparative Performance of Cache Consistency Algorithms" Technical Report UCB/CSD-93-753, June, 1993, submitted for publication.

[Gold93] Stephen Goldschmidt and John L. Hennessy, "The Accuracy of Trace-Driven Simulations of Multiprocessors", Proc. Sigmetrics'93, May, 1993, Santa Clara, CA, pp. 146-157.

[Grim93] K. Grimsrud, J. Archibald, M. Ripley, K. Flanagan, and B. Nelson, "BACH: A Hardware Monitor for Tracing Microprocessor-based Systems", Microprocessors and Microsystems, 17, 8, October, 1993, pp. 443-459.

[Harp93] D.T. Harper III and David Tuma, "A Parallel Algorithm for Cache Miss Ratio Evaluation", Proc. MASCOTS'93, January, 1993, San Diego, CA, pp. 79-82.

[Hill89] Mark D. Hill and Alan Jay Smith, "Evaluating Associativity in CPU Caches", IEEE Transactions on Computers, December, 1989, 38, 12, pp. 1612-1630.

[Katz89] Randy Katz, Garth Gibson and David Patterson, "Disk System Architectures for High Performance Computing", Proc. IEEE, 77, 12, December, 1989, pp. 1842-1858.

[Kona91] Pavlos Konas and Pen-chung Yew, "Parallel Discrete Event Simulation on Shared Memory Multiprocessors", Technical Report CSRD 1079, Center for Supercomputing Research and Development, University of Illinois, April, 1991.

[Koba84] Makoto Kobayshi, "Dynamic Characteristics of Loops", IEEETC, C-33, 2, February, 1984, pp. 125-132.

[Lara93] James Larus, "Efficient Program Tracing", IEEE Computer, May, 1993, pp. 52-61.

[Lee84] John K-F Lee and Alan Jay Smith, "Analysis of Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer, 17, 1, January, 1984, pp. 6-22.

[Matt70] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies", IBM Systems J., 1970, pp. 78-117.

[Misr86] J. Misra, "Distributed Discrete Event Simulation", Computing Surveys, 18, 1, March, 1986, pp. 39-65.

[Perl93] Chris Perleberg and Alan Jay Smith, "Branch Target Buffer Design and Optimization", IEEETC, 42, 4, April, 1993, pp. 396-412.

[Peut76] Bernard L. Peuto and Leonard J. Shustek, "An Instruction Timing Model of CPU Performance", Proc. 4'th Ann. Symp. on Computer Arch., March, 1977, pp. 165-178.

[Puza85] Tom Puzak, "Cache Memory Design", Ph.D. Thesis, ECE Dept., University of Massachusetts, 1985.

[Qin94] Xiaohan Qin and J.L. Baer, "A Parallel Trace-driven Simulator: Implementation and Performance", Technical Report, University of Washington, January, 1994.

[Rein93] Steven Reinhardt, Mark Hill, James Larus, Alvin Lebeck, James Lewis and David Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers", Proc. Sigmetrics'93, May, 1993, Santa Clara, CA, pp. 48-60.

[Rose91] Mendel Rosenblum and John Ousterhout, "The Design and Implementation of a Log-Structured File System", Proc. 13'th SOSP, October, 1991, Asilomar, CA, pp. 1-15. Republished, ACM TOCS, 10, 1, February, 1992, pp. 26-52.

[Samp89] A.D. Samples, "Mache: No Loss Trace Compaction", Proc. 1989 ACM Sigmetrics Conf., pp. 89-97.

[Sher72] Stephen Sherman, Forest Baskett III and J. C. Browne, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System", CACM, 15, 12, December, 1972, pp. 1063-1069.

[Sing94] Vigyan Singhal and Alan Jay Smith, "Characterization of Contention in Real Relational Databases" Technical Report UCB/CSD-94-801, Computer Science Division, UC Berkeley, March, 1994. Submitted for publication.

[Smit77] Alan Jay Smith, "Two Methods for the Efficient Analysis of Memory Address Trace Data", IEEE Transactions on Software Engineering, SE-3, 1, January, 1977, pp. 94-101.

[Smit78a] Alan Jay Smith, "Sequentiality and Prefetching in Data Base Systems", IBM Research Report RJ 1743, March 19, 1976, and ACM Transactions on Data Base Systems, 3, 3, September, 1978, pp. 223-247.

[Smit78b] Alan Jay Smith, "Sequential Program Prefetching in Memory Hierarchies", IEEE Computer, 11, 12, December, 1978, pp. 7-21.

[Smit78c] Alan Jay Smith, "Bibliography on Paging and Related Topics", Operating Systems Review, 12, 4, October, 1978, pp. 39-56.

[Smit81a] Alan Jay Smith, "Input/Output Optimization and Disk Architecture: A Survey", Performance Evaluation, 1, 2, 1981, pp. 104-117.

[Smit81b] Alan Jay Smith, "Long Term File Migration: Development and Evaluation of Algorithms", Communications of the ACM, 24, 8, August, 1981, pp. 521-532.

[Smit82], "Cache Memories", Computing Surveys, 14, 3, September, 1982, pp. 473-530.

[Smit85a] Alan Jay Smith, "Disk Cache - Miss Ratio Analysis and Design Considerations", ACM Transactions on Computer Systems, 3, 3, August, 1985, pp. 161-203.

[Smit85b] Alan Jay Smith, "Cache Evaluation and the Impact of Workload Choice", Proc. 12'th International Symposium on Computer Architecture, June 17-19, 1985, Boston, Mass, pp. 64-75.

[Smit87] Alan Jay Smith, "Line (Block) Size Selection in CPU Cache Memories", IEEE Transactions on Computers, C-36, 9, September, 1987, pp. 1063-1075.

[Thom87] James Thompson, "Efficient Analysis of Caching Systems", Report UCB/CSD 87/374, (Ph.D. Thesis), UC Berkeley, October, 1987.

[Thom89] James G. Thompson and Alan Jay Smith, "Efficient (Stack) Algorithms for Analysis of Write-Back and Sector Memories", January, 1987. ACM Transactions on Computer Systems, 7, 1, February, 1989, pp. 78-116.

[Toua91] Herve Touati and Alan Jay Smith, "Reducing and Manipulating Complex Trace Data", Software Practice and Experience, 21, 6, June, 1991, pp. 639-655.

[Tse94] John Tse and Alan Jay Smith, "An Accurate, Timing Based Evaluation of the Utility of Prefetching", in preparation.

[Zhou85] Songnian Zhou, Herve DaCosta and Alan Jay Smith, "A File System Tracing Package for Berkeley Unix", Proc. 1985 USENIX Summer Conference, Portland, Oregon, (Hosted by University of Oregon), June 12-14, 1985, pp. 407-419.