Line (Block) Size Choice for CPU Cache Memories

ALAN JAY SMITH, SENIOR MEMBER, IEEE

Abstract—**The line (block) size of a cache memory is one of the parameters that most strongly affects cache performance. In this paper, we study the factors that relate to the selection of a cache line size. Our primary focus is on the cache miss ratio, but we also consider influences such as logic complexity, address tags, line crossers, I/O overruns, etc. The behavior of the cache miss ratio as a function of line size is examined carefully through the use of trace driven simulation, using 27 traces from five different machine architectures. The change in cache miss ratio as the line size varies is found to be relatively stable across workloads, and tables of this function are presented for instruction caches, data caches, and unified caches. An empirical mathematical fit is obtained. This function is used to extend previously published *design target miss ratios* to cover line sizes from 4 to 128 bytes and cache sizes from 32 bytes to 32K bytes; *design target miss ratios* are to be used to guide new machine designs. Mean delays per memory reference and memory (bus) traffic rates are computed as a function of line and cache size, and memory access time parameters. We find that for high performance microprocessor designs, line sizes in the range 16-64 bytes seem best; shorter line sizes yield high delays due to memory latency, although they reduce memory traffic somewhat. Longer line sizes are suitable for mainframes because of the higher bandwidth to main memory.**

Index Terms—**Block size, buffer, cache memory, CPU performance, line size, miss ratios.**

I. Introduction

The cache line (block) size is the parameter, along with the overall cache size itself, that most strongly affects the cache performance. Excessively large or small line sizes can raise the miss ratio and greatly increase the storage delay component of average instruction time [18]; also, large line sizes have long transfer times and can create difficulties in multiprocessor systems by creating high levels of memory traffic.

In this paper, we consider the factors that determine the choice of line size, with particular attention to cache miss ratio. We begin, in this section, with a general discussion of the various ways in which the line size affects the cache design and performance, and the ways in which the machine design affects the line size choice. The rest of the paper is then concerned with a detailed examination of how the miss ratio varies with the line size; that study is based on extensive trace driven simulation.

A. Previous Research

There have been a number of surveys of cache memories and/or memory hierarchy performance which have, among other parameters, considered line size choice. The most recent and comprehensive of those surveys are [25] and [26]; earlier ones include [11], [15], [20], [23], and [28].

Some recent papers have looked more carefully at line size but still as one of several issues. Katz *et al.* [16] report that a line size in the range of 16-32 bytes seems best for the SPUR design. In [13] there is a study of line size as one of several parameters in a split (instructions/data/stack) cache; results in that paper are generally similar to those here although less detailed and less clearly specified; insufficient information is provided there to allow us to compare directly. Goodman [12] looks at line size as one of several parameters to vary to minimize bus traffic; we comment later on his results, since they differ from ours.

None of the papers mentioned above systematically focused on the line size issue, and none looked at the topic in enough depth that the results can be relied upon to guide the implementation of a variety of new machines. (In [16] studies are presented to guide SPUR, but are not intended for other machines.) That is one of our goals here.

There does exist one paper which considers primarily the line size issue: [17]. Kumar starts by assuming that the working set size of a program as a function of the block size b is $w(b) = k/b^a$; from that function, the properties of the working set model, and the task switch interval, it is possible to compute the miss ratio as a function of the line size. Kumar found that for a given working set parameter T and a given program, his model held; unfortunately, the value of a was rather sensitive to the program and T; furthermore, only three programs were studied. Those results, therefore, do not adequately address the issue of the effect of the line size on the miss ratio for a variety of line sizes and cache sizes.

Two papers ([14], [27]) look at the use of *sector* or *subblock* caches, in which a line or block (called a *sector* or *address sector* here) is broken up into *subsectors*, (or *transfer subsectors*), only some of which may be cache resident. Those papers conclude that the savings in address tag area make sector caches worthwhile when the subsector size is small; neither, however, looks seriously at the line size choice

Manuscript received June 6, 1985; revised February 25, 1987. This work was supported in part by the National Science Foundation under grant DCR-8202591, by the Defense Advance Research Projects Agency (DoD), under Arpa Order 4871, Monitored by Naval Electronic Systems Command under Contract N00039-84-C-0089, and by the State of California under the MICRO program. Computer time has also been provided by the Stanford Linear Accelerator Center under Department of Energy Contract DE-AC03-76SF-00515.

The author is with the Department of Electrical Engineering and Computer Sciences, Computer Science Division, University of California, Berkeley, CA 94720.

IEEE Log Number 8714523.

problem. (The terms sector/subsector or address sector/transfer subsector reflect the terminology adopted by the P896.2 working group. Other terms used are block/subblock and address block/transfer block.)

Finally, some authors have considered related topics. In [9] and [10], it is suggested that a variable number of lines be loaded into the cache on a miss, when a table indicates that it is desirable. Tan [29] suggests that different line sizes be used for instructions and data, since they have different locality characteristics.

B. Architectural Factors Influencing Line Size Choice

There are a number of factors relating to the design architecture (high level implementation) of machines that influence or determine line size. In this section, we discuss them.

In some machines, principally mainframes, line size has been determined by the *width of the memory modules and the degree of interleaving*. For example, the Amdahl 470V/6 and the IBM 370/168 both used 32 byte lines, since each had four memory banks, each of which was 8 bytes wide. There are two reasons for this: first, there is a slight savings in logic complexity in not having to cycle each memory bank more than once; more generally, short lines often minimize logic complexity. Second, memory latency is substantial and a significant additional delay would be experienced in waiting for the second double word from each memory bank.

Bus protocol tends to strongly influence the line size for microprocessor systems. For example, with a very primitive bus such that for every data item an address must be transmitted, there seems to be very little time saved by asking for more than one data item (e.g., word) on a miss, rather than asking for them as they are needed. Conversely, if one address can elicit several words of data, then a longer line size becomes advantageous. The ability to send or receive several data words in one bus transaction is central to the IEEE Futurebus design [3], [5], which is intended to be a high performance, state-of-the-art bus for use over more than the next decade in bus based systems.

In multiprocessor systems, *memory interference* [4] and *memory busy time* can be an issue. Larger line sizes will increase memory traffic and interference, as is discussed in more detail below; furthermore, a long line will keep the memory and bus busy for a relatively long period, which may affect (interfere with) the operation of other processors or I/O units. With regard to I/O, the phenomenon called *I/O overruns* may occur; this is when the memory system is unable to accept from or supply data as needed to an I/O stream, and thus the I/O operation must be aborted and restarted. I/O overruns can be minimized with sufficient buffering in the I/O data paths, but the transmission of long lines may cause the buffering to be overrun occasionally.

In a cache, lines are found by associative search, and each must be tagged with a (real or virtual) memory address. When the line size is small, the number of *bits of storage required to hold the address tags* can be a major part of the total storage available in the cache; e.g., with a 4 byte line and a 32 bit address space, almost half of the storage is needed for tags. This issue is addressed in [2] and [14]; both papers suggest the use of *sector* or *subblock* caches, in which part of a block (sector) may be cache resident and other parts may be absent; each subsector (subblock) is marked with a bit to indicate whether it is cache resident. Only one address tag is required per sector. Very small blocks seem to be a poor idea because of the storage (VLSI silicon area) required for an address tag for each block.

In many machines, *line crossers* can induce a performance penalty. A *line crosser* is a fetch or write which crosses the boundary between two cache lines. A line crosser in most machines will require two cache accesses, one to each target line; in the Amdahl 470V/6, the penalty is 65 ns for a read and 97.5 ns for a write [22]. If the memory addresses of line crossers are randomly distributed, then doubling the line size halves the frequency of line crossers. (Some machines require that instructions be aligned on 2 or 4 byte boundaries, which either reduces the frequency of or eliminates line crossers. It is easier to implement a machine which does not permit line crossers.)

With a *copy-back* cache in which an entire dirty line is copied back, a larger line size will increase memory traffic per copy back. (Also, a longer line is more likely to be dirty.) Increased copy-back memory traffic can be avoided if the larger line size has caused the miss ratio to drop enough to compensate; if dirty bits are maintained for partial lines and only the dirty portions are recopied, then further savings in memory traffic are possible, although at the cost of additional logic complexity. (Writing back only dirty subblocks may not help if each subblock requires an address cycle.)

C. Miss Ratio Factors

The major effect of line size choice on performance comes from its impact on the miss ratio; we concentrate on that in the remainder of this paper.

As will be shown later, increasing the line size generally decreases the miss ratio; getting more data on each fetch means that fewer fetches are required. When the line size becomes large enough, however, and starts to approach the cache size, increasing line size can lead to an increased miss ratio. This is because a program is concurrently referencing some number of contiguous areas of its address space; these areas collectively are known as the program *locality*. When all of those areas cannot be resident in the cache at the same time, as when the line size becomes large with respect to the cache size, the miss ratio increases. More generally, the larger the line size, the greater the degree of *memory pollution*; memory pollution occurs when either through prefetch or through the use of a large block size, material is loaded which is not referenced [24]. Memory pollution can have the effect of increasing the miss ratio by displacing from memory information which will be referenced again (prior to a reference to the information which displaced it).

In many machines, it is possible to describe the time to fetch a line as $a + c(L/d)$, where a and c are constants, L is the line size, and d is the data path width to memory. Here, a is the constant delay for any memory transaction, consisting primarily of memory latency and address transmission time. c

is the additional time per "buswidth" of data transmitted; if the data path is 4 bytes wide, then c is the cycle time of the bus, when one word is transmitted per bus cycle. If we let $m(L)$ by the miss ratio (for a given cache size) as a function of the line size, then the value of L that minimizes $(a + c(L/d))m(L)$ is a function of a/c only, and not the value of either individually. If a/c is low, then transfer time dominates latency, and short lines are favored; when $a/c \gg 1$, latency dominates transfer, and long lines tend to be favored. Since the primary performance penalty from a cache miss is in the delay to perform a fetch, the *fetch time* is the statistic of interest. Note that high performance machines may use *fetch bypass*, by which the target of a fetch is obtained first and the rest of the line is loaded into the cache later; in that case the transfer time costs not in the delay for that fetch but in other ways: the cache may be busy and other accesses are delayed, and the memory and bus may be busy and delay this or other processors.

Another penalty of a high miss ratio is the memory interference created in a multiprocessor system in which the path (e.g., the bus) to main memory is the limiting resource in the system. In existing or proposed multimicroprocessor systems, that is the case, and minimizing bus traffic is a very important goal; bus traffic per miss clearly increases with line size. It is worth noting, however, that the penalty is measured not in the number of bytes transferred but in the period of time that the bus is unavailable, which is a function of the bus protocol and is generally of the same form $a + c(L/d)$.

D. Overview

As noted earlier, the remainder of this paper will be concerned with an experimental investigation of the effect of the line size on the cache miss ratio. Our experimental methodology is that of trace driven simulation. In Section II, we describe our simulations and the traces used. The simulation results appear in Section III. Those results are used to establish *design target miss ratios* in Section IV. A discussion of appropriate line size choices is given in Section V, in which we comment on the performance to be expected from some new microprocessor designs.

II. Methodology

Our analysis of how miss ratio changes with varying line size is based on extensive trace driven simulation, as we explain here.

A. Trace Driven Simulation

A *program address trace* is a trace of the sequence of (virtual) addresses accessed by a computer program or programs. *Trace driven simulation* involves driving a simulation model of a system with an external trace of events rather than with a random number generator. Trace driven simulation is a very good way to study many aspects of cache design and performance, for a number of reasons. First, it is superior to either pure mathematical models or random number driven simulation because there does not currently exist any generally accepted or believable models for those characteristics of program behavior that determine cache performance; thus it is not possible to specify a realistic model nor to drive a simulator with a good representation of a program. A trace properly represents at least one real program, and in certain respects can be expected to drive the simulator correctly.

A simulator is also much better in many ways than the construction of prototype designs. It is far faster to build a simulator, and the design being simulated can be varied easily, sometimes by just changing an input parameter. Conversely, a hardware prototype can require man-years to build and can be varied little if at all. In the case of the study presented here, a hardware prototype with the requisite flexibility is infeasible.

There are a number of ways in which trace driven simulation is less than perfect; among them are 1) traces are only small workload samples and may not be representative; 2) operating system behavior is seldom sufficiently represented, nor is task switching; 3) input/output activity is not usually included; and 4) computer time limitations prevent the use of traces long enough to make use of (fill up) large caches. These issues and others are considered in more detail in [27]. We believe that for the purposes of this paper, there are two problems with trace driven simulation. First, measured miss ratios on real machines running production workloads are almost always higher than trace measurements would predict. Second, the limited length of our simulations and our use of cache flushing to simulate task switching mean that we never fill large caches, and thus our results do not extend beyond 32K byte caches. As explained below in more detail, we have emphasized in our experiments the relative changes in performance with line size, rather than considering the absolute level of the miss ratio. Evidence is presented later which supports that choice.

B. The Traces

A number of program address traces are available to the author for memory hierarchy simulations. 27 traces were selected as a representative sample, five from the Zilog Z8000, seven from the DEC VAX 11/780, four from the Motorola 68000, three from the CDC 6400, and eight from the IBM 370. The traces used are a subset (about half) of those used in [27]; we were not able to use more because of the large number of experiments for each trace. Various characteristics of the traces are given in [27], where we discuss them in more detail; see also [25].

Seven of the *IBM 370* traces were for user programs: the traces were FCOMP1 [Fortran compile of program that solves Reynolds partial differential equations (2330 lines)], CCOMP1 (Cobol compile, 240 lines, accounting report), FGO1 (Fortran Go [execution] step, factor analysis, 1249 lines, single precision), FGO2 (Fortran Go step, double precision analysis of satellite information, 2057 lines, FortG compiler), CGO1 (Cobol Go step, fixed assets program doing tax transaction selection), CGO2 (Cobol Go step, fixed assets, year end tax select), and CGO3 (Cobol Go step, projects depreciation of fixed assets). The eighth IBM 370 trace was of the MVS operating system; that is the MVS1 trace, and was part of a standard Amdahl Corporation workload. Each IBM

370 trace presupposes a memory with a 4 byte interface. Somewhat less than half of the memory references were instructions.

Five traces for the *Zilog Z8000* microprocessor were also analyzed. Each trace is for a program which is part of the UNIX system software. These Z8000 traces are ZOD (octal dump), ZSORT (sort program), ZVI (screen editor), ZGREP (search a file for a pattern), and ZPR (produces a printed listing of one or more files). A 2 byte memory interface is assumed. Over 75 percent of the memory references were instruction fetches.

The third set of traces was for the *DEC VAX 11/780.* These traces were VCCOM (the C compiler compiling a C program of 125 lines, written in C), VTROFF (the phototypesetter text formatting system, written in C), VPUZZLE (the well known "puzzle" program; used by Baskett to test raw CPU power, written in C), VOTMDL (parser/constructor, written in Pascal, uses set operations), VSPICE (the Spice circuit simulator, written in Fortran), LISPC (the Lisp compiler, written in Lisp) and VAXIMA (a symbolic manipulation system, derived from Macsyma, written in Lisp). The LISPC and VAXIMA traces are quite long, and sections 8 and 13 of those traces were used, respectively (LISP8, VAXIMA13). (Sections 8 and 13 were chosen since their characteristics seem to be representative of the traces as a whole; there is otherwise nothing special about those sections. See [27] for more information about the trace sections.) A four-byte memory interface is assumed. Approximately half of the memory addresses were instruction fetches.

The fourth set of traces is for the *CDC 6400*: TWOD1 (Fortran Go of a program that solves the two-dimensional scattering problem of an infinite circular cylinder), PPAL (Fortran Go of a phase plane analysis program solving a set of two simultaneous differential equations, excluding startup portion of program), and DPOLE (Fortran Go of a program that solves a three-dimensional scattering problem for a cube using the dipole approximation technique). These traces assume a one-word (60 bit) memory interface for data and a one-instruction (15 or 30 bits) interface for instructions; i.e., there is no memory in the instruction interface. Because all data references were for 60 bits (treated as 8 bytes), no results were obtained for less than 8 byte lines when using the CDC 6400 traces.

The last set of traces was for the *Motorola 68000.* These traces are M68MATCH (a pattern matching program, written in Pascal, taken from the Kernighan and Plauger book, *Software Tools in Pascal*), M68PL0 (the PL0 compiler from Wirth's book *Algorithms + Data Structures = Programs*), M68QSORT (recursive quicksort), and M68STAT (a trace statistics program). Each trace is only 30 000 memory addresses long, and distinguishes only reads and writes, but does not distinguish instruction fetches from data reads. The trace was gathered from a development system using hardware monitoring techniques and, of course, represents a 16 bit bus. Because instruction fetches and data references are not identified, no results were obtained for instruction caches or data caches for the 68000 traces; only a unified cache was considered.

C. The Simulator, the Simulations, and the Simulation Workloads

A large trace-driven simulation program was used to generate the results presented here. *LRU* stack techniques were used to generate miss ratios for all cache sizes simultaneously. A *demand fetch, fetch on write, copy-back cache with LRU replacement* is assumed. All caches are *fully associative*, except for those with 4 and 8 byte line sizes, which use 4 sets and are associative within each set. (The simulation time became too large when using fully associative mapping and such small line sizes; i.e., the stack became too deep.) The degree of associativity was kept as high as possible in order to remove associativity factors from the results.

Eight different *simulation workloads* were constructed. The first consisted of four of the IBM 370 traces: FGO1, FGO2, FCOMP1, and CCOMP1; that workload should be reasonably representative of an *IBM 370 scientific environment*, and consists of two compiles (albeit one of those is for a Cobol compiler) and two Fortran executions. The second is for an *IBM 370 commercial system* and consists of three Cobol executions: CGO1, CGO2, and CGO3. A *370 operating system* load is the third workload: the MVS1 trace. A *VAX running Unix Fortran, Pascal, and C programs* is the fourth workload: VCCOM, VSPICE, VOTMDL, VPUZZLE, and VTROFF. *LISP in a VAX environment* is the fifth workload: LISP8 and VAXIMA13. Five *Z8000 traces, all Unix utilities*, are the sixth workload. *Three CDC 6400 programs* (TWOD1, PPAL, DPOLE) are the seventh workload and the *four M68000 programs* are the eight workload. The total number of memory references processed for each workload, and the number of bytes in the lines referenced are shown for each case in Table I. (Note that because of the large number of simulation runs, computer time limitations meant that each run was kept relatively short; most were between 240 000 and 500 000 memory references. Thus, for each trace, only the initial part was used. This may distort the results somewhat, in some unknown way.)

Each simulation run used the various traces in a round-robin fashion, purging the cache each time the trace was switched, in order to simulate multiprogramming. (For the 370 OS workload, the same trace was restarted after each purge.) It would have also been possible to not purge when task switching, but that makes the results very sensitive to the number of programs used; such effects are visible in the plots in [25]. The result of our task switching is thus to average the behavior of the traces within each workload. The switch interval was 20 000 memory references in each case except for the M68000 traces, where the switch interval was 15 000. We note that our results are somewhat sensitive to the task switch (purge) interval in any case, since the more frequent the purges, the greater the advantages to large lines: they load the cache more quickly after a purge, and the effect of memory pollution is reduced since the cache is less frequently full.

Two sets of simulations were conducted in each case, except for the M68000 traces. In the first set, the cache design was a unified cache, i.e., it contained both instructions and data; miss ratios were computed over all memory references. In the second set of simulations, the cache was partitioned into

TABLE I

Traces	Line Size	Address Space Size - Bytes Total	Instructions	Data	Number of Memory Addresses
FGO1	4	68828	28192	40636	240,000
FGO2	8	76152	31632	44520	
FCOMP1	16	84832	35504	49325	
CCOMP1	32	95840	40640	55200	
	64	110912	46848	64064	
	128	132864	55552	77312	
CGO1	4	86524	16464	70060	240,000
CGO2	8	92960	19496	73464	
CGO3	16	100160	21840	78320	
	32	107360	24832	82528	
	64	114816	28160	86656	
	128	124800	32384	92416	
MVS1	4	52396	17728	34668	240,000
	8	57392	19568	37824	
	16	65376	22096	43280	
	32	76896	25312	51584	
	64	92608	28928	63680	
	128	115328	33280	82048	
VCCOM	4	51420	18508	32912	500,000
VSPICE	8	56968	19424	37544	
VOTMD1	16	62672	20752	41920	
VPUZZLE	32	70080	22368	47712	
VTROFF	64	79552	23616	55936	
	128	93440	23680	69760	
LISP8	4	52584	37404	15180	240,000
VAXIMA13	8	59768	40080	19688	
	16	72688	44672	28016	
	32	91200	50112	41088	
	64	120000	56320	63680	
	128	166016	63104	102912	
ZVI	4	22212	14960	7252	500,000
ZSORT	8	23936	16176	7760	
ZGREP	16	26704	18128	8576	
ZPR	32	30816	21120	9696	
ZOD	64	36416	25280	11136	
	128	43648	29568	14080	
CDCTWOD	4		25432		500,000
CDCPPAL	8	51576	26328	25248	
CDCDPOLE	16	56416	26784	29632	
	32	61632	27328	34304	
	64	66880	28544	38336	
	128	83584	30848	52736	
M68MATCH	4	10472			120,000
M68PL0	8	10880			
M68QSORT	16	11472			
M68STAT	32	12352			
	64	13312			
	128	15488			

instruction and data halves; the miss ratio for the instruction half was the miss ratio for instructions only, and likewise for the data half. Since the M68000 traces were not tagged by instructions or data, only a unified cache was simulated for those traces.

III. Simulation Results

A. Miss Ratios

The miss ratios for each cache size, line size, and workload were determined, and an *average miss ratio* was computed, for given cache and line sizes, over the various workloads. The averages were obtained by averaging the individual simulation results, and they are shown in Figs. 1, 2, and 3, respectively, for a unified cache (instructions and data), and separately for an instruction cache and a data cache. As can be seen in those figures, the miss ratio is generally declining, except when the line size is close to the cache size. Also, as one might expect, the drop in the miss ratio with increasing line size is most pronounced for instructions and least evident for data. This is because instructions are much more likely to be used sequentially, although there is sequentiality in data reference patterns as well, due to the clustering of related variables in storage and sequential use of array elements.

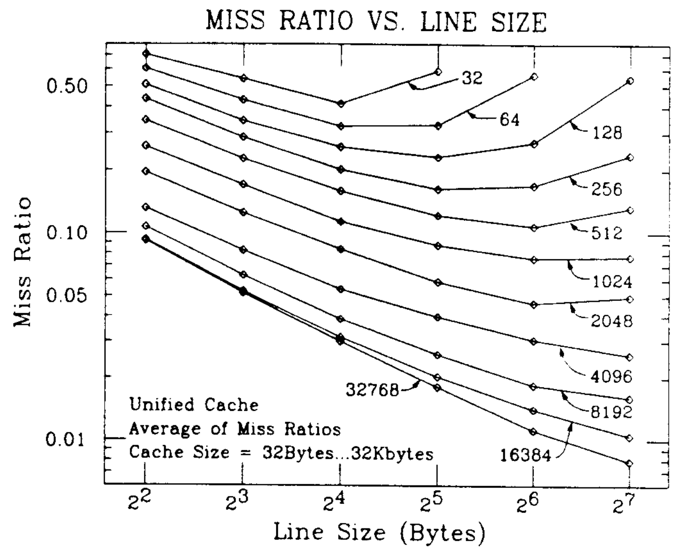

Fig. 1. Average miss ratio for a unified (combined instruction/data) cache, averaged over all simulation runs. Cache size varies from 32 to 32K bytes.

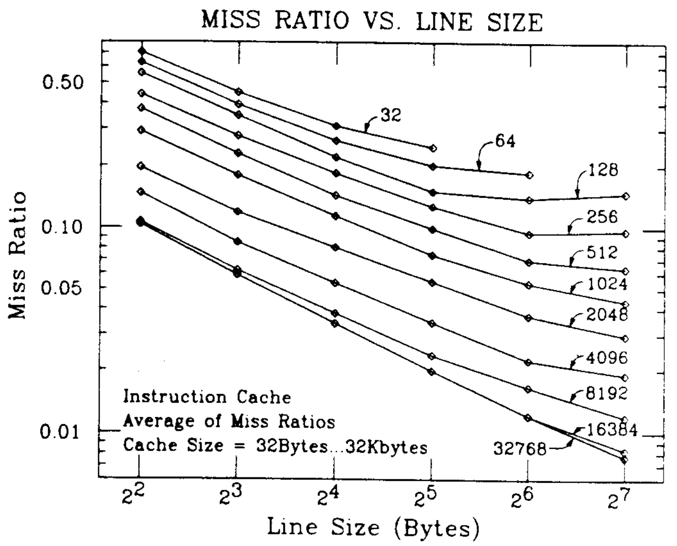

Fig. 2. Average miss ratio for an instruction cache, averaged over all simulation runs. Cache size varies from 32 to 32K bytes.

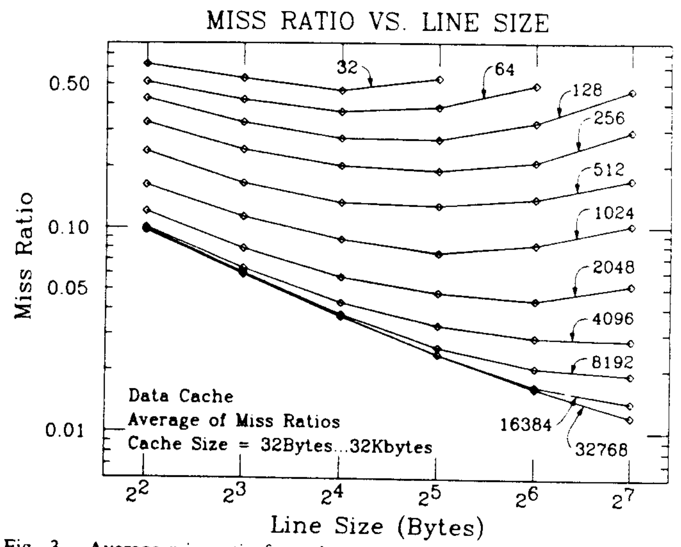

Fig. 3. Average miss ratio for a data cache, averaged over all simulation runs. Cache size varies from 32 to 32K bytes.

It has been observed elsewhere [27] that there is enormous variability among traces in their measured miss ratios; miss ratios are quite workload dependent. To demonstrate that here, we computed the *coefficient of variation* for each data point plotted in Figs. 1-3. Each entry in Figs. 1-3 is the average of 6-8 simulation points (for the M68000 traces, there were no instruction or data cache runs, so there are only six data points for instruction and data caches), and the coefficient of variation is the ratio of the standard deviation to the mean; a zero coefficient of variation implies a constant value, and a large coefficient of variation means that there is a high degree of variability. The average coefficient of variation, over all 189 entries was 0.58, which is rather large. The variation in the miss ratio in our simulations, for example, is from 2.5 to 20.5 percent for a 32K byte cache with a 4 byte line.

B. Ratio of Miss Ratios

Because of the high variability in the miss ratios, and because it is well known that observed miss ratios are usually higher than those obtained from trace driven simulation [27], we decided to compute the relative change in the miss ratio as a function of line size; as will be shown, this statistic is considerably more stable than the miss ratio.

For each simulation, cache size and line size, the ratio of the miss ratio at that cache size and line size to that for the same cache size but half the line size was computed (i.e., for a 1K cache and a 16 byte line, the value obtained is the ratio of the miss ratio for a 1K cache and 16 byte line to that for a 1K cache and an 8 byte line.) Those values are called *ratio of miss ratios or ratio of ratios*, and their averages over the various simulations are given in Table II. (The average here is the average of the ratios. It is also possible to take the ratio of the averages, which yields similar but not identical numbers.)

The average *coefficient of variation* was also computed for the entries in Table II; the value obtained was 0.13, which is less than 22 percent of the value (0.58) obtained for the miss ratio averages. Our expectation that this ratio of ratios would be considerably more stable than the miss ratio has been shown to be correct.

The ratio of miss ratios (from Table II) for a unified cache has been plotted in Fig. 4. As can be seen, the values "bounce around" a fair amount; this irregularity in the data is due to the fact that these values are averages over only a small number of samples (6-8 simulations per data point, 27 traces used). Abrupt changes in the miss ratio can be observed from the data for each individual simulation, when a frequently executed loop does or does not fit in the cache, as the line size changes. We do not feel that the values presented in Table II are sacred or exact, and believe that the utility of these data would be enhanced by smoothing out the irregularities. We have "smoothed" the data by eye; the smoothed ratio of ratios data appears in Table III. (This is similar to casting out outliers, and is *not* an attempt to introduce a "fudge factor.") These smoothed data will be used later to derive our design target miss ratios.

It is worth looking at the values in Tables II and III. We note that most of the values are between 0.5 and 1.0; this means that each doubling of the line size decreases the miss ratio, but never by as much as 50 percent. Since a line twice as long means that twice as much data are fetched per miss, it is also clear that each increase in the line size also increases the memory fetch traffic.

The values that appear in Table II and Fig. 4 are sufficiently regular that we judged it worthwhile to *fit those data with a continuous curve*; such a mathematical fit can be used for optimization studies.

TABLE II
AVERAGE OVER ALL SIMULATIONS
RATIO OF MISS RATIO TO THAT FOR LINE HALF AS LARGE

Cache Type: Unified	Line Size				
Size	8	16	32	64	128
32	0.775	0.761	1.444		
64	0.711	0.762	1.024	1.740	
128	0.664	0.783	0.942	1.184	2.046
256	0.653	0.714	0.822	1.162	1.453
512	0.654	0.693	0.767	0.914	1.451
1024	0.653	0.691	0.831	0.880	1.023
2048	0.636	0.678	0.731	0.787	1.132
4096	0.586	0.622	0.719	0.746	0.809
8192	0.581	0.593	0.645	0.661	0.753
16384	0.569	0.581	0.616	0.633	0.685
32768	0.564	0.575	0.601	0.601	0.660
Cache Type: Instructions					
32	0.647	0.688	0.749		
64	0.655	0.691	0.751	0.838	
128	0.645	0.659	0.687	0.955	0.929
256	0.694	0.676	0.723	0.754	0.931
512	0.664	0.684	0.736	0.705	0.910
1024	0.656	0.682	0.693	0.790	0.801
2048	0.619	0.697	0.737	0.701	0.832
4096	0.586	0.633	0.651	0.660	0.831
8192	0.581	0.593	0.598	0.667	0.690
16384	0.573	0.578	0.581	0.624	0.657
32768	0.573	0.578	0.581	0.624	0.634
Cache Type: Data					
32	0.836	0.884	1.151		
64	0.781	0.873	1.143	1.284	
128	0.734	0.835	1.004	1.328	1.439
256	0.718	0.853	0.944	1.124	1.577
512	0.712	0.814	0.956	1.122	1.314
1024	0.744	0.746	0.836	1.015	1.499
2048	0.660	0.711	0.787	0.853	1.071
4096	0.619	0.654	0.727	0.770	0.835
8192	0.603	0.620	0.667	0.723	0.817
16384	0.602	0.618	0.646	0.672	0.747
32768	0.602	0.618	0.646	0.662	0.697

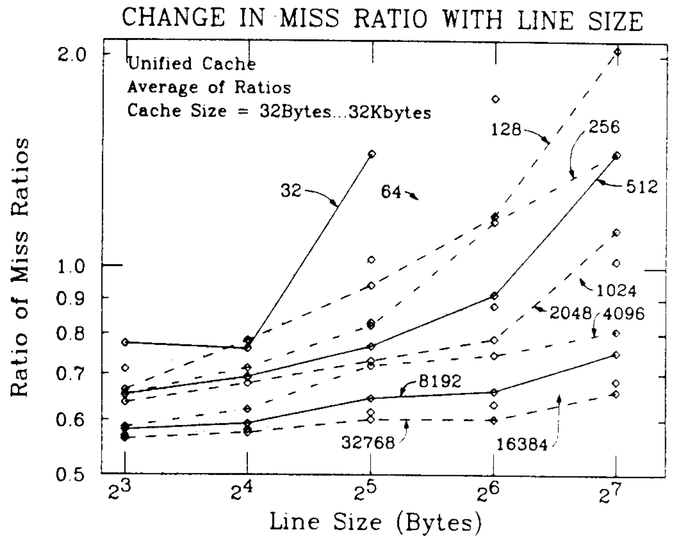

Fig. 4. Change in miss ratio with line size. Each point gives the ratio of the miss ratio for that line size and cache size to the miss ratio for the same cache size and a line size half as large. Figures are averages of all simulation runs, for a unified cache.

TABLE III
SMOOTHED AVERAGE OF RATIOS
RATIO OF MISS RATIO TO THAT FOR LINE HALF AS LARGE

Cache Type: Unified	Line Size				
Size	8	16	32	64	128
32	0.775	0.900	1.500		
64	0.711	0.820	1.200	1.500	
128	0.692	0.750	0.942	1.300	1.600
256	0.653	0.714	0.860	1.070	1.400
512	0.654	0.693	0.800	0.914	1.300
1024	0.653	0.680	0.770	0.850	1.100
2048	0.636	0.660	0.731	0.787	0.950
4096	0.586	0.622	0.680	0.720	0.850
8192	0.581	0.593	0.630	0.661	0.753
16384	0.569	0.581	0.600	0.633	0.685
32768	0.564	0.575	0.590	0.601	0.660
Cache Type: Instructions					
32	0.660	0.690	0.749		
64	0.650	0.685	0.740	0.860	
128	0.645	0.680	0.730	0.830	0.960
256	0.630	0.670	0.710	0.780	0.931
512	0.620	0.660	0.690	0.750	0.910
1024	0.610	0.650	0.670	0.730	0.860
2048	0.600	0.640	0.650	0.701	0.832
4096	0.595	0.620	0.630	0.680	0.750
8192	0.581	0.600	0.610	0.640	0.690
16384	0.577	0.585	0.589	0.620	0.657
32768	0.573	0.578	0.581	0.590	0.634
Cache Type: Data					
32	0.836	0.900	1.300		
64	0.781	0.873	1.100	1.400	
128	0.734	0.850	1.004	1.328	1.450
256	0.718	0.830	0.970	1.200	1.400
512	0.712	0.814	0.956	1.122	1.314
1024	0.744	0.760	0.860	1.015	1.150
2048	0.660	0.711	0.787	0.880	1.071
4096	0.619	0.654	0.700	0.770	0.900
8192	0.603	0.620	0.667	0.723	0.817
16384	0.602	0.618	0.646	0.672	0.747
32768	0.602	0.618	0.630	0.662	0.697

TABLE IV
FITTED RATIO OF RATIOS

Cache Type: Unified	Line Size				
Size	8	16	32	64	128
32	0.75	0.95	1.34		
64	0.70	0.85	1.08	1.64	
128	0.66	0.78	0.94	1.24	2.15
256	0.64	0.73	0.86	1.04	1.44
512	0.61	0.70	0.80	0.93	1.17
1024	0.60	0.67	0.76	0.86	1.02
2048	0.58	0.65	0.72	0.81	0.92
8192	0.56	0.62	0.68	0.74	0.82
16384	0.55	0.61	0.66	0.72	0.78
32768	0.54	0.60	0.65	0.70	0.75
Cache Type: Instructions					
32	0.66	0.72	0.80		
64	0.64	0.70	0.77	0.87	
128	0.63	0.68	0.74	0.83	0.97
256	0.62	0.67	0.72	0.79	0.90
512	0.61	0.65	0.70	0.76	0.85
1024	0.6	0.64	0.68	0.74	0.81
2048	0.59	0.63	0.67	0.72	0.78
4096	0.59	0.62	0.66	0.70	0.75
8192	0.58	0.61	0.65	0.68	0.73
16384	0.57	0.61	0.64	0.67	0.71
32768	0.57	0.60	0.63	0.66	0.70
Cache Type: Data					
32	0.82	0.95	1.16		
64	0.78	0.89	1.05	1.35	
128	0.76	0.85	0.97	1.19	1.67
256	0.73	0.81	0.91	1.08	1.39
512	0.71	0.78	0.87	1.00	1.22
1024	0.69	0.75	0.83	0.93	1.10
2048	0.68	0.73	0.80	0.89	1.02
4096	0.66	0.71	0.77	0.85	0.95
8192	0.65	0.70	0.75	0.81	0.90
16384	0.64	0.68	0.73	0.79	0.86
32768	0.63	0.67	0.71	0.76	0.83

The fitting process was purely empirical. A variety of functional forms was postulated, since there is no statistical method to pick the correct functional form, and in each case the parameters were varied to minimize the mean square error (i.e., the square of the differences between the fitted and fitting data were summed. The parameters $a \cdots f$ were varied until that sum was minimized.) We kept trying new functional forms until we could not reduce the mean square error without adding an unreasonable number of additional parameters; there was no particular "theory" underlying the choice of functional forms. The functional form which was found to yield the best fit is

$$e^*(1 + a/(b + f^* \log C\text{size-loglinesize}^* \cdot (1 + c^*(\text{loglinesize-2})^d)$$

where $\log C$size is the $\log_2$ of the cache size in bytes and loglinesize is the $\log_2$ of the line size in bytes. The parameter values found to minimize the mean square error in each case are

cache type	error	a	b	c	d	e	f
unified cache	0.295	2.656	4.197	2.357	0.247	0.113	0.667
inst. cache	0.097	2.639	7.502	2.818	0.093	0.113	0.443
data cache	0.471	3.772	6.210	3.318	0.088	0.099	0.476

where "error" is the sum of the squares of the errors, as explained above. Note that the fit was done only over the range of line sizes 4–128 bytes and cache sizes 32 bytes–32K bytes; we do not expect that the fitted curves will have the proper behavior outside of that range.

The fitted values appear in Table IV.

IV. DESIGN TARGETS

One of the goals of this paper is to provide for the computer system designer a set of numbers that he or she can use to estimate the performance impact of certain design choices. Specifically, we would like to present realistic values for the miss ratio as a function of cache size and line size. This same task was undertaken in [27], but only for the single line size of 16 bytes, and a set of *design target miss ratios (DTMR)* was proposed there for unified caches, and instruction and data caches, over the range of cache sizes of 32 bytes to 64K bytes. In that paper, the DTMR's were derived by examining trace driven simulations from 57 traces (including those used here), and combining those data with other, real, measurements reported in the literature. We note that the miss ratios observed in practice will be very workload dependent and highly variable, and will be unlikely to match our DTMR's exactly; the DTMR's are provided for two reasons: a) to make available miss ratios around which to design, in the absence of better data; b) to provide the customer of a product with numbers independent of the vendor and his marketing department. Our earlier DTMR's were validated against published measurements taken from hardware monitors to the extent possible; one of our data points calculated here is validated against published measurements in Section V.

TABLE V
DESIGN TARGET MISS RATIOS

Cache Type: Unified	Miss Ratio					
	Line Size:					
Size	4	8	16	32	64	128
32	0.717	0.556	0.500	0.750		
64	0.686	0.488	0.400	0.480	0.720	
128	0.674	0.467	0.350	0.330	0.428	0.686
256	0.643	0.420	0.300	0.258	0.276	0.386
512	0.596	0.390	0.270	0.216	0.197	0.257
1024	0.473	0.309	0.210	0.162	0.137	0.151
2048	0.405	0.258	0.170	0.124	0.098	0.093
4096	0.329	0.193	0.120	0.082	0.059	0.050
8192	0.232	0.135	0.080	0.050	0.033	0.025
16384	0.182	0.103	0.060	0.036	0.023	0.016
32768	0.124	0.070	0.040	0.024	0.014	0.009
Cache Type: Instructions						
32	0.725	0.478	0.330	0.247		
64	0.674	0.438	0.300	0.222	0.191	
128	0.615	0.397	0.270	0.197	0.164	0.157
256	0.592	0.373	0.250	0.177	0.138	0.129
512	0.562	0.348	0.230	0.159	0.119	0.108
1024	0.504	0.308	0.200	0.134	0.098	0.084
2048	0.391	0.234	0.150	0.098	0.068	0.057
4096	0.271	0.161	0.100	0.063	0.043	0.032
8192	0.172	0.100	0.060	0.037	0.023	0.016
16384	0.148	0.085	0.050	0.029	0.018	0.012
32768	0.091	0.052	0.030	0.017	0.010	0.007
Cache Type: Data						
32	0.731	0.611	0.550	0.715		
64	0.660	0.515	0.450	0.495	0.693	
128	0.561	0.412	0.350	0.351	0.467	0.677
256	0.470	0.337	0.280	0.272	0.326	0.456
512	0.345	0.246	0.200	0.191	0.215	0.282
1024	0.283	0.211	0.160	0.138	0.140	0.161
2048	0.256	0.169	0.120	0.094	0.083	0.089
4096	0.247	0.153	0.100	0.070	0.054	0.048
8192	0.214	0.129	0.080	0.053	0.039	0.032
16384	0.161	0.097	0.060	0.039	0.026	0.019
32768	0.108	0.065	0.040	0.025	0.017	0.012

A. Design Target Miss Ratios

The ratio of ratios data presented in Tables II and III can be used to compute the miss ratio for one line size from the miss ratio (for that cache size) for another line size. In our case, we compute the miss ratios for line sizes of 4, 8, 32, 64, and 128 bytes from our design target miss ratios for 16 byte lines. The results of applying the ratios from Table III (the smoothed ratios) to the 16-byte line DTMR's from [27] are shown in Table V and Figs. 5-7. (The values obtained by using the unsmoothed ratios are very similar, but are somewhat less regular.) In Section V, we discuss how our predictions compare to parameter choices for real systems.

B. Design Target Bus (Memory) Traffic

As noted earlier, bus bandwidth can be the limiting resource in a multimicroprocessor computer system, and thus memory traffic is a very significant performance factor. Memory traffic consists of two components: fetch traffic and write or copy-back traffic. The former may be calculated by multiplying the miss ratio by the line size to get traffic in bytes/memory reference, and the fetch traffic is shown in Table VI (based on Table V). Again, we note that in all cases, memory fetch traffic increases with line size.

The traffic in the other direction, from cache to main memory, will depend on whether the cache uses write through or copy back. With write through, every word or double word is written immediately back to main memory at the time of the

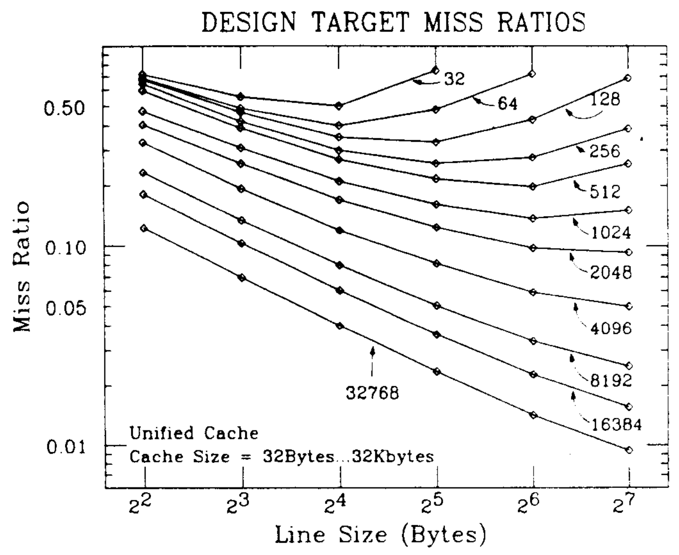

Fig. 5. Design target miss ratios, for unified (instruction/data) caches of 32-32K bytes and line sizes of 4-128 bytes.

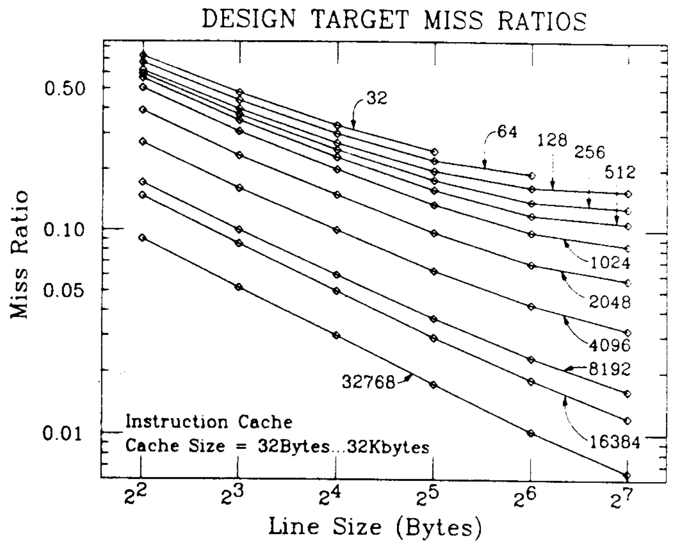

Fig. 6. Design target miss ratios, for instruction caches of 32-32K bytes and line sizes of 4-128 bytes.

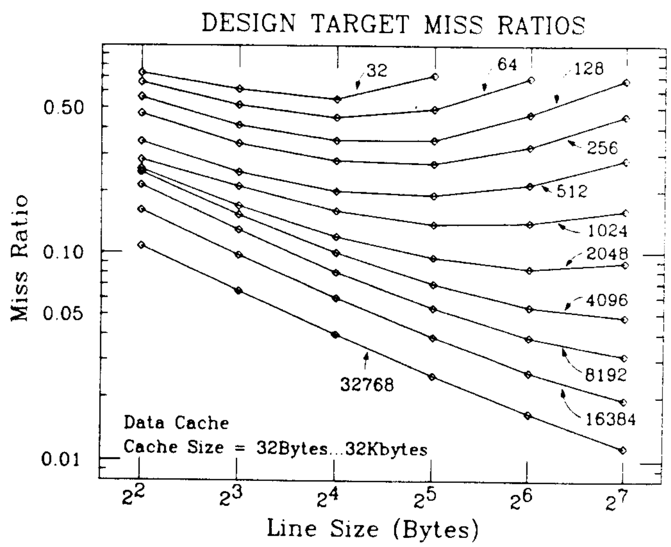

Fig. 7. Design target miss ratios, for data caches of 32-32K bytes and line sizes of 4-128 bytes.

TABLE VI
DESIGN TARGET BUS FETCH TRAFFIC (BYTES/CACHE REFERENCE)

Cache Type: Unified	Memory Traffic (bytes/cache reference)					
	Line Size:					
Size	4	8	16	32	64	128
32	2.9	4.4	8.0	24.0		
64	2.7	3.9	6.4	15.4	46.1	
128	2.7	3.7	5.6	10.5	27.4	87.7
256	2.6	3.4	4.8	8.3	17.7	49.5
512	2.4	3.1	4.3	6.9	12.6	32.9
1024	1.9	2.5	3.4	5.2	8.8	19.4
2048	1.6	2.1	2.7	4.0	6.3	11.9
4096	1.3	1.5	1.9	2.6	3.8	6.4
8192	0.9	1.1	1.3	1.6	2.1	3.2
16384	0.7	0.8	1.0	1.2	1.5	2.0
32768	0.5	0.6	0.6	0.8	0.9	1.2
Cache Type: Instructions						
32	2.9	3.8	5.3	7.9		
64	2.7	3.5	4.8	7.1	12.2	
128	2.5	3.2	4.3	6.3	10.5	20.1
256	2.4	3.0	4.0	5.7	8.9	16.5
512	2.2	2.8	3.7	5.1	7.6	13.9
1024	2.0	2.5	3.2	4.3	6.3	10.8
2048	1.6	1.9	2.4	3.1	4.4	7.3
4096	1.1	1.3	1.6	2.0	2.7	4.1
8192	0.7	0.8	1.0	1.2	1.5	2.1
16384	0.6	0.7	0.8	0.9	1.2	1.5
32768	0.4	0.4	0.5	0.6	0.7	0.8
Cache Type: Data						
32	2.9	4.9	8.8	22.9		
64	2.6	4.1	7.2	15.8	44.4	
128	2.2	3.3	5.6	11.2	29.9	86.6
256	1.9	2.7	4.5	8.7	20.9	58.4
512	1.4	2.0	3.2	6.1	13.7	36.1
1024	1.1	1.7	2.6	4.4	8.9	20.6
2048	1.0	1.4	1.9	3.0	5.3	11.4
4096	1.0	1.2	1.6	2.2	3.4	6.2
8192	0.9	1.0	1.3	1.7	2.5	4.0
16384	0.6	0.8	1.0	1.2	1.7	2.5
32768	0.4	0.5	0.6	0.8	1.1	1.5

write; in that case, the write traffic is invariant with cache size and line size, and a constant may be added to each entry in Table VI. Frequencies of writes for 57 traces appear in [27] and those data may be used to make realistic estimates of write traffic. *For architectures such as the IBM 370 and the DEC VAX, about half of the memory references are data accesses*, and for most architectures studied in [27], *about one third of the data references were writes.*

In the case of copy back, it has been found [27] *that about half of the data lines pushed are dirty*. Thus, the traffic rates for the data cache can be increased by 50 percent to reflect copy back. The instruction cache traffic rate remains unaffected, since modifications to the instruction stream should be extremely rare. The traffic rate for the unified cache can be estimated by breaking down the misses into the instruction portion and data portion, and then multiplying the latter by 1.5 to represent data line pushes. (Note that in equilibrium, the number of data line fetches will match the number of data lines replaced.)

C. Instruction Delays Due to Cache Misses

As explained above, the primary importance of the miss ratio lies in the delays in loading the cache from main memory. For many or most machine designs, the time to service a miss can be represented as a function of the form $a + c(L/d)$, where a and c are constants, d is the data path width, and L is the line size. The mean delay per memory reference can be computed by multiplying our DTMR's by the delay for

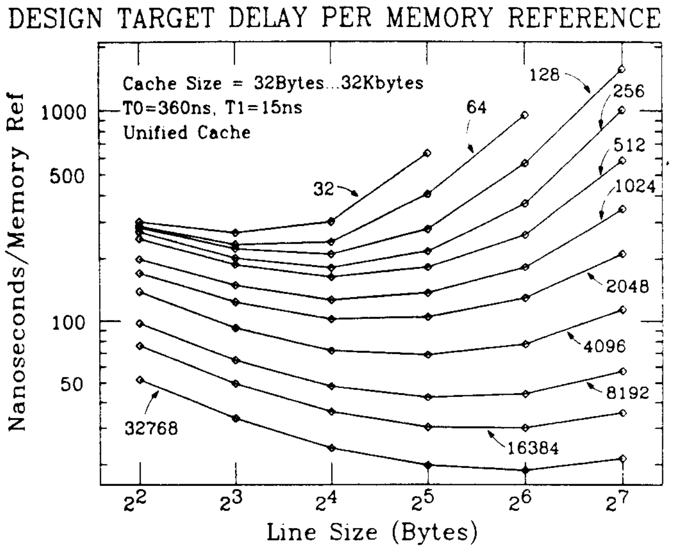

Fig. 8. Average memory delay per memory reference, based on design target miss ratios for a unified cache. Cache size from 32 bytes to 32K bytes, and line size from 4 to 128 bytes. Memory access latency ($T0$) is 360 ns; additional time for each byte of the line is 15 ns ($T1$).

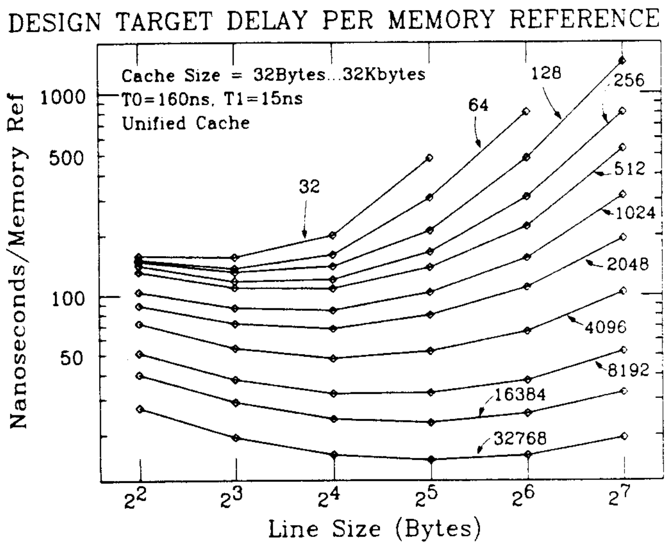

Fig. 9. Average memory delay per memory reference, based on design target miss ratios for a unified cache. Cache size from 32 bytes to 32K bytes, and line size from 4 to 128 bytes. Memory access latency ($T0$) is 160 ns; additional time for each byte of the line is 15 ns ($T1$).

a miss. We will refer to the miss service time as $T0 + b*T1$ where $T0$ is the constant overhead for a miss (address cycle and latency), b is the number of bytes in the line, and $T1$ is the additional transfer time per byte. The labels $T0$ and $T1$ are used in Figs. 8-10. As noted earlier, in Section I-C, the line size that minimizes the mean memory reference delay is a function of a/c or $T0/T1$.

We have computed the mean delay per memory reference for three different sets of parameters, representing a very wide range of $T0/T1$ ratios. In the first case, we let a = 360 ns, d = 4 bytes, and c = 60 ns (i.e., $T0$ = 360 ns, $T1$ = 15 ns/byte.) These parameters correspond to typical levels of performance possible from the (P896) IEEE Futurebus [3], [5] and also to the behavior of the Fairchild CLIPPER [7] when using a zero wait state memory. The mean delays in nanoseconds are shown in Table VII, and are plotted for a

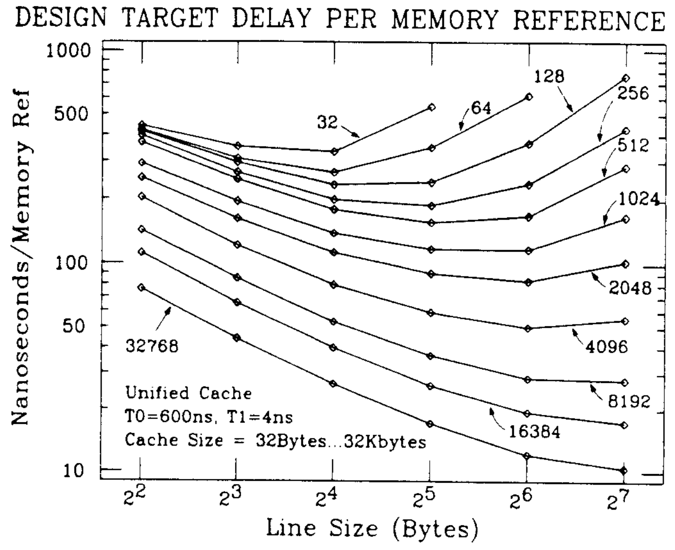

Fig. 10. Average memory delay per memory reference, based on design target miss ratios for a unified cache. Cache size from 32 bytes to 32K bytes, and line size from 4–128 bytes. Memory access latency ($T0$) is 600 ns; additional time for each byte of the line is 4 ns ($T1$).

TABLE VII
DESIGN TARGET MEAN MEMORY DELAY PER MEMORY REFERENCE
DELAY = 360 NS + 15 NS/BYTE

Cache Type: Unified	Mean Memory Delay per Memory Reference					
	Line Size:					
Size	4	8	16	32	64	128
32	301.	267	300.	630.		
64	288.	234.	240.	403.	950.	
128	283.	224.	210.	277.	566.	1563.
256	270.	202.	180.	217.	364.	881
512	250.	187.	162.	181.	261.	585.
1024	199.	148.	126.	136.	181.	345.
2048	170.	124.	102.	104.	129.	212.
4096	138.	93.	72.	69.	78.	114.
8192	98.	65.	48.	42.	44.	57.
16384	76.	50.	36.	30.	30.	36.
32768	52.	33.	24.	20.	19.	21.
Cache Type: Instructions						
32	304.	230.	198.	208.		
64	283.	210.	180.	186.	252.	
128	258	191.	162.	166.	216.	358.
256	249.	179.	150.	149.	183.	294.
512	236.	167.	138.	133.	157.	247.
1024	212.	148.	120.	113.	129.	192.
2048	164.	113.	90.	82.	90.	130.
4096	114.	77.	60.	53.	57.	73.
8192	72.	48.	36.	31.	31.	37.
16384	62	41.	30.	25.	24.	27.
32768	38.	25.	18.	15.	14.	15.
Cache Type: Data						
32	307.	293	330.	532.		
64	277.	247.	270.	416	915.	
128	236.	198.	210.	295.	616.	1543.
256	197.	162.	168.	228.	430.	1040.
512	145.	118	120.	161.	283.	643.
1024	119.	101	96.	116.	184.	366.
2048	107.	81.	72.	79.	110.	203.
4096	104.	73	60.	59.	71.	111.
8192	90.	62.	48.	45.	51.	72.
16384	68.	47.	36.	33.	34.	44.
32768	45.	31.	24.	21.	22.	27.

unified cache in Fig. 8. This design is one with moderate to high memory latency, and for comparison we have done the same computation with all parameters the same except that a = 160 ns; i.e., with a much lower ratio $T0/T1$. ($T0$ = 160 ns, $T1$ = 15 ns.) These results are shown in Table VIII and Fig. 9.

TABLE VIII
DESIGN TARGET MEAN MEMORY DELAY PER MEMORY REFERENCE
DELAY = 160 NS + 15 NS/BYTE

Cache Type: Unified	Mean Memory Delay per Memory Reference					
	Line Size:					
Size	4	8	16	32	64	128
32	158.	156.	200.	480		
64	151.	137.	160.	307.	806.	
128	148.	131.	140.	211.	480.	1426.
256	141.	118.	120.	165	309.	804.
512	131.	109	108.	138.	221.	534.
1024	104.	86.	84.	103	154.	314.
2048	89.	72.	68.	80.	109.	193.
4096	72.	54.	48.	52.	66.	104.
8192	51.	38.	32.	32.	37.	52.
16384	40.	29.	24.	23.	26	32.
32768	27.	19.	16.	15.	16.	19.
Cache Type: Instructions						
32	159.	134.	132.	158.		
64	148.	123.	120.	142	214.	
128	135.	111.	108.	126.	183.	327
256	130.	104.	100.	114.	155.	268.
512	124.	98	92.	102.	133.	225.
1024	111.	86.	80.	86	110.	175.
2048	86.	66.	60.	62	77.	118.
4096	60.	45.	40.	40	48.	67.
8192	38.	28.	24.	23.	26.	34.
16384	33.	24.	20.	19	20.	25.
32768	20	15.	12.	11.	12.	14.
Cache Type: Data						
32	161.	171.	220.	458		
64	145.	144.	180.	317.	776.	
128	123.	115.	140.	225.	523.	1407.
256	103.	94.	112.	174.	365.	949
512	76.	69.	80.	122.	240	586.
1024	62.	59.	64.	88.	156	334
2048	56.	47.	48.	60.	93	185.
4096	54.	43.	40.	45.	60.	101.
8192	47.	36.	32.	34.	43.	66.
16384	36.	27.	24.	25.	29.	40.
32768	24.	18.	16.	16.	19.	24.

It is also possible to predict that bus transmission rates will increase. The IEEE Futurebus obtains high performance by using a two-edge handshake between the sender and receiver, without requiring all units on the bus to acknowledge each data word. It is possible to obtain still higher performance by using a synchronous protocol in which the bus is treated as a transmission line, and no acknowledgment is given after each word. To study that case, we let a = 600 ns, and c = 16 ns (i.e., $T0$ = 600 ns, and $T1$ = 4 ns). This represents a very high ratio of $T0/T1$, although higher ratios of $T0/T1$ would be possible if cache misses were handled by software [6]. The results appear in Table IX and Fig. 10.

It can be seen from Tables VII–IX that the minimum memory delay is obtained for a line size that varies with the cache size. For example, for a 256 byte instruction cache, such as is used on the Motorola 68020 [19] the mean delay is minimized in the first case by a 16–32 byte line, in the second case by a 16 byte line, and in the third case by a 64 byte line. (The timing characteristics for the 68020 memory are, however, not likely to be exactly the same as proposed here.) For a single chip instruction cache of 4K bytes, the delay is minimized in the first case by a line size of 32 bytes (range of 16–64 bytes), in the second by a line size of 16–32 bytes (range 8–64 bytes), and in the third case by a line size of 128 bytes or larger (range > =32 bytes). For a 4K byte data cache, the figures are, respectively, 32 (8–64) bytes, 16 (8–32) bytes, and 64 bytes (16–128).

TABLE IX
DESIGN TARGET MEAN MEMORY DELAY PER MEMORY REFERENCE
DELAY = 600 NS + 4 NS/BYTE

Cache Type:	Mean Memory Delay per Memory Reference					
Unified	Line Size:					
Size	4	8	16	32	64	128
32	442.	351.	332	546.		
64	422.	308	266.	349.	616.	
128	415.	295.	232	240.	367.	762.
256	396.	265.	199.	188.	236.	430.
512	367.	246.	179.	157.	169.	285
1024	292.	195.	139	118.	118	168.
2048	249.	163	113.	90.	84	103
4096	203.	122	80.	59.	50	56.
8192	143.	85.	53	37.	29.	28.
16384	112.	65.	40.	26.	20.	17
32768	76.	44	27.	17.	12	10
Cache Type: Instructions						
32	446.	302.	219.	180		
64	415.	277.	199.	162.	163.	
128	379.	251.	179.	143	140.	175.
256	365.	236.	166.	129.	119.	143.
512	346.	220.	153.	116.	102.	120.
1024	311.	194.	133	98.	84.	94.
2048	241.	148.	100.	71.	58.	63.
4096	167.	102	66.	46.	37	36.
8192	106.	63.	40.	27.	20.	18.
16384	91.	54.	33.	21	16	13
32768	56.	33.	20.	13	9.	7.
Cache Type: Data						
32	451.	386.	365.	521.		
64	406.	326.	299	360.	593.	
128	346.	260.	232	256.	399.	752.
256	289.	213	186.	198.	279.	507.
512	213.	155.	133.	139.	184.	313.
1024	174.	133.	106.	100.	120.	179.
2048	157.	107.	80	69.	71.	99.
4096	152.	97.	66.	51.	46.	54.
8192	132.	82.	53.	39.	33.	35.
16384	99.	61.	40.	28.	22	22.
32768	66.	41.	27.	18.	14.	13.

It is worth noting that within reasonable ranges of line sizes, for a given cache size, performance is comparable; the selection of a line size in the range of 16–64 bytes is likely to be satisfactory, if not optimal, for a wide variety of systems.

D. Bus Busy Time

In the case of a multiprocessor system with a shared common bus to memory, the extent to which each processor keeps the bus busy (in use) is almost as important as (or more important than) the memory reference delays suffered by each processor. In such a system, the limiting resource is usually bus bandwidth, and the aggregate system performance is bounded by the amount of traffic the bus can support. A tradeoff which permits more processors on the bus may permit greater overall system throughput, even though each processor may function more slowly. Thus, for example, a cache that is slow but with a high hit ratio may be better for overall multiprocessor system performance than a faster cache with a lower hit ratio, even though the latter might yield, in a uniprocessor system, a lower mean memory reference time and greater throughput.

In most existing and proposed bus systems, however, the bus is held from the beginning of the address cycle to the last data cycle, so the bus busy time is equal to the delay experienced at the processor for a cache miss. (Actually, the processor may experience a few cycles delay beyond that for bus busy, due to additional cycles to detect the miss, reinitiate the fetch, etc.) Our point here is that the bus busy is not necessarily proportional to the number of bytes transmitted (bus traffic); a simple-minded minimization of the bus traffic is not the correct optimization procedure.

E. Comparisons

It is worth comparing our results to those in [17]. There, the optimal line size for the Amdahl 470V/6 is determined to be 128 bytes. That machine's cache is 16K bytes, a = 520 ns, d = 8 bytes, and c = 32.5 ns (4.0625 ns/byte). We compute a mean delay per memory reference, for lines of 8, 16, 32, 64, and 128 bytes, of respectively, 56.9, 35.1, 23.4, 17.9, and 16.6 ns. Our results, therefore, agree with Kumar's in this case. (Although our data do not go above 128 byte lines, the curve is clearly becoming flat.)

For the IBM 370/168, the parameters are a = 160 ns, d = 8 bytes, and c = 80 ns [22]. Our computed delays are 24.7, 19.2, 17.3, 18.4, and 23.0 ns. The optimal line size is thus 32 bytes, which happens to match the actual line size.

We noted earlier that Goodman [12] had some results on line size. In that paper, enough data are given that the ratio of ratios values for cache sizes of 4K and 16K may be calculated. For a 4K byte cache, the ratio of ratios are (warm start simulation) 0.663, 0.701, 0.824, 1.048, 1.091, and (cold start) 0.704, 0.762, 0.928, 1.089, and 1.254. For the 16K byte cache, the numbers are 0.648, 0.620, 0.607, 0.703, and 0.864, and 0.604, 0.677, 0.585, 0.664, and 0.754. These figures are considerably less favorable to large line sizes than ours. The differences could be due to a genuinely different workload or a small workload sample (6 VAX 11/780 traces). The figures in [13] seem similar to ours, but exact numbers (from tables) are not available.

V. Evaluation, Prediction, and Validation

It is interesting to apply our design target miss ratios from Table V to the cache designs for some high performance microprocessors. As noted earlier, the Motorola 68020 [19] has a 256 byte instruction cache, which is organized as 4 byte lines. We predict that that design will have a 59 percent instruction fetch miss ratio. Performance would likely have been better with a larger line size.

The Zilog Z80,000 [1], [21] has a 256 byte on-chip cache which can be used for instructions, for data, or as a unified cache. It has a sector organization with 16 byte sectors (large blocks); those large blocks can be loaded in their entirety or subsectors only may be loaded. Considering only the use of full sectors, we predict miss ratios of 30, 25, and 28 percent if the cache is used as a unified cache, instruction cache or data cache, respectively. This compares to a projected miss ratio of 12 percent from [1]. The reasons for the difference between our predictions and Alpert's are given in [27].

For the Fairchild CLIPPER [7], which has a 4K byte instruction cache, a 4K byte data cache, 2-way set associative mapping, and a 16 byte line size, we predict miss ratios as follows. The caches are 2-way set associative, and other experiments (see [7]) suggest that the miss ratio for 2-way set associativity is about 20 percent higher than that for a fully associative design. The data cache should, therefore, have a miss ratio of about 12 percent. The instruction cache uses prefetch, and combining results from [27] on the effect of prefetch, with the effect of 2-way set associative mapping, we predict a miss ratio of about 3.6 percent.

For validation, we note the results in [8]. In that paper, hardware monitor measurements of the VAX 11/780 are presented; the 11/780 has an 8K byte unified cache with 8 byte lines. For one workload (normal daytime use), a miss ratio of 10.3 percent is observed; for the other workload (remote terminal emulator, heavy timesharing load), the miss ratio is 15.8 percent. These two figures bracket our estimate of 13.5 percent (Table V), which lends some credibility to our DTMR's.

VI. Conclusions

There have been four main purposes to this paper: to quantify the variation in the miss ratio with changes in the line size; to provide to the system designer a set of design target miss ratios around which she or he can design a new implementation of a possibly new architecture; to provide customers a basis for evaluating vendor claims; and to provide standardization groups (such as that for the IEEE P896 Futurebus) a basis by which to select necessarily fixed parameters. The first goal has been realized in Table III, where the change in the miss ratio with line size is shown; a continuous curve mathematical fit to those data is also provided. The ratio of ratios data have been applied to previously published design target miss ratios for caches with 16 byte lines to get design target miss ratios over a range of line sizes; those appear in Table V. We have used that information to predict bus traffic and average memory delays per memory reference. The range of possible bus timing parameters has provided a basis for standardization efforts. Also discussed are the other tradeoffs in selecting a line size.

There are three important ways in which this work could be usefully extended. First, a wider variety of workloads and traces should be examined to verify that the results are consistent with those given here, and the sensitivity of our results to task switch frequency should be determined. Second, the results presented here are purely experimental; it would be valuable to have some model of program behavior from which these results could be derived. Third, there needs to be an additional effort to validate our results against real machines running real workloads, as measured by hardware monitors.

Acknowledgment

The traces used for this research have come from many sources. My thanks to Amdahl Corporation and B. Harding for the IBM 370 traces, to Zilog Corporation, J. Banning and J. Porcar for the Z8000 traces, to Signetics Corporation and M. Freeman for the M68000 traces, to J. Lee for the CDC 6400 traces, and to R. Henry and G. Taylor for the VAX 11/780 traces. Thanks also to H. Sachs and others who asked the kind of questions that prompted this research.

References

[1] D. Alpert, D. Carberry, M. Yarmamura, Y. Chow, and P. Mak, "32-bit processor chip integrates major system functions," *Electronics*, pp. 113-119, July 14, 1983.

[2] D. Alpert, "Performance tradeoffs for microprocessor cache memories," Comput. Syst. Lab. Tech. Note 83-239, Stanford Univ., Stanford, CA, Dec. 1983.

[3] R. V. Balakrishnan, "The proposed IEEE 896 futurebus—A solution to the bus driving problem," *IEEE Micro*, pp. 23-27, Aug. 1984.

[4] F. Baskett and A. J. Smith, "Interference in multiprocessor computer systems with interleaved memory," *Commun. ACM*, vol. 19, pp. 327-334, June 1976.

[5] P. Borrill and J. Theus, "An advanced communication protocol for the proposed IEEE 896 futurebus," *IEEE Micro*, pp. 42-56, Aug. 1984.

[6] D. R. Cheriton, G. A. Slavenburg, and P. Boyle, "Software controlled caches in the nebula multiprocessor," Comput. Sci. Dep. Tech. Rep., Stanford Univ., 1986.

[7] J. Cho, A. J. Smith, and H. Sachs, "The memory management architecture and the cache and memory management unit for the Fairchild CLIPPER," UC Berkeley CS Tech. Rep. UCB/CSD 86/289, Mar. 1986.

[8] D. Clark, "Cache performance in the VAX 11/780," *ACM Trans. Comput. Sys.*, vol. 1, pp. 24-37, Feb. 1983.

[9] G. C. Driscoll, T. R. Puzak, H. E. Sachar, and R. D. Villani, "Staging length table—A means of minimizing cache memory misses using variable length lines," *IBM Tech. Disclos. Bull.*, vol. 25, p. 1285, Aug. 1982.

[10] G. C. Driscoll, J. J. Losq, T. R. Puzak, G. S. Rao, H. E. Sachar, and R. D. Villani, "Cache miss directory—A means of prefetching 'missed' lines," *IBM Tech. Disclos. Bull.*, vol. 25, p. 1286, Aug. 1982.

[11] D. H. Gibson, "Considerations in block-oriented systems design," in *Proc. SJCC*, 1967, pp. 75-80.

[12] J. R. Goodman, "Cache memory optimization to reduce processor memory traffic," CS Rep., Univ. Wisconsin, Jan. 1985.

[13] I. J. Haikala and P. H. Kutvonen, "Split cache organizations," Rep. C-1984-40, Dep. Comput. Sci., Univ. Helsinki, Finland, Aug. 1984, in *Proc. Performance '84*. New York: Elsevier-North Holland, 1984, pp. 459-472.

[14] M. Hill and A. J. Smith, "Experimental evaluation of on-chip

microprocessor cache memories," in *Proc. 11th Annu. Symp. Comput. Architecture*, Ann Arbor, MI, June 1984, pp. 158-166.

[15] K. R. Kaplan and R. O. Winder, "Cache-based computer systems," *IEEE Computer*, pp. 30-36, Mar. 1973.

[16] R. H. Katz, S. J. Eggers, G. A. Gibson, P. M. Hanson, M. D. Hill, J. M. Pendleton, S. A. Ritchie, G. S. Taylor, D. A. Wood, and D. A. Patterson, "Memory hierarchy aspects of a multiprocessor RISC: Cache and bus analysis," Rep. UCB/CSD 85/221, Univ. California, Berkeley, Jan. 1985.

[17] B. Kumar, "A model of spatial locality and its application to cache design," Tech. Rep., Comput. Syst. Lab., Stanford Univ., Stanford, CA, 1979 (unpublished).

[18] M. H. MacDougall, "Instruction level program and processor modeling," *IEEE Computer*, pp. 14-24, July 1984.

[19] D. MacGregor, D. Mothersole, and B. Moyer, "The Motorola MC 68020," *IEEE Micro*, pp. 101-118, Aug. 1984.

[20] R. M. Meade, "On memory systems design," in *Proc. FJCC*, 1970, pp. 33-43.

[21] A. Patel, "An inside look at the Z80,000 CPU: Zilog's new 32-bit microprocessor," in *Proc. NCC*, 1984, pp. 83-91.

[22] B. L. Peuto and L. J. Shustek, "An instruction timing model of CPU performance," in *Proc. 4th Annu. Symp. Comput. Architecture*, pp. 165-178, Mar. 1977.

[23] A. L. Scherr, "Analysis of storage performance and dynamic relocation techniques," *IBM SDD*, Poughkeepsie Tech. Rep. TR00.1494, Sept. 1966.

[24] A. J. Smith, "Sequentiality and prefetching in data base systems," *ACM Trans. Data Base Syst.*, vol. 3, pp. 223-247, Sept. 1978.

[25] ——, "Cache memories," *Comput. Surv.*, vol. 14, pp. 473-530, Sept. 1982.

[26] ——, "CPU cache memories," in *Handbook for Computer Designers*, Flynn and Rossman, Eds., to be published.

[27] ——, "Cache evaluation and the impact of workload choice," in *Proc. 12th Int. Symp. Comput. Architecture*, Boston, MA, June 17-19, 1985, pp. 64-75.

[28] W. D. Strecker, "Cache memories for the PDP-11 family computers," in *Proc. 3rd Annu. Symp. Comput. Architecture*, Jan. 1976, pp. 155-158.

[29] K. G. Tan, "Cache organization with various line sizes," *IBM Tech. Disclos. Bull.*, vol. 25, pp. 1282-1284, Aug. 1982.

Alan Jay Smith (S'73-M'74-SM'83) was born in New Rochelle, NY. He received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA, the latter in 1974.

He is currently a[redacted] Professor in the Computer Science Division of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he has been on the faculty since 1974, and was Vice Chairman of the EECS Department from July 1982 to June 1984. His research interests include the analysis and modeling of computer systems and devices, operating systems, computer architecture, and user interfaces. He has published a large number of research papers.

Dr. Smith is a member of the Association for Computing Machinery, the Society for Industrial and Applied Mathematics, the Computer Measurement Group, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi. He is Chairman of the ACM Special Interest Group on Operating Systems (SIGOPS), is on the Board of Directors of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS), was an ACM National Lecturer (1985-1986), is an IEEE Distinguished Visitor, and is an Associate Editor of the *ACM Transactions on Computer Systems* (TOCS). He won the IEEE Best Paper Award for the best paper in the IEEE TRANSACTIONS ON COMPUTERS in 1979.