Experimental Evaluation of On-Chip Microprocessor Cache Memories

Mark D. Hill
Alan Jay Smith

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, California 94720

Abstract

Advances in integrated circuit density are permitting the implementation on a single chip of functions and performance enhancements beyond those of a basic processors. One performance enhancement of proven value is a cache memory; placing a cache on the processor chip can reduce both mean memory access time and bus traffic. In this paper we use trace driven simulation to study design tradeoffs for small (on-chip) caches. Miss ratio and traffic ratio (bus traffic) are the metrics for cache performance. Particular attention is paid to sub-block caches (also known as sector caches), in which address tags are associated with blocks, each of which contains multiple sub-blocks; sub-blocks are the transfer unit. Using traces from two 16-bit architectures (Z8000, PDP-11) and two 32-bit architectures (VAX-11, System/370), we find that general purpose caches of 64 bytes (net size) are marginally useful in some cases, while 1024-byte caches perform fairly well; typical miss and traffic ratios for a 1024 byte (net size) cache, 4-way set associative with 8 byte blocks are: PDP-11: .039, .156, Z8000: .015, .060, VAX 11: .080, .160, Sys/370: .244, .489. (These figures are based on traces of user programs and the performance obtained in practice is likely to be less good.) The use of sub-blocks allows tradeoffs between miss ratio and traffic ratio for a given cache size. Load forward is quite useful. Extensive simulation results are presented.

1. Introduction

Advances in integrated circuit density are permitting the single chip implementation of features, functions and performance enhancements beyond those of basic eight and sixteen bit processors. Processors now being designed include not only full 32-bit architecture instruction sets, but also have sufficient area for performance enhancements such as buffering, pipelining, and cache memories. However, the chip area will not be sufficient for a number of years to include all of these performance enhancing features in their full generality. In this paper we study the design and use of on-chip cache memories, which we believe to be one of the best uses for additional chip area.

Caches are a time-tested mechanism for improving memory system performance by reducing access time and memory traffic through the exploitation of temporal and spatial locality of reference. Temporal locality specifies that a location recently referenced is likely to be referenced again in the near future, or conversely, that information used in the near future is likely to consist primarily of information used recently. Spatial locality is the identical concept for locations near the recent reference. A complete survey of cache memories can be found in Smith [1].

On-chip caches will generally differ from traditional caches. Initially these caches will be small (32 to 2048 bytes) because the limited chip area must be allocated among the instruction set implementation, the cache, and other possible performance enhancements. As fabrication techniques provide room, some of the additional area will be used to increase cache size. Blocks in on-chip caches will tend to be smaller than traditional cache blocks because packaging limitations prevent large parallel loads. In microprocessor systems, relative to mini and mainframe computers, bus traffic can seriously limit system performance. This problem is particularly acute if the bus is to be shared among two or more microprocessors. Concern over bus traffic should favor schemes that exploit temporal locality more than spatial locality. On-chip caches will tend to be more intelligent than traditional caches because special-purpose cache control logic is relatively inexpensive in VLSI and can easily access processor state information.

This paper examines the performance of small caches, using trace driven simulations. In particular, we examine a cache organization called *sub-block placement* (*Sub-block placement* has also been called *sector placement* in the IBM System/360 Model 85 [2].) In sub-block placement, the address tags in the cache are associated with a larger amount of data than is usually brought into the cache on a single miss. This larger amount is known as a *block*. Smaller units, called *sub-blocks,* are used as the unit of memory transfer. Basically a block is composed of an address tag and two or more sub-blocks. When a reference is made to a location not in the cache, an entire block is allocated but only the missing sub-block is loaded. Address tags are associated with the larger sized blocks in order to minimize the chip area devoted to address tags, and in the case of the 360/85, to minimize the extent of the associative search. Sub-blocks are used as the unit of memory transfers to minimize bus traffic. The expectation in sub-block placement is that the savings in tag overhead will compensate for the rigidity of the mapping.

This paper is organized into six sections. This section concludes with a review of the literature. Section 2 examines the design of on-chip caches, outlines the effects of VLSI on cache design, and proposes some examples. Section 3 explains cache design parameters, performance and cost metrics, and the choice of trace-driven simulation. Section 4 presents the results of our simulations. We find that *sub-block* placement is a flexible cache structure for trading off memory access latency and bus traffic. The use of "nibble mode" memories suggests the use of larger block sizes than would otherwise be optimal. "Load forward" is found to be helpful in significantly cutting bus traffic at a cost of a small increase in the miss ratio. Conclusions are presented in Section 5.

1.1. Review of Literature

The first cache memory implementation was for the IBM System/360 Model 85. Results were published by Liptay [2] in 1968. This cache is the only early cache to use a sub-block placement structure. (Liptay called this design the *sector cache*.) Results in Section 4.1 show that it performs poorly by today's standards.

Few cache studies in the literature have been done for the small cache sizes (less than 2048 bytes of data) that we propose for initial on-chip caches. In 1974, Bell, Casasent, and Bell [3] published results on small caches with single-word (16-bit) blocks and direct-mapped placement for the DEC PDP-8, using traces of two scientific programs and the PDP-8 Assembler. The miss ratios that they reported are generally higher than we have found, perhaps due to their use of direct mapping. For example, a cache of 512 data bytes was found to have a miss ratio between 0.46 and 0.62. We found a

0194-7111/84/0000/0158$01.00©1984 IEEE

miss ratio of 0.10 for a comparable PDP-11 cache.

In 1976, Strecker [4] summarized the research that led to the design of the cache memory in the DEC PDP-11/70. He used traces of scientific applications and an assembler. For direct-mapped caches with a block size of 4 bytes (2 PDP-11 words), the miss ratio dropped from 0.15 through 0.10, 0.05, and 0.02 as the cache data size was doubled from 256 bytes up to 2048 bytes; these results are similar to our own. Strecker also notes that improved performance is obtained as the associativity in a set-associative cache increases from 1 to 2 to 4, but that little is gained for degrees of associativity of greater than 4. He also presented results showing that there is little difference in the performance of LRU, FIFO, and RANDOM replacement algorithms. The PDP-11/70 cache was designed to be 1024 data bytes, with 4-byte blocks, and two-way set associative with random replacement.

Recent work by Goodman [5] examines small caches within a multiprocessor environment. Among other issues, he examines the sub-block structure (sector cache) used in the System/360-85.[1] His results, based on six UNIX[2] traces, support sub-block placement.

2. On-Chip Caches

This section is concerned with design issues for on-chip caches. The first sub-section discusses the effects of VLSI on on-chip caches, including the disparity between on-chip and off-chip communication and the ability to inexpensively build specialized cache control. Next, two examples are proposed: the *minimum* cache and the *smart* cache. The minimum cache costs very little and can reduce off-chip accesses by one-third; the smart cache out-performs current caches of the same size. Finally, the RISC II instruction cache is presented as an implemented example.

2.1. Effects of VLSI

On-chip caches will differ from MSI (Medium Scale Integration) cache memories for at least four reasons. First, block sizes will tend to be smaller, because integrated circuit packaging limitations limit the number of bits that can be handled in parallel. If large blocks are used, long latency times result because the blocks must be time-multiplexed over only a few pins. Second, since most successful microprocessors today are bus-limited [6], on-chip caches will be increasingly concerned with minimizing bus traffic. This trend will reduce the main memory-to-cache transfer size. Third, intra-chip communication is less expensive than inter-chip communication. Therefore more processor state information can be used to control the on-chip cache. Data communication off-chip requires that signals be scaled to reasonable levels, such as TTL compatible (5 volts), and be able to drive large capacitive loads. This costs chip area, power, and delay. The cost of the intra-chip transfer of information, e.g. the current instruction-in-execution, may be low enough to justify their incremental improvements to cache performance. Last, the current freedom from standard building blocks in VLSI can allow on-chip caches to use more complex custom control. For example, wide associative searches can be easily implemented in VLSI with content-addressable memory (CAM) cells.

2.2. Two Proposals for On-Chip Cache Architectures

The two architectures for on-chip caches proposed here were not explicitly studied but serve to illustrate the potential for VLSI on-chip caches. The first, called a *minimum* cache, is a cross between an instruction buffer and a cache. The second, a *smart* cache, is similar to a traditional cache but uses custom VLSI to capitalize on the processor state information that is available on-chip.

An instruction buffer holds one or more blocks of the instruction address space[7], feeds into the instruction fetch stage of the CPU pipeline, and may or may not be capable of recognizing when a branch target hits a location already in the buffer. Instruction buffers which do not recognize branch targets differ from caches because while they reduce latency for consecutive instruction accesses, they do not reduce the number of bytes required from the memory system. For example, the DEC VAX-11/780 and 750 have instruction buffers that contain only eight contiguous bytes [8]. The CRAY-1 has four instruction buffers that can each hold 64 contiguous 16-bit instructions for a total size of 512 bytes [9]; the CRAY-1's instruction buffers recognize branch targets, and thus can hold entire loops.

Minimum caches can improve performance, even with relatively high miss ratios, since they may significantly cut the traffic ratio. One possible architecture for a minimum cache is 32 data words broken into 16 2-word blocks, where only the requested word is loaded on a miss. For this architecture, it is reasonable to use 2-way set-associative placement with *RANDOM* replacement. A minimum cache for a 32-bit machine would require about 190 bytes of RAM (16 blocks * [29 tag bits + 2 valid bits + 64 data bits] / 8 bits/byte), plus some overhead for buffers and matching logic. Minimum caches tend to have their best effect on performance when used as instruction buffers.

Smart caches require more area than the minimum cache, perhaps up to 2048 bytes of data storage. Their performance is enhanced by exploiting the processor state information available on-chip. Special-purpose logic can examine reference patterns to prefetch instruction codes and operands from main memory or prefetch information out of the cache memory into even faster buffers. For example, the S-1 [10] attempts to dynamically predict jump targets, and the RISC II instruction cache prefetches instruction data from its private store (see Section 2.3). Lee and Smith [7] present empirical evidence supporting the use of a cache for recent jump targets to reduce the detrimental effects of branches on pipelined machines. Alternatively, the tops of certain stacks in a programming environment could be cached. These "intelligent" schemes may work better than methods which are ignorant of the semantics of data reference patterns. Effective prefetching reduces latency at a cost of increased memory traffic and at a risk of memory pollution (fetching data which is not subsequently used, while replacing data that may yet be used) [11]. Intelligent data replacement reduces memory pollution, thereby reducing bus traffic and latency.

2.3. An Implemented Example: RISC II Instruction Cache

The RISC II instruction cache [12] is a single-chip instruction cache implementation used to study architectural concepts for on-chip cache memories. The cache includes two architecture innovations: a remote program counter for decreased access time, and support for dynamic code expansion for increased effective cache size.

The RISC II instruction cache is a single 45,000 transistor NMOS chip designed specifically to work with the RISC II implementation [13] of the RISC architecture [14]. The cache operates with an access time of 250 ns. It holds 512 bytes arranged in 64 direct-mapped blocks of 8 bytes each. Results quoted in this subsection are from simulations with limited benchmarks done for [12]. Various cache sizes performed with miss ratios of 0.148 (512 bytes), 0.125 (1024 bytes), 0.098 (2048 bytes), and 0.078 (4096 bytes). Thus for these sizes, doubling the cache size reduced miss ratio by about 20 percent.

The *remote program counter* is special-purpose logic that reduces the effective access time on cache hits. The remote program counter attempts to guess the next instruction address, so the cache can begin to fetch data out of its private store before the processor presents the actual instruction fetch address. The remote program counter correctly predicted 89.9 percent of the next-instruction addresses using limited instruction-decode ability and static *jump-likely* hints. This reduced the access time seen from the processor by 42.2 percent.

[1] Goodman refers to the *block* as a *address block* and to the *sub-block* as a *transfer block*.

[2] UNIX is a trademark of Bell Laboratories.

The effective size of an instruction cache can be increased with *code-compaction*. The RISC II instruction cache allows selected half-word (16-bit) instructions to be used to reduce code size by 20 percent. These half-word instructions are expanded to their standard 32-bit representation before they are given to the processor. Miss ratios are improved by 27.0 percent over runs with standard code without impacting the processor's simple instruction-decode PLA.

3. Methods

This section presents the rationale for the choice of cache design parameters in this study, discusses performance metrics, and justifies the use of trace driven simulation as the evaluation tool. Miss and traffic ratios are used to measure performance for caches of different net size (data size), block size, and sub-block size. Gross cache size (the size of address tags and data) is used to measure cost.

3.1. Cache Design Parameters

Table 1 presents cache design parameter choices used in our simulation studies. *Cache size, sub-block size,* and *block size* are all important to cache performance. *Cache size* directly affects performance and cost in a way that dominates all other design decisions. The miss ratio of a cache declines monotonically with cache size, but the cost, physical size, and access time will increase with increasing cache size. Optimal cache sizes thus must be selected with regard for these factors.

Sub-block size is the number of bytes moved to the cache by a minimum size data transfer. *Block size*, also called *line size*, is the maximum number of bytes of data that are associated with a single address tag. In most caches, the sub-block size and the block size are identical and are collectively referred to as the *block size*. Dividing large blocks into smaller sub-blocks affects performance in two ways. First, the miss ratio will increase because more than one miss can occur within a single block. Second, the traffic ratio will decrease because only the parts of the block that are used will be loaded. In VLSI we expect the smaller sub-block size to reduce the cost of a miss, because less data is brought across limited pins. Sub-block placement can also be viewed as a way of grouping n small blocks under one address tag. This grouping reduces data placement freedom by restricting the n sub-blocks to consecutive locations. This loss in flexibility of placement will cause the miss and traffic ratios to increase. However, the cache will be more cost-effective since address tag overhead is reduced by a factor of n.[3]

Cache design parameters not varied in the studies of this paper are associativity, replacement algorithm, fetch algorithm, and write-back algorithm. All experiments were performed with 4-way set-associative caches. Smith [15] and others have shown that 4-way set-associative mapping provides hit ratios very close to those of a fully associative design. Changes in associativity have been extensively studied and have smaller effects than the parameters studied here [1, 4]. Experiments were done using *LRU* replacement since LRU permits more efficient simulation [16] and reasonable alternatives perform comparably. All cache fetches were done on demand, although sub-block placement with load-forward is analogous to prefetching. Prefetching studies were beyond the scope of this study. Write-back issues were filtered out of our results by calculating performance metrics for only data reads and instruction fetches. The caches studied hold both instructions and data. Further studies should look at partitioning instruction and data caches, prefetching, and write through vs. copy back factors.

3.2. Cache Performance Metrics

The purpose of a cache is to improve memory system performance by reducing the effective access time to memory and processor-memory bandwidth requirements. Effective access time t_{eff} is most simply modeled as:

$$t_{eff} = t_{cache} * (1 - m) + t_{mem} * m,$$

where m is the miss ratio of the cache, t_{cache} is the access time of the cache, and t_{mem} is the access time of main memory. t_{cache} and t_{mem} are not easy to obtain through any method that does not intimately consider a particular implementation of a machine. t_{cache} is a function of the technology, organization, and complexity of a cache. t_{mem} is related to memory technology, organization, complexity, bus interference, cache sub-block size, and whether loading and fetching are overlapped. Miss ratios are often quoted for caches because they are largely implementation-independent (if the processor-to-cache datapath width is held constant) and thus can be computed in architectural studies. Nevertheless, a reduction in miss ratio will not guarantee an improvement in memory system performance. For example, prefetching data should reduce the misses when the data is actually requested. Still t_{eff} may grow if the additional logic increases t_{cache}. Other aspects of cache design which affect performance include the methods of updating main memory and maintaining multi-cache consistency.

The two most important architectural performance metrics are *miss ratio* and *traffic ratio*. The miss ratio is the number of cache misses divided by the number of cache accesses. The miss ratio is always less than or equal to 1 because the number of misses cannot be more than the number of cache accesses. It is of primary importance if sufficient bus bandwidth is available so that reduced latency is the overriding goal, as is generally the case for high end mainframes. The relative importance of miss ratio and traffic ratio vary with the ratio of cache and main memory access times; the smaller the ratio, the less important are reductions in the miss ratio.

The traffic ratio is the ratio of memory bus traffic in a system with a cache to that in a system without a cache. The scaled traffic ratio (see section 4.3) scales the traffic ratio to reflect the fact that in some designs, as with nibble or page mode memories, transfer times are not linear with transfer sizes. Two competing factors affect the traffic ratio. First, repeated references to words already in the cache reduce the traffic ratio. For example, if words are brought into the cache and used n times before they are removed, the traffic ratio will be $\frac{1}{n}$. Second, block sizes that are bigger than a single word will tend to increase the traffic ratio because some data brought into the cache will never be referenced. For example, if only one reference is made to a block of w words while it is in the cache, the traffic ratio will be $\frac{w}{1}$. Consequently, the traffic ratio for a cache with w-word blocks will be less than 1 if and only if on the average more than w references are made to a block each time the block is resident in the cache. The traffic ratio is important if the memory bus is the bottleneck, either because the single processor is too fast for the bus, or because there are multiple processors on the same bus. It is also important because of the contention between the processor, which wants to use the cache, and the bus which is loading and unloading it.

The results of Section 4.3 model the bus traffic to fetch w sequential words in a single transaction by a cost of the form $a + b*w$. This form reflects transfer times in systems with nibble-mode memories [17] or shared busses with transactional overhead.

The most important cost metrics are *gross cache size* and *cache complexity*. By gross cache size we mean the size of the data and tag area together. Historically, cache performance numbers have been given with respect to cache data size (net cache size) only. However, on-chip caches can have non-trivial tag areas because block sizes are smaller, associativity may be greater, and address spaces for the newer architectures are large (32-bit). The gross cache sizes calculated in this paper assume a 32-bit address space even though some of the traces come from 16-bit machines, since we are interested in the newer 32-bit architectures. The effects of cache complexity are important, but are not studied here because they are

[3] We neglect the lower-order effects of changes in the number of bits in the address tag.

implementation-dependent and are poorly measured by architecture-level, trace-driven simulation.

3.3. Trace-Driven Simulation

Trace-driven simulation experiments were used in these studies for several reasons. First, such simulations are repeatable and allow cache design parameters to be varied so that effects can be isolated. They are cheaper than hardware monitoring and do not require access to or the existence of the machine being studied. Simulation results can be obtained in many situations where analytic model solutions are intractable without questionable simplifying assumptions. Further, there does not currently exist any generally accepted model for program behavior, let alone one that is suitable for cache evaluation; workloads in trace-driven simulation are represented by samples of real workloads and contain complex embedded correlations that synthetic workloads often lack. Lastly, a trace-driven simulation is guaranteed to be representative of *at least one* program in execution.

A trace-driven cache simulator [18] was written which can vary, among other parameters, cache size and cache block size and associativity. It supports sub-blocks and load forward.

Tables 2, 3, 4, and 5 present the traces used to produce the results of this paper. They are normal production programs; no synthetic benchmarks have been used. Traces were run for 1 million addresses without context switches. Traces were created for the Z8000 and PDP-11 by assuming 2 byte data paths and for the System/370 and VAX-11 assuming 4 byte data paths to memory. Multiple-trace miss and traffic ratios are the unweighted average of the miss and traffic ratios of individual runs. We note that it is likely that our results will indicate better performance than will actually be achieved in practice. First, the omission of task switching effects will bias our estimated performance upward, although the small sizes of the caches studied make this effect minor. More significant is the fact that it is known [1] that most misses occur in the operating system, and we have not used, or had available, any operating system traces. Finally, it has often been observed that even when the two factors noted have been taken into account, measurements made on production systems show higher miss ratios than previously predicted. Anyone designing a system using these results should keep these comments in mind, and should appropriately interpret either our results or their own.

4. Results

This section presents the results of trace-driven simulations of small memory caches with various block and sub-block sizes. First, the original use of sub-block placement in the IBM System/360 Model 85 (the first machine with a cache memory) is examined. Second, the results from PDP-11, Z8000, and VAX-11 traces show caches as small as 64 words are somewhat useful, while 1024-byte on-chip caches can have miss and traffic ratios lower than 0.10 and 0.20. System/370 traces do not perform as well. Third, an evaluation based on the use of paged-mode memories, nibble-mode memories, and a transactional bus are shown to double optimum sub-block size. Last, a way of prefetching sub-blocks called *load-forward* is introduced.

4.1. Early Sub-Block Placement

The only early cache to use a sub-block placement structure is the IBM System/360 Model 85 Cache [2]. This 16-Kbyte cache associates an address tag with a 1024-byte block, but does memory transfers with 64-byte sub-blocks.[5] beginning with the address the processor requested. This organization was used because it required only 16 entries in expensive associative address-matching hardware.

[5] Liptay refers to the *block* as a *sector* and to the *sub-block* as a *block*.

[6] To our knowledge, these are the only public studies comparing the the 360/85's mapping scheme to today's set-associative mapping.

Technological advancements over the last 15 years have reduced the cost of associative search logic, and our results show that the sub-block or sector cache performs poorly relative to the set associative design most commonly used today. [6] Table 6 compares the 360/85 to 4-, 8-, and 16-way set associative mappings of 64-byte blocks with *LRU* replacement. This figure is three times greater than the miss ratio for a 4-way set associative cache. The reason for the 360/85 cache's poor performance is that data can be resident in the cache in only one of 16 blocks that are much too large (1024 bytes). We find that 72 percent of the sub-blocks in a block (11.52 of 16 sub-blocks) are never referenced in the period a block is resident.

4.2. Sub-Block Placement for On-Chip Caches

Before presenting PDP-11, Z8000, VAX-11, and System/370 results, let us explain how to read the figures in this paper. Figure 1, for example, shows the miss ratio versus the traffic ratio from PDP-11 runs for various block, sub-block, and cache sizes. Solid lines connect caches with constant block size bx, and dashed lines connect caches with constant sub-block size sx. For example, the cache with block size 16 and sub-block size 4 is found at the intersection of $b16$ and $s4$. The *miss ratio*, the number of cache misses divided by the number of cache accesses, is a measure of the effectiveness of a cache in reducing memory access latency. The *traffic ratio*, the ratio of bus traffic in a system with a cache divided by the bus traffic without a cache, is a measure of the effectiveness of a cache in reducing memory bandwidth requirements. The *gross cache size*, the combined size of the tag and data area of the cache, is a measure of the cost of a cache. Gross cache size is used because the different schemes presented here require widely varying amounts of address tag area.

4.2.1. PDP-11 Results

Figures 1 and 2 show the miss ratio versus the traffic ratio for PDP-11 traces; see Table 2 for description of traces. Table 7 present the gross cache sizes for the results of Figure 2.

Figure 1 shows that some organizations of caches with net sizes of 32, 128, and 512 bytes can actually increase bus traffic; i.e. their traffic ratio is greater than 1.0. For example, the 128-byte cache (net) with block and sub-block sizes of 16 bytes almost doubles bus traffic. This happens when too many words in a sub-block are brought into the cache and never referenced. Caches with a sub-block size of 1 word will always have traffic ratios less than or equal to 1.0.

Doubling the block size affects the gross cache size by halving the total address tag area. This is significant if blocks are small. For example, the 512-byte cache with block and sub-block size of 2-bytes (2,2) occupies 50.0 percent more area (1536 bytes vs. 1024 bytes) than the 512-byte cache with 4-byte blocks and 2-byte sub-blocks (4,2). Yet, it only performs 16.7 percent better in miss ratio and 16.9 percent better in traffic ratio.

Doubling the sub-block size halves the number of sub-block-valid bits in a cache block. This does not significantly affect cache size because the number of sub-block-valid bits in a block is small compared to the tag and data area. For example, going from a 32,4 to a 32,8-cache decreases the total size by only 1.4 percent. A cache with the capability of varying sub-block size can be set to run at different operating points depending on the relative importance of miss ratio and traffic ratio. Such a cache requires somewhat more control and enough sub-block-valid bits for the smallest sub-block size. Figure 2 shows a solid line labeled $b32$ with net cache size 1024 that intersects all sub-block sizes between 2 and 32 bytes. These various sub-block sizes allow a system implementor to trade the miss ratio against the traffic ratio. In a system with considerable unused bus bandwidth, the sub-block size could be set to 32 bytes to realize miss and traffic ratios of 0.033 and 0.533 (Table 7).

In another system that is bus-limited either by a slower bus or more devices (processors) using the bus, the sub-block size could be set as low as 2 bytes. This would increase the miss ratio by a factor of 6 to 0.190 but decrease the traffic ratio by a factor of 3 to 0.190.

A *minimum* cache for these results is a 64-byte cache with 4-byte blocks and 2-byte sub-blocks. This 4,2 64-byte cache can cut memory references and bus traffic by one-third as compared to a system without a cache. On the other hand, a 16,8, 1024 byte cache results in miss and traffic ratios of 0.052 and 0.206.

4.2.2. Z8000 Results

Figures 3 and 4, and part of Table 7 show results from UNIX utilities written in C and compiled for the Z8000 (see last five traces in Table 3). These traces yield better performance than the PDP-11 traces; they are plotted on the same scale for comparison. The results quoted are warm-start ratios. *(Warm-start* ratios do not count the misses taken to initially fill the cache with relevant data.) As previously noted, the warm-start ratios are slightly optimistic.

As with the PDP-11 results, a 4,2 64-byte *minimum* cache is a start. A 16,8 1024-byte cache can be implemented a gross cost of 1264 bytes to yield miss and traffic ratios of 0.023 and 0.092. (These results for the Z8000 are so much better than for any other architecture and set of traces that we consider these results overly optimistic and unrepresentative. Since these same traces were used in [19] we have some doubts about the performance projections for the Z80,000.)

4.2.3. VAX-11 Results

Figure 5 and part of Table 7 show results from six VAX-11 traces (see Table 4). A 8,4 64-byte *minimum* cache has miss and traffic ratios of 0.6072 and 0.6072. A 16,8 1024-byte cache reduces these numbers to 0.1058 and 0.2116.

4.2.4. System/370 Results

Figure 6 and part of Table 7 show results from four System/370 traces (see Table 5). For clarity, Figure 6 is scaled differently than Figures 1 through 5. *Minimum* caches do not work well with our System/370 runs. A 8,8 64-byte cache will reduce memory reference in half (miss ratio 0.5475), but will actually slightly increase bus traffic with respect to a system with *no* cache (traffic ratio 1.0950). The best cache studied here for the System/370 traces was 16,8 1024-byte cache that has miss and traffic ratios of 0.2632 and 0.5264.

4.2.5. Inter-Architecture Comparisons

It is clear from Table 7 that miss ratios are generally increasing as one goes from the Z8000 to the PDP-11 to the VAX 11 to the System/370 trace driven simulation results. We believe that these differences are not reflective of the architectures, except for address space size, so much as the traces used. The Z8000 traces are all Unix utilities, ported from the PDP-11 version of Unix, and are mostly small, compact pieces of code. The PDP-11 programs are also relatively small, running as they do in a 16 bit address space. (In both cases, the 16 bit address is one cause of the good results.) The VAX programs are a mixture of small and large, and the System/370 programs are large, using hundreds of kilobytes of storage. We believe that the 370 programs are the most reliable indicator of actual performance for a 32-bit microprocessor, since "real" workloads, to be experienced in practice, are likely to contain many large, complex, memory intensive programs.

4.3. Nibble-Mode Results

All the traffic ratio results so far assume that the cost of a memory access is directly proportional to the number of bytes read. This assumption is accurate for many current microprocessor systems, often because of bus protocol compatibility constraints. However, some new memory chips provide paging or nibbling addressing modes, and some multiprocessor memory busses incur an overhead with each transaction. Both of these conditions will produce a cost to reference w sequential words of the form $a + b*w$. Therefore, the average cost is reduced if several words are simultaneously fetched from adjacent locations. On-chip caches can exploit nibble-mode memories for loading sub-blocks. Caches with larger sub-blocks derive more benefit than those with smaller sub-blocks. Bursky [17] reports that typical access times are 160 ns for the first word and 55 ns for subsequent words. If we approximate the ratio of 160 to 55 as 3 to 1 and assign unit-cost to getting one word, then the cost of getting w sequential words is $1 + \frac{1}{3}(w-1)$. A cache with a sub-block size of w words will always ask for w sequential words at an average cost of $\frac{1}{w}[1 + \frac{1}{3}(w-1)]$. The standard traffic ratio multiplied by the above factor produces a *scaled traffic ratio* for nibble-mode memories. This equation also reflects the cost of using a bus cycle to present an address to the memory before waiting 105 ns for the first word and 55 ns for subsequent words.

Figures 7 and 8 show PDP-11 results scaled to reflect the economies-of-scale for transferring more than one 16-bit word in a single access (See also, Table 7). Scaling with 16-bit words should put an upper bound on the improvement that can be expected from scaling with 32-bit words. Curves of constant block size and varying sub-block size (solid lines) minimize the scaled traffic ratio at a sub-block size of 4 or 8 bytes, rather than 2-byte result with the standard memory interface. Since the scaled traffic ratio penality for bringing in additional data is smaller than in the case of a standard memory interface, a cache designed for these addressing modes will tend to have a larger sub-blocks We observe that the optimum sub-block size under the assumptions of this section approximately doubles from the optimum results of Section 4.2.

4.4. Load-Forward Results

Load-forward is a mechanism for combining the miss ratio benefits of a large block size with the low bus traffic of sub-block placement. Load forward involves fetching the target sub-block and the subsequent sub-blocks within the same block; it is thus a limited form of prefetching. Load forward is being used in the 256-byte on-chip cache of the Z80,000 [19]. The Z80,000 cache has 16 blocks of 16 bytes. The blocks are replaced using an *LRU* stack on-chip. Data is fetched from memory in one-word (two-byte) sub-blocks with an option to *load-forward*

Program and data references within a cache block exhibit a forward bias. A program typically branches to a random location within a cache block, proceeds sequentially forward, and then branches again. Data references also tend to proceed forward because of processing of arrays, character strings, and individual variables whose storage is defined by the programmer in order of use [11]. For this reason, data just before a reference is less likely to be referenced than data just after it. Thus, a larger block size will result in more data being loaded from memory locations that are *before* the point of reference. Load-forward, by fetching only the part of the block forward of the target of the fetch, should be less likely to load unneeded information than a fetch which gets the entire block. Load forward may or may not remember which sub-blocks have already been loaded and if it does remember, it can load only those sub-blocks not already cache resident. Such optimized operation is more complex. The simpler scheme has redundant bus traffic in the few instances of a backwards reference within a cache block, but allows the main memory system to function autonomously.

Load-forward was studied with traces CPP, C1 and C2 using the latter redundant-load scheme. Since results (Figure 9 and Table 8) show that few redundant loads were made, there was not enough gain to justify experimenting with the optimized scheme. The point labeled $b16$–$s2$–LF–$g328$[7] on the 256-byte cache curve in Figure 9

[7] LF stands for *load–forward*, and $g328$ for gross cache size of 328 bytes.

corresponds to the performance of the Z80,000 cache for these particular benchmarks.

The load-forward mechanism reduces bus traffic at a small cost in miss ratio. For the Z80,000 design, changing from a sub-block size equal to the block size to a 2-byte sub-block size with load-forward reduces the traffic ratio by 20 percent for a cost of 7.0 percent in the miss ratio. Therefore, load-forward is useful if it is easy to implement and the traffic ratio is of some concern. We expect that load-forward will be especially effective in the instruction space.

5. Conclusions and Summary

We believe and the simulations presented in this paper suggest that some of the limited area on high-end microprocessor chips is best used as the top of the memory hierarchy rather than providing additional special-purpose logic.

Very small *minimum* on-chip caches are somewhat useful in reducing latency and off-chip traffic for all but the System/370 runs. A 64-byte cache with a block size of 2 words and a sub-block size of 1 word reduces memory accesses and bus traffic by one-third in PDP-11, Z8000, and VAX-11 runs. On the 32-bit VAX-11, this cache requires only 95 bytes of RAM cells to hold address tags and data. Unfortunately, a *minimum* cache of this size reduced System/370 misses by only 16 percent.

A more aggressive goal for a on-chip cache is to reduce references by a factor of ten (miss ratio 0.10) and bus traffic by a factor of five (traffic ratio 0.20). This is achieved with a 512-byte (net) cache with 4-byte blocks and 4-byte sub-blocks (4,4 512-byte) in PDP-11 runs, with a 8,4 512-byte cache in Z8000 runs, and with a 16,8 1024-byte cache in VAX-11 runs. The best cache studied here for the System/370 traces is a 16,8 1024-byte cache that can cut references by a factor of four and halve bus traffic.

It is possible to trade off the miss and traffic ratios by varying the sub-block size, for a fixed block size. As the sub-block size decreases, the miss ratio increases and the traffic ratio decreases.

When either nibble-mode (paged-mode) memories or a memory bus with transactional overhead are used, the bus traffic cost should be modeled as $a + b*w$, where w is the number of single words fetched on a single transaction. For nibble-mode memories that required 160 ns for the first word and 55 ns for subsequent words, the optimum sub-block size roughly doubled relative to to the optimum size found in other results.

Load-forward increases the traffic ratio slightly but cuts the miss ratio by a much larger factor, relative to the same block and sub-block sizes but without load forward. It appears to be advantageous if the processor and memory designs permit; compatibility with previous designs may be an inhibiting factor.

On-chip caches are one good use for the microprocessor chip area available for performance enhancements whenever the on-chip cache can be at least 32 to 128, words, depending on the workload. On-chip caches are especially effective when at least 256 words.

6. Acknowledgements

This paper is based upon work supported by the National Science Foundation under Grant MCS82-02591 and by the Defense Advanced Research Projects Agency under contract N00039-82-C-0235.

We would like to thank Dave Patterson, Yale Patt, Nick Tredennick, and many others for their suggestions. Susan Dentinger and Susan Eggers provided useful comments on drafts of this paper. Also thanks to John Lee for PDP-11 traces, John Lee, Bill Harding and the Amdahl Corporation for IBM System/370 traces, Juan Porcar and the Zilog Corporation for Z8000 traces, and Robert Henry for the VAX 11 traces.

References

1. A.J. Smith, "Cache Memories," *Computing Surveys*, vol. 14, no. 3, pp. 473 - 530, September, 1982.
2. J.S. Liptay, "Structural Aspects of the System/360 Model 85, Part II: The Cache," *IBM Systems Journal*, vol. 7, no. 1, pp. 15-21, 1968.
3. J. Bell, D. Casasent, and C.G. Bell, "An Investigation of Alternative Cache Organizations," *IEEE Trans. on Computers*, vol. C-23, no. 4, pp. 346-351, April 1974.
4. W.D. Strecker, "Cache Memories for PDP-11 Family Computers," *Proc. Third International Symposium on Computer Architecture*, pp. 155-158, January 1976.
5. J.R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," *Proc. Tenth International Symposium on Computer Architecture*, pp. 124-131, Stockholm, Sweden, June 1983.
6. N. Tredennick, *Personal Communication*, 1983.
7. J.K.F. Lee and A.J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer*, vol. 17, no. 1, pp. 6 - 22, January, 1984.
8. *VAX Hardware Handbook*, Digital Equipment Corporation, Maynard, Massachusetts 01754, 1980.
9. D.P. Siewiorek, C.G. Bell, and A. Newell, *Computer Structures: Principles and Examples*, New York, 1982. McGraw Hill
10. B.T. Hailpern and B.L. Hitson, *S-1 Architecture Manual*, January 1979. Stanford Univ. CSL Report STAN-CS-79-715.
11. A.J. Smith, "Sequential Program Prefetching in Memory Hierarchies," *Computer*, vol. 11, no. 12, pp. 7-21, December 1978.
12. D.A. Patterson, P. Garrison, M.D. Hill, D. Lioupis, C. Nyberg, T.N. Sippel, and K.S. Van Dyke, "Architecture of a VLSI Instruction Cache for a RISC," *Proc. Tenth International Symposium on Computer Architecture*, pp. 108-116, Stockholm, Sweden, June 1983.
13. M.G.H. Katevenis, R.W. Sherburne, D.A. Patterson, and C.H. Séquin, "The RISC II Micro-Architecture," *Proc. VLSI 83 Conference*, Trondheim, Norway, August 1983.
14. D.A. Patterson and C.H. Séquin, "RISC I: A Reduced Instruction Set VLSI Computer," *Proc. Eighth International Symposium on Computer Architecture*, pp. 443-457, Minneapolis, Minnesota, May 1981.
15. A.J. Smith, "A Compararative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 2, pp. 121-130, March 1978.
16. R.L. Mattson, J. Gecsei, D.R. Slutz, and I.L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78 - 117, 1970.
17. D. Bursky, "Innovative Chip Designs Lead to Dense, Superfast RAMs," *Electronic Design* , pp. 97-112, 18 August 1983.
18. M.D. Hill, *Evaluation of On-Chip Cache Memories*, December 1983. Master's Report, U.C. Berkeley.
19. D. Alpert, D. Carberry, M. Yamamura, Y. Chow, and P. Mak, "32-bit Processor Chip Integrates Major System Functions," *Electronics*, pp. 113-119, 14 July 1983.

Table 1. Simulation Parameters

Parameter	Values
cache size (bytes)	32, 64, 128, 256, 512, 1024
block size (bytes)	2, 4, 8, 16, 32, 64
sub-block size (bytes)	2, 4, 8, 16, 32
associativity	4-way
cache partitioning	data & instructions mixed
replacement algorithm	LRU
fetch algorithm	demand load-forward

Table 2. PDP-11 Workload

Trace	Language: Comments
OPSYS	C: toy operating system
PLOT	Fortran: printer plotter program
SIMP	Fortran: pipeline simulation program
TRACE	PDP-11 Assembly: tracing program tracing ED
ROFF	PDP-11 Assembly: text output and formatting program
ED	C: text editor

Table 3. Z8000 Workload

Trace	Language: Comments
CPP	C: first phase of C compiler
C1	C: second phase of C compiler
C2	C: third phase of C compiler
OD	C: Unix utility for dumping files in *ASCII*
GREP	C: Unix utility for string searching
SORT	C: Unix utility for sorting
LS	C: Unix utility for listing files
NM	C: Unix utility for printing a specially compiled object file's symbol table
PR	C: Unix utility for formating text files for printing

Table 4. VAX-11 Workload

Trace	Language: Comments
spice	Fortran: circuit simulation
otmdl	Pascal: constructs LR(0) parser
sedx	C: stream editor
qsort	C: Quick sort
troff	C: text formatter
c2	C: third phase of C compiler

Table 5. IBM System/370 Workload

Trace	Language: Comments
FGO1	Fortran Go Step: single precision factor analysis
FCOMP1	Compile of a program that solves Reynolds partial differential equation
PGO1	PL/I Go Step
PGO2	PL/I Go Step: program does CCW analysis

Table 6. IBM System/360 Model 85 Results
(Based on: 1 Fortran Go Step, 1 Fortran Compile, 2 Cobol Go Steps, and 2 PL/I Go Steps.)

Cache Organization	Miss Ratio	Relative to 360/85
360/85	0.0258	1.000
4-way	0.0088	0.341
8-way	0.0081	0.314
16-way	0.0076	0.294

Table 8. Load-Forward (LF) Results
(Based on three Z8000 traces: cpp, c1, and c2.)

Net, Gross Cache Size (bytes)	Block, Sub-Block Size (bytes)	Miss Ratio	Traffic Ratio	Traffic Ratio (nibble)
64				
94	8,8	0.257	1.028	0.514
97	8,2,LF	0.263	0.865	
97	8,2	0.678	0.678	0.678
192	2,2	0.612	0.612	0.612
256				
314	16,16	0.120	0.960	0.400
328	16,2,LF	0.128	0.772	
328	16,2	0.489	0.489	0.489
376	8,8	0.164	0.656	0.328
388	8,2,LF	0.169	0.567	
388	8,2	0.454	0.454	0.454
768	2,2	0.402	0.402	0.402

Table 7.

Net, Gross Cache Size (bytes)	Block, Sub-Block Size (bytes)	PDP-11 Miss Ratio	PDP-11 Traffic Ratio	PDP-11 Traffic Ratio (nibble)	Z8000 Miss Ratio	Z8000 Traffic Ratio	Z8000 Traffic Ratio (nibble)	VAX-11 Miss Ratio	VAX-11 Traffic Ratio	VAX-11 Traffic Ratio (nibble)	IBM System/370 Miss Ratio	IBM System/370 Traffic Ratio	IBM System/370 Traffic Ratio (nibble)
64													
79	16,8	0.399	1.596	0.798	0.330	1.320	0.660	0.4249	0.8498	0.5665	0.5794	1.1588	0.7725
80	16,4	0.557	1.114	0.743	0.508	1.016	0.677	0.6483	0.6483	0.6483	0.8726	0.8726	0.8726
82	16,2				0.857	0.857	0.857						
94	8,8	0.339	1.356	0.678	0.298	1.192	0.596	0.3892	0.7784	0.5189	0.5475	1.0950	0.7300
95	8,4	0.479	0.958	0.639	0.461	0.922	0.615	0.6072	0.6072	0.6072	0.8375	0.8375	0.8375
97	8,2	0.739	0.739	0.739	0.762	0.762	0.762						
126	4,4	0.425	0.850	0.567	0.432	0.864	0.576	0.5652	0.5652	0.5652	0.8180	0.8180	0.8180
128	4,2	0.666	0.666	0.666	0.671	0.671	0.671						
192	2,2	0.620	0.620	0.620	0.583	0.583	0.583						
256													
284	32,32	0.146	2.336	0.876	0.079	1.264	0.474	0.1528	1.2224	0.5093	0.2377	1.9016	0.7923
285	32,16	0.191	1.528	0.637	0.107	0.856	0.357	0.2061	0.8244	0.4122	0.3234	1.2936	0.6468
287	32,8	0.291	1.164	0.582	0.156	0.624	0.312	0.3003	0.6006	0.4004	0.4691	0.9382	0.6255
291	32,4	0.418	0.836	0.557	0.245	0.490	0.327	0.4759	0.4759	0.4759	0.7331	0.7331	0.7331
299	32,2	0.599	0.599	0.599	0.421	0.421	0.421						
314	16,16	0.144	1.152	0.480	0.082	0.656	0.273	0.1739	0.6956	0.3478	0.2722	1.0888	0.5444
316	16,8	0.204	0.816	0.408	0.124	0.496	0.248	0.2614	0.5228	0.3485	0.4006	0.8012	0.5341
320	16,4	0.302	0.604	0.403	0.203	0.406	0.271	0.4207	0.4207	0.4207	0.6300	0.6300	0.6300
328	16,2	0.478	0.478	0.478	0.355	0.355	0.355						
376	8,8	0.168	0.672	0.336	0.108	0.432	0.216	0.2367	0.4734	0.3156	0.3645	0.7290	0.4860
380	8,4	0.254	0.508	0.339	0.175	0.350	0.233	0.3596	0.3596	0.3596	0.5794	0.5794	0.5794
388	8,2	0.407	0.407	0.407	0.312	0.312	0.312						
504	4,4	0.218	0.436	0.291	0.157	0.314	0.209	0.3553	0.3553	0.3553	0.5438	0.5438	0.5438
512	4,2	0.351	0.351	0.351	0.287	0.287	0.287						
768	2,2	0.297	0.297	0.297	0.273	0.273	0.273						
1024													
1084	64,16	0.081	0.646	0.269	0.041	0.328	0.137	0.1088	0.4352	0.2176	0.2042	0.8168	0.4084
1092	64,8	0.118	0.472	0.236	0.063	0.252	0.126	0.1704	0.3408	0.2272	0.3092	0.6184	0.4123
1108	64,4	0.178	0.356	0.237	0.104	0.208	0.139	0.2825	0.2825	0.2825	0.4970	0.4970	0.4970
1108	64,2				0.104	0.104	0.104						
1136	32,32	0.033	0.533	0.200	0.013	0.208	0.078	0.0588	0.4704	0.1960	0.1266	1.0128	0.4220
1140	32,16	0.049	0.391	0.163	0.024	0.192	0.640	0.0863	0.3452	0.1726	0.1859	0.7436	0.3718
1148	32,8	0.075	0.298	0.149	0.039	0.156	0.078	0.1360	0.2720	0.1813	0.2855	0.5710	0.3807
1164	32,4	0.116	0.232	0.155	0.065	0.130	0.087	0.2267	0.2267	0.2267	0.4645	0.4645	0.4645
1196	32,2	0.190	0.190	0.190	0.047	0.047	0.047						
1256	16,16	0.033	0.265	0.110	0.013	0.104	0.043	0.0675	0.2700	0.1350	0.1700	0.6800	0.3400
1264	16,8	0.052	0.206	0.103	0.023	0.092	0.045	0.1058	0.2116	0.1411	0.2632	0.5264	0.3509
1280	16,4	0.081	0.162	0.108	0.039	0.078	0.052	0.1748	0.1748	0.1748	0.4308	0.4308	0.4308
1312	16,2	0.133	0.133	0.133	0.072	0.072	0.072						
1504	8,8	0.039	0.156	0.078	0.015	0.060	0.030	0.0804	0.1608	0.1072	0.2443	0.4886	0.3257
1520	8,4	0.061	0.122	0.081	0.030	0.060	0.040	0.1332	0.1332	0.1332	0.4017	0.4017	0.4017
1552	8,2	0.101	0.101	0.101	0.055	0.055	0.055						
2016	4,4	0.048	0.096	0.064	0.022	0.044	0.029	0.1044	0.1044	0.1044	0.3742	0.3742	0.3742
2048	4,2	0.081	0.081	0.081	0.045	0.045	0.045						
3072	2,2	0.072	0.072	0.072	0.037	0.037	0.037						

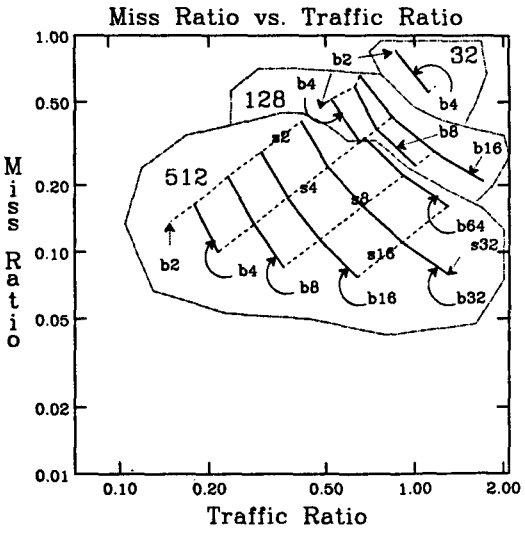


Figure 1. PDP-11 Results-Part I. Results for caches with net sizes of 32, 128, and 512 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

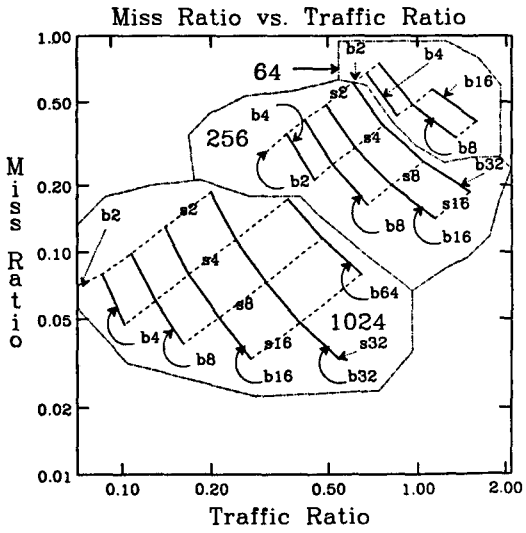


Figure 2. PDP-11 Results-Part II. Results for caches with net sizes of 64, 256, and 1024 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

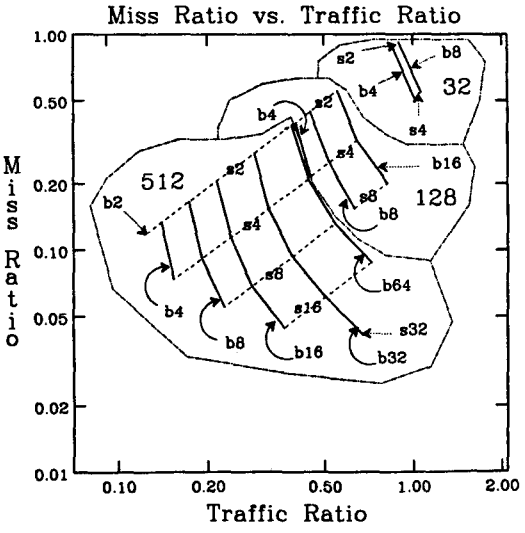


Figure 3. Z8000 Results-Part I. Results for caches with net sizes of 32, 128, and 512 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

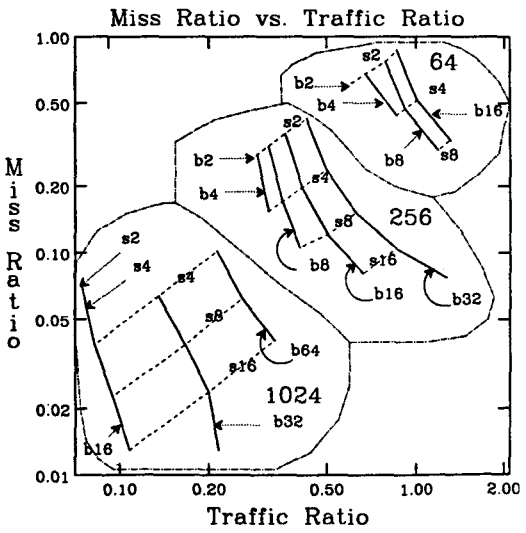


Figure 4. Z8000 Results-Part II. Results for caches with net sizes of 64, 256, and 1024 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

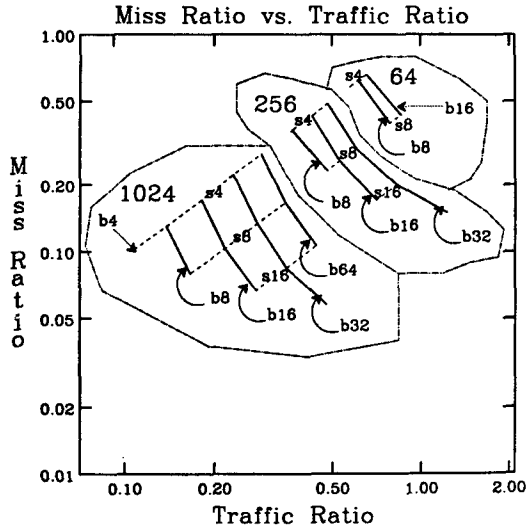


Figure 5. VAX-11 Results. Results for caches with net sizes of 64, 256, and 1024 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

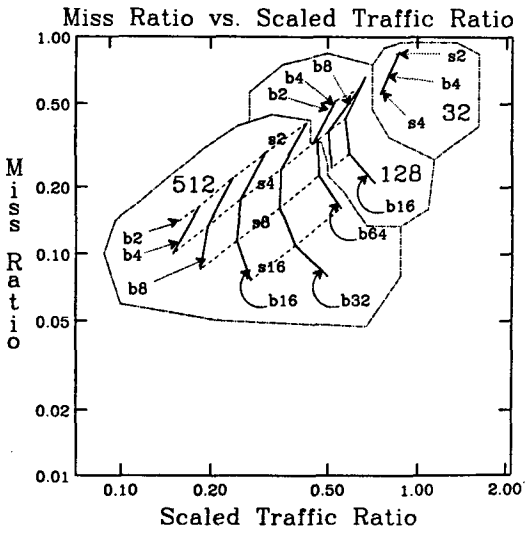
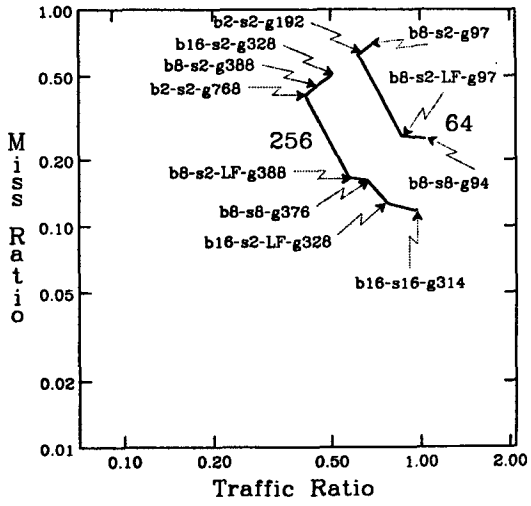


Figure 7. PDP-11 Results for Nibble Mode-Part I. Results for caches with net sizes of 32, 128, and 512 bytes. Nibble-mode assumes that an access for w sequential words costs $1 + \frac{1}{3}(w - 1)$.



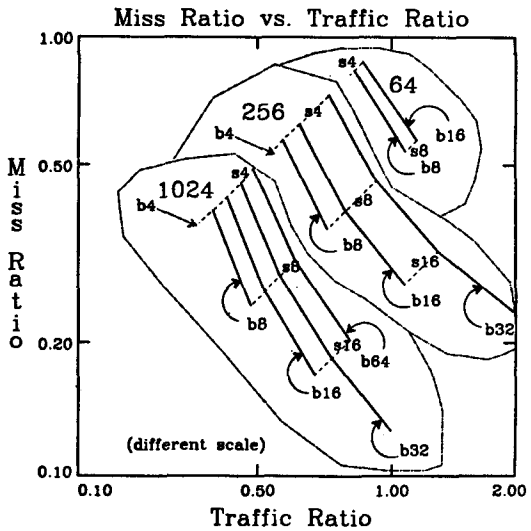


Figure 6. System/370 Results. Results for caches with net sizes of 64, 256, and 1024 bytes. Solid lines connect caches with constant block size *bx*. Dashed lines connect caches with constant sub-block size *sx*.

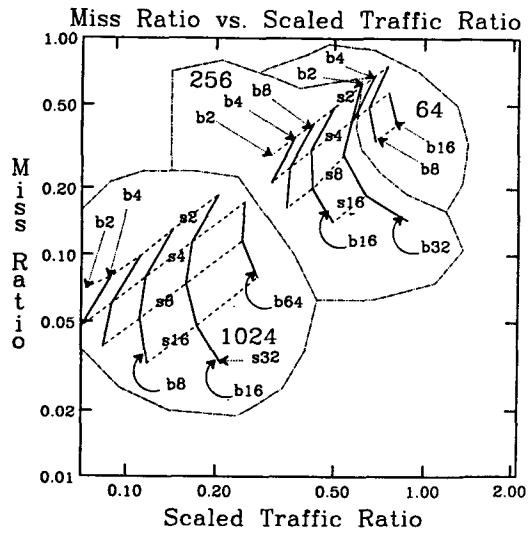


Figure 8. PDP-11 Results for Nibble Mode-Part II. Results for caches with net sizes of 64, 256, and 1024 bytes. Nibble-mode assumes that an access for w sequential words costs $1 + \frac{1}{3}(w - 1)$.

Miss Ratio vs. Traffic Ratio

Figure 9. Load-Forward Results. Results for caches with net sizes of 64 and 256 bytes. *bx* is block size, *sx* is sub-block size, LF stands for *load-forward*, and *gx* is the gross cache size.