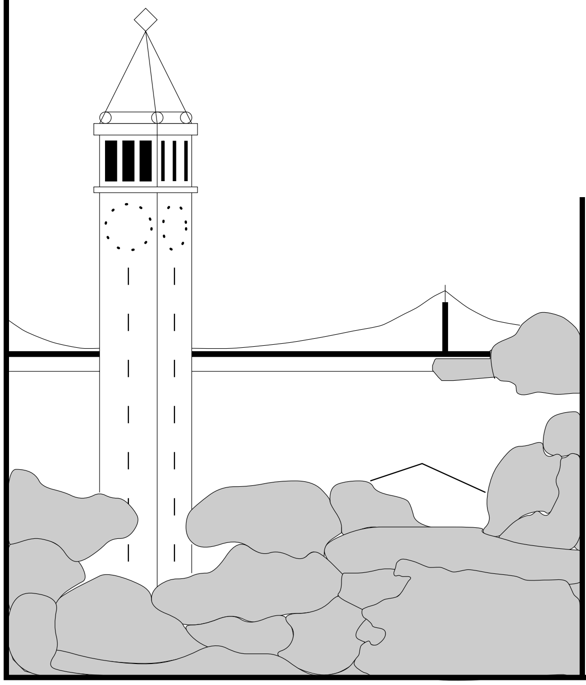
Sector Cache Design and Performance

Jeffrey B. Rothman and Alan Jay Smith



Report No. UCB/CSD-99-1034

January 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Sector Cache Design and Performance

Jeffrey B. Rothman and Alan Jay Smith*
Computer Science Division
University of California
Berkeley, CA 94720-1776

January 1999

Abstract

The IBM 360/85, possibly the first commercially available CPU with a cache memory, used a cache with a sector design, by which the cache consisted of sectors (with address tags) and subsectors (or blocks, with valid bits). It rapidly became clear that superior performance could be obtained with the now familiar set–associative cache design. Because of changes in technology, the time has come to revisit the design of sector caches.

Sector caches have the feature that large numbers of bytes can be tagged using relatively small numbers of tag bits, while still only transferring small blocks when a miss occurs. This suggests the use of sector caches for multilevel cache designs. In such a design, the cache tags can be placed at a higher level (e.g., on the processor chip) and the cache data array can be placed at a lower level (e.g., off–chip).

In this paper, we present a thorough analysis of the design and use of uniprocessor sector caches. We start by creating a standard workload and then we calculate miss ratios for a wide range of sector cache designs. Those miss ratios are transformed into Design Target Miss Ratios, which are intended to be "typical" miss ratios, suitable for use for design purposes ("design targets"). The miss ratios are then used to estimate performance, using typical timings, for a variety of one level and two level cache designs.

We find that for single level caches, sector caches are seldom advantageous. For multilevel cache designs with small amounts of storage at the first level caches, as would be the case for small on-chip caches, sector caches can yield significant performance improvements. For multilevel designs with large amounts of first level storage, sector caches provide relatively small improvements.

*Partial funding for this research has been provided by the State of California under the MICRO program, Sun Microsystems, Fujitsu Microelectronics, Toshiba, Microsoft, Cirrus, and Quantum.

1 Introduction

Cache memories are high speed buffers used to temporarily hold, for repeated access, portions of the contents of a larger and slower memory. Most modern caches are organized as a set of entries. Each entry consists of a block (or line) of data, and an address tag; the address tag is the location of the data in the larger (main) memory. The cache is accessed associatively - the key to the cache is not the location in the cache but the location in the main memory. To accelerate access, of course, the cache may not be fully–associative, but rather may be set–associative, direct–mapped or hashed.

Possibly the first commercial computer using a cache, the IBM 360/85 [Lip68] used a slightly more complex design, called a *sector cache*. A sector cache is organized as a set of *sectors*, and there is an address tag associated with each sector. The sector itself is divided into *subsectors*. Each subsector has a valid bit, and thus only some of the subsectors of a sector need to be present. When there is a miss to a sector, a resident sector is evicted, an address tag is set to the address of the new sector, and a single subsector is fetched. When there is a miss to a subsector, but the sector containing it is already present in the cache, only that needed subsector is fetched. This cache, in the IBM 360/85, was 16K bytes, and consisted of 16 sectors of 16 64–byte blocks.

Note that we use the terminology "sector" and "subsector". In [Lip68], these are known respectively as "sector" and "block." In [Goo83], these are called "address block" and "transfer block." In [HS84] these are called "block" and "subblock." We believe that the sector/subsector terminology is the clearest and least verbose. For normal, non-sectored caches, we refer to the unit of data transfer and addressing as either a "block" or a "line."

A sector in a sector cache appears in figure 1b. Figures 1a and 1c show standard cache blocks.

The original reason for the sector cache was technological; the discrete transistor logic of the time made a sector design easier to build than the currently more common non-sectored design. Unfortunately, the performance of the sector design in the 370/168 was inferior to the non-sectored design, as was shown in [HS84], and sector cache designs largely disappeared; one other machine with a sector cache was the Zilog Z8000 [ACY+83]. The problem with the sector design was that a sector would typically be evicted from the cache before all of

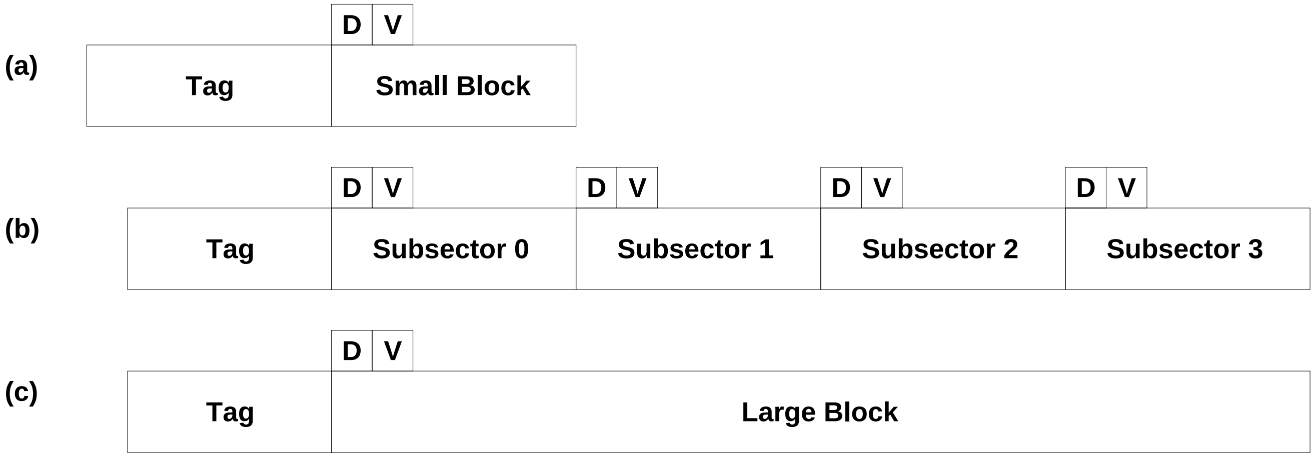


Figure 1: Diagram of a sector in a sectored data cache. (a) A standard cache entry, with a small block size. (b) A sectored cache entry, with four subsectors per sector. (c) A standard cache entry with a large block. V is the valid bit, and D represents a dirty bit.

its subsectors were loaded, and thus a large fraction of the cache capacity would be unused.

Sector caches do have, however, one important advantage. In a normal, (non-sectored) cache, the only way to have a very large cache capacity with a relatively small number of tag bits is to make the cache blocks (lines) very large; the problem in that case is that every miss requires that a large block be fetched in its entirety. With a sector cache, it is possible to fetch only a portion of a block (or sector), and thus the time to handle a miss, and the bus traffic, can both be significantly reduced. Thus, although it is likely that sector caches will have higher miss ratios than normal caches, there is the possibility that when timing is considered, the sector cache will be found to have better performance.

There have been a few previous studies of sector caches. Early studies include [Goo83, HS84, Goo87]. [Prz90a] provided a more extensive and thorough study, and found that in some cases that sector caches outperformed standard caches.

In this paper, we provide a more thorough and extensive study of uniprocessor sector cache designs than has previously appeared. We start by creating a standard workload and then we calculate (using trace driven simulation) miss ratios for a wide range of sector cache designs. Those miss ratios are transformed into Design Target Miss Ratios, which are intended to be an average and representative workload [Smi85, Smi87]. The miss ratios are then used to estimate performance, using typical timings, for a variety of one-level and two-level cache designs.

As we will show below in detail, we find that for single level caches, sector caches are seldom advantageous. For multilevel cache designs with small amounts of storage at the first level caches, as would be the case for small on-chip caches, sector caches can yield significant performance improvements. For multilevel designs with large amounts of first level storage, sector caches can provide small improvements.

2 Methodology

In evaluating sector caches, we wanted to be sure that we had a large, diverse, and realistic workload. We were particularly concerned that the size of the workload (in terms of the size of the address space referenced) be large enough to induce capacity misses from the largest caches studied. Our approach was to use a large set of varied workloads (many run from beginning to end) combined together into a single, multiprogrammed trace.

The results for this paper were generated using *trace driven simulation* (TDS) [Smi94]. All but one trace were generated on a MIPS R2000 based workstation using two trace generation tools (Cerberus [Rot] and Pixie [Smi91]). Many of the workloads were run to completion to create a full range of events for simulation. It was noted early on in the simulation process that the range of instruction and data spaces accessed by the individual traces generally was not big enough to fully exercise (and thus get meaningful results from) caches larger than 64K bytes (Table 27 in Appendix B). This problem was also noted in [HS89] for short traces. The solution we developed was to simulate a multiprogrammed environment, described in Section 2.2.

2.1 Workloads

A diverse group of 24 programs were chosen to drive the sector cache simulations. The characteristics of the programs can be found in Table 1. The workloads come from five basic categories: SPEC92 Integer, SPEC92 Floating Point, Unix utilities, Parallel Workloads run in uniprocessor mode, and an IBM/370 trace including user and supervisor memory references.

Five of the parallel traces come from the Stanford SPLASH applications [SWG92]. These workloads consist of **barnes**, a simulation of N-particle interactions; **cholesky**, the cholesky decomposition of a matrix; **locus**, a commercial quality VLSI circuit router; **mp3d**, a simulation of rarefied air-flow through a wind-tunnel with rectangular cross-section; and **water**, a hydro-dynamic simulation of water molecule interactions. The other parallel workloads are small parallel programs developed at U.C. Berkeley. These applications consist of **matrix**, a matrix multiply algorithm; **maxflow**, a calculation of the maximum flow possible through a graph; and **sor**, a solution to Laplace's equation using successive overrelaxation.

Description and Characteristics of Workloads						
Group	Program Name	Program Description	Unique Bytes Accessed (Thousands)	Percent Data	Total Refs (Millions)	Percent Data
SPLASH	barnes	N-particle interaction	48.0	48.6	13.7	27.0
	cholesky	Matrix decomposition	285.6	95.4	13.1	23.5
	locus	VLSI router	272.0	90.1	13.1	23.7
	mp3d	Rarified flow sim.	272.8	92.0	11.3	23.8
	water	Liquid dynamics	62.4	49.1	13.8	27.4
Berkeley	matrix	Matrix multiply	144.0	95.0	6.6	34.4
	maxflow	Graph flow	63.2	89.1	12.8	21.7
	sor	Laplace's equation	60.4	84.9	11.3	33.1
SPEC92 FLOAT	alvinn	Neural net	61.6	90.9	12.3	18.7
	doduc	Nuclear reactor sim.	189.2	52.4	13.8	27.6
	fpppp	Quantum chemistry	179.6	47.5	15.5	35.5
	tomcatv	Mesh generator	3223.2	99.7	14.7	31.9
SPEC92 INTEGER	cc1	GNU C compiler	326.4	47.1	13.0	23.4
	compress	File Compression	387.6	98.4	10.5	23.4
	eqntott	Truth table gen.	598.4	98.2	12.7	21.0
	espresso	Boolean minimizer	84.8	68.5	12.3	18.9
	xlisp	9 queens problem	70.4	71.8	13.6	26.6
MISC Traces	as1	Mips assembler	738.0	91.1	13.9	28.3
	cpp	C preprocessor	108.4	84.9	13.2	24.2
	fortran	FORTRAN compiler	497.2	81.7	12.9	22.5
	troff	Text formatter	117.6	64.9	12.8	21.6
	yacc	LR-1 Parser gen.	55.2	76.8	13.2	24.2
	MVStrace	IBM 370 OS trace	1645.2	85.8	23.6	47.3
Multiprogrammed Workload			9491.2	88.2	303.7	27.2

Table 1: Summary of program characteristics.

The SPEC92 Integer workloads include **cc1**, the gcc C compiler; **compress**, a file compression utility using an adaptive Lempel-Ziv algorithm; **eqntott**, a translator which generates a logical representation of a boolean equations into a truth table; **espresso**, which attempts to minimize the number of terms of a boolean function represented as a truth table; and **xlisp**, a lisp interpreter solving the 9 queens problem. The applications are written in

C and mainly use integer arithmetic in calculations.

The SPEC92 Floating-Point workloads include **alvinn**, a C-based neural net simulator that learns to maneuver an autonomous vehicle using net backpropagation; **doduc**, a simulation of the time evolution of a thermohydraulic modelization of a nuclear reactor's component using Monte Carlo methods; **fpppp**, a quantum chemistry program that measure the performance of a two electron integral derivative computation; and **tomcatv**, a vectorized mesh generation program. All the codes except alvinn were written in FORTRAN; and all use intensive floating–point calculations.

Another grouping of applications consists of a few commonly used UNIX utilities: **cpp**, the pre-processor for the standard UNIX C compiler; **fortran**, the MIPS FORTRAN compiler; **troff**, a text formatting program; and **yacc**, an LR(1) parser generator which takes a context–free grammar as input.

The last trace, **MVStrace**, consists of user and supervisor memory references from an IBM/370 using the MVS operating system. This trace is a concatenation of the complete MVS1 and MVS2 traces used in [Smi85] and [Smi87]. The traces other than MVStrace contain only user space activity.

Table 27 in Appendix B shows the data and instruction miss ratios for all of the individual traces run straight through without multiprogramming, and the multiprogrammed trace, using 16–byte cache blocks; as is explained below, the traces were combined into a single multiprogrammed trace for our studies. Over the range of cache sizes considered (4K to 512K), the individual instruction cache miss ratios reached their lower limits at fairly small cache sizes (approximately half of them with 16K caches) for the instruction streams from the samples we used. The situation was similar for the data caches, but for somewhat larger cache sizes. Appendix B shows the miss ratios of the individual workloads.

As noted in [Smi85], examining user space references exclusively results in over-optimistic estimates of performance for two main reasons: 1) task switching causes all or part of the cache to be flushed, making the performance of the memory system worse than would be predicted from a simple workload evaluation; 2) OS code is typically large, not very compact, and has frequent branches. It fact, it is well known that commercial workloads and operating systems workloads have miss ratios that are much higher than are observed for the types of programs typically studied by trace driven simulation [Jow98]. (Also see the discussion in

[GHPS93] about this issue.) For this reason we have simulated a multiprogramming workload using actually observed task switching patterns [Kob86] and have used the MVS trace to add some OS references into the cache simulation.

2.2 Multiprogrammed Traces

In order to obtain a trace which displayed miss ratio characteristics closer to those observed in practice, and also in order to ensure that the address space referenced was significantly larger than the largest cache to be studied, we combined all of the individual traces into a single multiprogrammed trace. This approach also thereby yielded an "average" over the constituent traces, without having to explicitly compute an average, and also more closely resembles the workload on a real machine, which is typically multiprogrammed.

A published study of task switching behavior [Kob86] was used as the basis for our multiprogramming model. In that paper, it was shown that the LRU stack model could be used to represent the sequence of active tasks. That is, let all of the tasks in the multiprogramming set be placed in an LRU stack. When it is time to select another process to run, the i'th task in the LRU stack is selected with probability p(i).

Probablistic and Cumulative Distribution of Task Selection										
Distribution	Position from Top of Stack									
	1	2	3	4	5	6	7	8	9	10
Probability	0.886	0.040	0.021	0.014	0.010	0.008	0.006	0.004	0.003	0.002
Cumulative	0.886	0.926	0.947	0.961	0.972	0.979	0.985	0.989	0.992	0.994

Table 2: Top 10 probability distribution for choosing a task to run from the stack.

We approximated the stack distance probability distribution p(i) from [Kob86] by the following equation:

$$p(x) = 0.05e^{-0.315x} + 0.85x^{-6.0} \tag{1}$$

where x is the MRU distance from the top of the stack ($x \in [1..n]$), 1 is the most recently used task. Table 2 shows an example of the probability function of choosing the top ten tasks in the stack.

To generate our multiprogrammed trace, each of the original traces was treated as a separate task. The tasks were organized into an LRU stack. The initial ordering of the task stack was arbitrary but consistent for all simulations. A task was chosen using the stack probability function (Equation 1), run for a time quantum, and put back on the top of the stack. The quanta had an exponential distribution with mean 20,000 memory references. Each trace was considered to reference a separate virtual address space, and each block in the cache was tagged with a 6-bit address space ID; thus the cache was not purged when the active task was switched. This process was repeated until all the traces were exhausted.

2.3 Simulated Timing

Some of our simulations were used to generate only miss ratios, and some incorporated timing in order to estimate performance. All times were in numbers of processor cycles, rather than in absolute real times. For the timing simulations, cache hits always returned results in a single cycle; the delay caused by cache misses depended on the particular timing being simulated. We considered two different main memory latency times (6 and 15 cycles) which would be appropriate for single level caches. For two–level caches we used a longer main memory access delay (24 cycles), with slower per word transfer times. The memory delay for single level caches was modeled as

$$Delay_1 = 1 + mr_1 * Miss\ Penalty_1, \tag{2}$$

where mr_1 is the miss ratio of the level one cache and $Miss\ Penalty_1$ is the penalty associated with a cache miss. The first part of the delay (value **1**) is the time (in processor cycles) it takes to do a look-up in the on-chip cache and retrieve the data for a hit, or determine that a miss has occurred. Note that we have ignored, as beyond the scope of this paper, the fact that the absolute cache access time will vary slightly with the cache size, due to increases in the levels of decoding logic and the increased capacitance on the bit-lines with more cache blocks.

For two–level caches, we modeled different latencies and transfer rates at each level:

$$Delay_2 = 1 + mr_1 * Miss\ Penalty_1 + mr_2 * Miss\ Penalty_2 \tag{3}$$

where mr_1 and mr_2 are the miss ratios at each level with respect to all processor references, with $Miss\ Penalty_1$ and $Miss\ Penalty_2$ the access delays between the L1 and L2 caches, and between the L2 cache and main memory, respectively. These two miss penalties are functions of the fetch size, and the latency and bandwidth of data transfers between various levels of the memory hierarchy.

Due to the computationally intensive nature of multilevel cache simulations, the design target miss ratios derived from single level simulations (mr_1 and mr_2) were used to generate the results for two–level caches. Using the miss ratios this way assumes that the contents of the L1 cache are a proper subset of the L2 cache. When the ratio of the sector sizes from the L1 and L2 caches is large, this inclusion principle of multilevel memory hierarchies can be violated [BW88]. Under such circumstances, a write–back from the L1 cache could cause an unanticipated extra L2 miss when the corresponding L2 sector and subsector are no longer present. This could possibly cause an underreporting in the L2 miss ratio, but we do not believe this will have a significant impact upon our results.

2.4 Cache Parameters

All of the simulations conducted here used a fully–associative cache. Although in real caches associativities of 1 to 4 are most common [Smi82], the use of a fully–associative design allows us to study the design of sector caches separately from the effects of limited associativity and conflict misses. As shown in [Smi78, HS89], the effects of limited associativity can be calculated from the fully–associative miss ratio function. In any case, the miss ratios for 4- and 8-way associativity are very close to those for fully–associative designs. The simulations used a true least recently used (LRU) replacement policy with demand fetching of data from memory, and allocation of sectors in the cache for write misses (write allocate).

Besides the usual cache design parameters of cache size and block size, we also study a factor called *degree of sectoring* for subdividing a block into smaller fetch units. The unit associated with the tag (the sector) is varied between 16 and 512 bytes. The subunit used for fetching (subsector) ranges between 4 and $min(sector\ size, 512)$ bytes. The range of cache sizes under test was 4K bytes to 512K bytes.

2.5 Tag Bits

The number of bits used to control the cache depends on a number of factors. These factors include the cache size, the number of sector frames (places to put a sector) in the cache, the set–associativity, bits for validity and dirtiness of the sectors, and the number of bits in the address. Instruction caches only need to determine if a subsector is valid; data and unified caches need an additional bit to determine if the subsector has been modified, requiring a write–back when the sector is evicted from the cache. For calculations involving the number of tag bits, we use the number of bits required to maintain the LRU replacement policy for an 8-way set–associative cache (although the simulation results were generated by a fully–associative cache simulator). The rest of the bits are those bits necessary to uniquely identify each sector (address tag bits) and to maintain status (valid and dirty bits). We refer to the combination of all these bits as the **tag bits**. Equation 4 (evaluated over the design space in Tables 17 and 18 in Appendix A) shows the number of tag bits necessary to control instruction and data/unified caches using 48-bit virtual addresses, which we use for comparing caches in this paper.

In set–associative caches, some number of bits are required to implement the replacement policy. In [Smi82], an efficient mechanism for implementing a close approximation to LRU (and also used in the IBM 370/168-3) is shown to require 10 bits per set. (The 8 elements of the set are divided into four pairs, with the pairs kept in LRU order using 6 bits. The LRU element of each pair is determined with one bit per pair.) Our calculations assume 10 bits per set.

The number of tag and other overhead bits used in further comparisons can be calculated by the following equation:

$$\begin{aligned} tag\ bits &= sectors * ((statebits * subsectors\ per\ sector) + \\ &\quad (addrbits + log_2 assoc - log_2 cache\ size) + (lru\ bits/assoc)) \end{aligned} \tag{4}$$

where $statebits$ is one for an instruction cache or two for data or unified caches, $addrbits$ the number of virtual address bits (we use 48), $assoc$ the set–associativity (we use 8–way for bit calculations), and $lru\ bits$ the number of bits to implement replacement policy for a set (10 bits per set as described above).

3 Simulation Results

In this section we study the various implementation–independent characteristics of sector caches, including miss ratios and memory traffic characteristics. From the miss ratios we perform a regression analysis to determine the importance of sector size, subsector size, and cache size on the miss ratio. The sector utilization is examined, which provides insight into why sector caches aid in reducing bus traffic without a correspondingly large increase in the miss ratio.

Using an extension of the techniques in [Smi82, Smi87], we develop design target miss ratios (DTMRs) for sector caches. The fraction of sector dirtiness is combined with design target miss ratios to create design target traffic ratios (DTTRs).

3.1 Sector Cache Miss Ratios

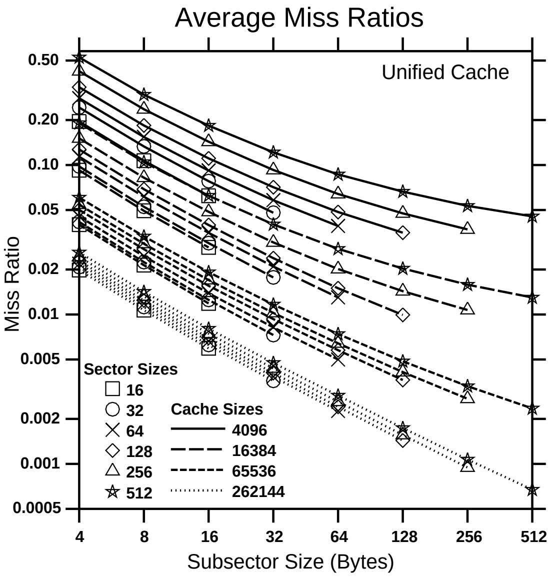


Figure 2: Unified cache miss ratios, cache sizes 4K, 16K, 64K, 256K.

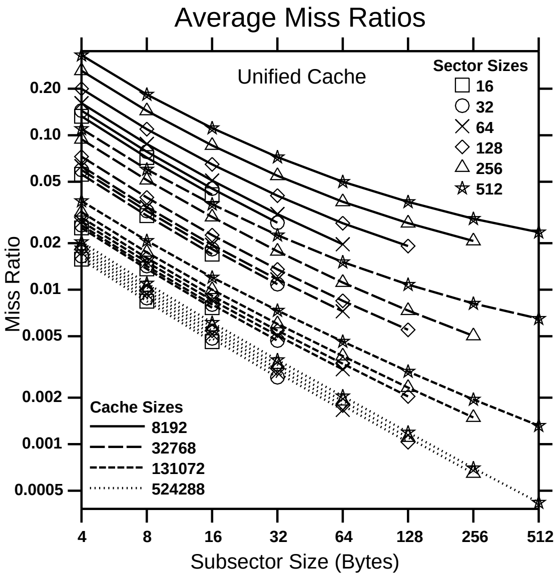


Figure 3: Unified cache miss ratios, cache sizes 8K, 32K, 128K, 512K.

One of the most widely used measures of cache memory performance, the miss ratio, consists of the ratio of references that are not satisfied by the cache to the total number of references. Unlike memory access delay (discussed in Section 4.2), the miss ratio provides an implementation–independent measure of cache performance. From the miss ratio a number of other useful statistics can be calculated, such as the *fetch traffic* (average bytes fetched per memory reference), *traffic ratio* and the average *memory access delay.* The traffic ratio is the ratio of traffic between the processor and main memory with and without a cache, which takes fetch size and eviction traffic into account. Memory access delay is the effective access time per memory reference, which includes such factors as the miss ratio, fetch size, memory latency and memory bandwidth.

Table 4 shows the miss ratios for unified, instruction and data caches determined by our cache simulations. The general features to notice about the miss ratios are that for a fixed subsector size, increasing the sector size makes the miss ratio worse, due to inflexibility in being able to map the subsectors into the cache. The best miss ratios for a given subsector is when it is equal to the sector size (as in non–sector caches). Since the graphs (Figures 2–3 here and Figures 9–12 in Appendix B) plotted from Table 4 on log-log scales look fairly linear, we ran several regressions to attempt to predict the miss ratio from the input parameters of cache, sector, and subsector size. The first (and more complex equation we tried) is:

$$\begin{aligned} \ln mr &= a_0 + a_1 \ln(s) + a_2 \ln(ss) + a_3 \ln(c) + a_4 \ln(ss)\ln(s) + \\ & a_5 \ln(c)\ln(s) + a_6 \ln(c)\ln(ss) + a_7 \ln(c)\ln(s)\ln(ss) \end{aligned} \tag{5}$$

where **c** is the cache size, **s** is the sector size, and **ss** is the subsector size, all sizes in bytes. This is an empirical equation that combines the factor parameters in a straight-forward manner. A somewhat simpler equation was also tested in a regression:

$$\ln mr = a_0 + a_1 \ln(s) + a_2 \ln(ss) + a_3 \ln(c) \tag{6}$$

and was found to be fairly close in predicting the unified and instruction miss ratios as measured by the R^2 closeness of fit (Table 3). An analytic derivation of the relationship between the miss ratio and the cache size in [Prz90b] predicts that the miss ratio should fall proportionally to c^{-1}, but the paper reports values of $c^{-0.49}$ and $c^{-0.54}$ taken from experiments. These values come from the combined weighted miss ratios of instruction and data streams using a Harvard Architecture (separate instruction and data caches). [Smi82] suggests a rule-of-thumb relationship between miss ratio and cache size as $mr(c) \propto c^{-0.5}$. Our measurements, using the simpler equation, roughly fall between the measured and predicted values in [Prz90b].

Cache Type	a_0 1	a_1 $\ln s$	a_2 $\ln ss$	a_3 $\ln c$	a_4 $\ln s \ln ss$	a_5 $\ln c \ln s$	a_6 $\ln c \ln ss$	a_7 $\ln c \ln s \ln ss$	R^2
Unified	4.0460	0.050836	-1.8133	-0.53821	0.29779	-0.0046421	0.063787	-0.020309	0.99070
	4.5010	0.21079	-0.69638	-0.68379					0.97033
Instruction	7.3820	-0.048458	-1.3189	-0.96138	0.12184	0.012602	0.023273	-0.0069225	0.99698
	6.4598	0.21207	-0.81150	-0.93710					0.99547
Data	2.8190	-0.32130	-1.9984	-0.35909	0.42251	0.021036	0.077520	-0.029813	0.98690
	2.7750	0.17412	-0.61138	-0.48752					0.92587

Table 3: Regression of the miss ratio from Table 4 for the three types of caches. c is cache size, s is sector size, and ss is subsector size.

The coefficient (in the second, simpler equation) associated with sector size (a_1) has similar magnitudes for the three types of caches. a_1 demonstrates the impact of increasing the sector size (independent of fetch size), which adversely affects the miss ratio when increased by displacing more data from the cache on sector eviction. The coefficient associated with subsector size (a_2), which serves as a measure of spatial locality, demonstrates the higher spatial locality of the instruction stream. Because $|a_2|$ is less than 1.0 (from Equation 6), the miss ratio does not decrease as rapidly as the subsector size increases, which results in an increase in traffic with increasing subsector size (as described in Section 3.4).

3.2 Subsector Utilization

Table 5 shows a sampling of the average fraction of subsectors referenced while its parent sector resides in the cache (the full version can be found in Table 19 in Appendix A). That is, it shows how many of the subsectors were loaded before the sector was evicted (replaced). Where the subsector and sector sizes are equal (i.e., the cache is not a sector cache), the fraction of subsectors referenced is 1.0, since all demand fetched blocks are used. An examination of each row in the table where the subsector size is four bytes shows the fraction of words touched in a sector of that size. Since the sector miss ratio (our miss ratios are for subsectors) is invariant across all subsector sizes for given cache and sector sizes, the fraction of words touched in a sector provides a good idea of the amount of spatial locality as block (sector) sizes increase. Smaller sector sizes have a high utilization of subsectors; increasing the sector size reduces the utilization. For 256– and 512–byte sectors, at no point do more than 50 percent of the words get referenced (on average) before the sector is evicted,

Miss Ratios for Unified, Instruction, and Data Sector Caches																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.19543	0.24247	0.27991	0.33167	0.42458	0.52508	0.13834	0.15318	0.20375	0.23059	0.26715	0.30956	0.17382	0.18898	0.22306	0.25496	0.29301	0.38723
	8	0.10687	0.13317	0.15406	0.18400	0.23620	0.29576	0.07284	0.08099	0.10843	0.12299	0.14290	0.16609	0.09829	0.10778	0.12751	0.14711	0.17360	0.23725
	16	0.06224	0.07800	0.09191	0.11078	0.14317	0.18356	0.03949	0.04422	0.06004	0.06837	0.07974	0.09329	0.06062	0.06742	0.08379	0.09900	0.11943	0.17502
	32		0.04794	0.05814	0.07103	0.09271	0.12167		0.02508	0.03447	0.03944	0.04617	0.05453		0.04392	0.05830	0.07256	0.08923	0.13705
	64			0.03908	0.04846	0.06399	0.08660			0.02053	0.02360	0.02786	0.03319			0.04311	0.05633	0.07079	0.11370
	128				0.03526	0.04750	0.06640				0.01486	0.01768	0.02118				0.04611	0.05895	0.09898
	256					0.03708	0.05330					0.01150	0.01396					0.05087	0.08764
	512						0.04534						0.00961						0.07975
8K	4	0.13253	0.14428	0.16111	0.19994	0.26153	0.32989	0.08690	0.09202	0.10403	0.11559	0.14259	0.17281	0.13311	0.14300	0.15763	0.17797	0.22229	0.24789
	8	0.07165	0.07823	0.08779	0.10948	0.14388	0.18361	0.04547	0.04827	0.05486	0.06115	0.07577	0.09205	0.07366	0.07965	0.08881	0.10125	0.12708	0.14592
	16	0.04122	0.04512	0.05088	0.06479	0.08588	0.11132	0.02441	0.02602	0.02979	0.03337	0.04168	0.05087	0.04475	0.04873	0.05510	0.06716	0.08536	0.10013
	32		0.02714	0.03087	0.04063	0.05468	0.07218		0.01445	0.01671	0.01885	0.02388	0.02941		0.03067	0.03548	0.04714	0.06280	0.07478
	64			0.01966	0.02694	0.03707	0.04998			0.00971	0.01105	0.01419	0.01759			0.02412	0.03525	0.04928	0.05960
	128				0.01917	0.02698	0.03703				0.00680	0.00893	0.01120				0.02774	0.04069	0.04975
	256					0.02068	0.02881					0.00575	0.00733					0.03506	0.04334
	512						0.02356						0.00504						0.03870
16K	4	0.09227	0.09782	0.11364	0.12717	0.15084	0.19084	0.05763	0.05992	0.06505	0.07255	0.08177	0.09883	0.11989	0.12273	0.12791	0.13686	0.15466	0.18547
	8	0.04941	0.05255	0.06131	0.06892	0.08233	0.10465	0.02994	0.03119	0.03399	0.03806	0.04310	0.05233	0.06558	0.06743	0.07080	0.07677	0.08748	0.10552
	16	0.02811	0.03001	0.03517	0.03969	0.04862	0.06242	0.01590	0.01662	0.01822	0.02052	0.02340	0.02860	0.03941	0.04074	0.04314	0.04755	0.05796	0.07154
	32		0.01770	0.02094	0.02382	0.03037	0.04009		0.00906	0.01001	0.01138	0.01310	0.01618		0.02493	0.02682	0.03036	0.04049	0.05334
	64			0.01300	0.01498	0.02014	0.02748			0.00568	0.00652	0.00760	0.00951			0.01746	0.02045	0.03022	0.04234
	128				0.00993	0.01432	0.02030				0.00388	0.00459	0.00586				0.01432	0.02375	0.03529
	256					0.01072	0.01583					0.00283	0.00368					0.01970	0.03076
	512						0.01299						0.00243						0.02767
32K	4	0.05582	0.05962	0.06303	0.07264	0.09420	0.10994	0.02887	0.03224	0.03610	0.03851	0.04153	0.04845	0.08613	0.09035	0.09308	0.09695	0.11812	0.12933
	8	0.03015	0.03232	0.03429	0.03955	0.05123	0.06009	0.01498	0.01682	0.01895	0.02026	0.02192	0.02569	0.04741	0.05022	0.05194	0.05437	0.06584	0.07335
	16	0.01698	0.01830	0.01951	0.02256	0.02957	0.03577	0.00792	0.00898	0.01021	0.01096	0.01191	0.01406	0.02726	0.02937	0.03057	0.03224	0.04020	0.04901
	32		0.01086	0.01166	0.01354	0.01774	0.02253		0.00491	0.00564	0.00609	0.00665	0.00792		0.01826	0.01919	0.02049	0.02535	0.03469
	64			0.00724	0.00847	0.01114	0.01510			0.00322	0.00349	0.00384	0.00463			0.01245	0.01354	0.01678	0.02618
	128				0.00550	0.00732	0.01079				0.00207	0.00229	0.00281				0.00915	0.01154	0.02077
	256					0.00502	0.00814					0.00140	0.00174					0.00836	0.01740
	512						0.00649						0.00112						0.01518
64K	4	0.03989	0.04162	0.04523	0.05014	0.05432	0.06058	0.01472	0.01574	0.01961	0.02135	0.02581	0.02990	0.06386	0.06657	0.07018	0.07261	0.07584	0.08131
	8	0.02139	0.02242	0.02453	0.02733	0.02971	0.03329	0.00768	0.00821	0.01019	0.01113	0.01358	0.01581	0.03523	0.03696	0.03947	0.04101	0.04304	0.04639
	16	0.01186	0.01252	0.01386	0.01557	0.01701	0.01919	0.00409	0.00437	0.00540	0.00592	0.00734	0.00862	0.02023	0.02145	0.02343	0.02451	0.02591	0.02813
	32		0.00728	0.00820	0.00932	0.01025	0.01167		0.00238	0.00291	0.00321	0.00407	0.00483		0.01309	0.01479	0.01564	0.01670	0.01836
	64			0.00499	0.00577	0.00641	0.00740			0.00161	0.00179	0.00232	0.00280			0.00950	0.01022	0.01111	0.01244
	128				0.00364	0.00412	0.00486				0.00103	0.00137	0.00168				0.00667	0.00746	0.00861
	256					0.00273	0.00332					0.00083	0.00103					0.00521	0.00623
	512						0.00235						0.00064						0.00468
128K	4	0.02511	0.02615	0.02763	0.02930	0.03177	0.03754	0.00722	0.00773	0.00836	0.00924	0.01028	0.01139	0.05480	0.05653	0.05842	0.06034	0.06318	0.06696
	8	0.01363	0.01423	0.01509	0.01604	0.01746	0.02074	0.00381	0.00408	0.00442	0.00489	0.00544	0.00604	0.03002	0.03108	0.03227	0.03348	0.03526	0.03773
	16	0.00765	0.00801	0.00855	0.00913	0.01001	0.01201	0.00206	0.00221	0.00240	0.00266	0.00296	0.00329	0.01700	0.01772	0.01855	0.01940	0.02065	0.02244
	32		0.00469	0.00505	0.00544	0.00601	0.00733		0.00122	0.00133	0.00148	0.00164	0.00184		0.01052	0.01116	0.01182	0.01280	0.01423
	64			0.00305	0.00332	0.00372	0.00462			0.00076	0.00084	0.00094	0.00105			0.00685	0.00739	0.00816	0.00929
	128				0.00204	0.00233	0.00296				0.00050	0.00055	0.00062				0.00461	0.00522	0.00610
	256					0.00149	0.00195					0.00033	0.00037					0.00343	0.00415
	512						0.00132						0.00023						0.00291
256K	4	0.01982	0.02076	0.02185	0.02287	0.02430	0.02601	0.00367	0.00404	0.00456	0.00536	0.00618	0.00702	0.04741	0.04860	0.05035	0.05201	0.05375	0.05602
	8	0.01069	0.01122	0.01184	0.01242	0.01322	0.01419	0.00194	0.00214	0.00242	0.00284	0.00327	0.00372	0.02562	0.02634	0.02739	0.02842	0.02951	0.03090
	16	0.00594	0.00625	0.00662	0.00697	0.00745	0.00804	0.00105	0.00116	0.00131	0.00154	0.00178	0.00202	0.01424	0.01471	0.01539	0.01609	0.01683	0.01776
	32		0.00360	0.00383	0.00406	0.00436	0.00474		0.00064	0.00073	0.00085	0.00099	0.00112		0.00845	0.00894	0.00946	0.01001	0.01070
	64			0.00226	0.00241	0.00261	0.00287			0.00042	0.00049	0.00057	0.00064			0.00528	0.00566	0.00607	0.00659
	128				0.00143	0.00156	0.00174				0.00028	0.00033	0.00038				0.00336	0.00366	0.00405
	256					0.00095	0.00107					0.00020	0.00023					0.00224	0.00253
	512						0.00067						0.00014						0.00161
512K	4	0.01576	0.01649	0.01743	0.01826	0.01922	0.02037	0.00174	0.00188	0.00211	0.00261	0.00310	0.00382	0.04012	0.04119	0.04263	0.04416	0.04558	0.04754
	8	0.00842	0.00882	0.00934	0.00981	0.01035	0.01099	0.00092	0.00100	0.00112	0.00138	0.00164	0.00203	0.02135	0.02201	0.02283	0.02374	0.02458	0.02574
	16	0.00461	0.00484	0.00515	0.00543	0.00574	0.00612	0.00050	0.00054	0.00061	0.00075	0.00089	0.00110	0.01161	0.01203	0.01253	0.01311	0.01365	0.01439
	32		0.00271	0.00290	0.00308	0.00328	0.00351		0.00030	0.00034	0.00042	0.00050	0.00061		0.00670	0.00702	0.00742	0.00778	0.00829
	64			0.00166	0.00178	0.00190	0.00205			0.00019	0.00024	0.00028	0.00035			0.00398	0.00425	0.00449	0.00484
	128				0.00103	0.00110	0.00119				0.00014	0.00017	0.00020				0.00243	0.00258	0.00280
	256					0.00065	0.00070					0.00010	0.00012					0.00151	0.00164
	512						0.00042						0.00007						0.00099

Table 4: Average miss ratios for sector caches. Cache, sector and subsector (SS) sizes are in bytes.

Fraction of Subsectors Used for Unified, Instruction, and Data Sector Caches										
Sector Size (Bytes)										
Size		Unified Cache			Instruction Cache			Data Cache		
Cache	SS	16	64	256	16	64	256	16	64	256
4K	4	0.78494	0.44771	0.17891	0.87573	0.62038	0.36297	0.71685	0.32337	0.08999
	16	1.00000	0.58802	0.24132	1.00000	0.73130	0.43336	1.00000	0.48589	0.14672
	64		1.00000	0.43141		1.00000	0.60565		1.00000	0.34786
	256			1.00000			1.00000			1.00000
8K	4	0.80379	0.51220	0.19757	0.88998	0.66967	0.38756	0.74367	0.40848	0.09907
	16	1.00000	0.64700	0.25953	1.00000	0.76705	0.45311	1.00000	0.57115	0.15216
	64		1.00000	0.44813		1.00000	0.61695		1.00000	0.35141
	256			1.00000			1.00000			1.00000
16K	4	0.82064	0.54636	0.21993	0.90617	0.71639	0.45100	0.76063	0.45788	0.12265
	16	1.00000	0.67644	0.28357	1.00000	0.80250	0.51620	1.00000	0.61775	0.18385
	64		1.00000	0.46986		1.00000	0.67059		1.00000	0.38346
	256			1.00000			1.00000			1.00000
32K	4	0.82200	0.54415	0.29298	0.91169	0.70137	0.46371	0.78989	0.46713	0.22080
	16	1.00000	0.67365	0.36782	1.00000	0.79335	0.53209	1.00000	0.61360	0.30057
	64		1.00000	0.55418		1.00000	0.68622		1.00000	0.50201
	256			1.00000			1.00000			1.00000
64K	4	0.84124	0.56623	0.31084	0.90048	0.76139	0.48635	0.78912	0.46186	0.22756
	16	1.00000	0.69399	0.38938	1.00000	0.83818	0.55289	1.00000	0.61679	0.31093
	64		1.00000	0.58697		1.00000	0.70050		1.00000	0.53325
	256			1.00000			1.00000			1.00000
128K	4	0.82079	0.56569	0.33344	0.87521	0.68968	0.48418	0.80565	0.53310	0.28777
	16	1.00000	0.70047	0.42006	1.00000	0.79059	0.55721	1.00000	0.67707	0.37619
	64		1.00000	0.62458		1.00000	0.70965		1.00000	0.59432
	256			1.00000			1.00000			1.00000
256K	4	0.83418	0.60430	0.40087	0.87199	0.68624	0.48745	0.83213	0.59606	0.37569
	16	1.00000	0.73242	0.49149	1.00000	0.78919	0.56148	1.00000	0.72896	0.47043
	64		1.00000	0.68852		1.00000	0.71439		1.00000	0.67888
	256			1.00000			1.00000			1.00000
512K	4	0.85492	0.65419	0.46439	0.86878	0.67833	0.49213	0.86430	0.66916	0.47291
	16	1.00000	0.77268	0.55517	1.00000	0.78466	0.56749	1.00000	0.78679	0.56641
	64		1.00000	0.73537		1.00000	0.72223		1.00000	0.74606
	256			1.00000			1.00000			1.00000

Table 5: Fraction of subsectors touched while a sector is in the cache, sampled over the test space. SS is the subsector size in bytes.

regardless of the cache size. We can also see that the spatial locality of the data stream is significantly worse than that of the instruction stream, particular for smaller caches with large sector sizes.

The sector utilization illustrates why some sector caches perform reasonably well. In our cache simulations only the demanded sectors are brought into the cache, and so at least one subsector must be touched while the sector is present in the cache. This constrains the utilization to be: $\frac{subsector\ size}{sector\ size} \leq utilization \leq 1.0$. If the utilization is closer to the lower bound, then much of the cache space is not being used, and the miss ratio of the cache will be close to that of a cache reduced in size by a factor equal to the number of subsectors per sector (e.g., given four subsectors per sector, the effective cache size will one fourth as big). If the utilization is closer to the upper bound (good spatial locality), then the miss ratio will be that of a cache using a full sector fetch, multiplied by the number of subsectors per sector.

For example, a 32K data cache with 256–byte sectors and 64–byte subsectors has a utilization fraction of 0.502. This more than twice the lower bound of 0.250 ($\frac{64}{256}$), but significantly less than 1.0. We can see (in Table 5) that 77.9 percent of the words in a 256–byte sector are not touched. Using subsector fetches in this case aids in reducing the useless traffic. Traffic is reduced by not fetching a whole 256–byte sector at a time, since on average two out of four subsectors are not referenced for 256–byte subsectors. However, the miss ratio is increased by a factor of 2.0, which is better than the worst cases 4.0 (all subsectors used) and 2.88 (only one subsector per sector used, the same miss ratio of an 8K cache with 64–byte blocks).

3.3 Design Target Miss Ratios

A problem with any study of cache and memory system design is to come up with a set of "typical" miss ratios, since a machine designer needs those numbers to work from. In [Smi85] a set of "design target miss ratios" (DTMRs) were proposed, as lying within the range of the various published measurements, and they were put forth as suitable for the role of "typical." Results in [GHPS93] suggest that they serve that purpose reasonably well. The original DTMRs were for a fully–associative cache with 16–byte lines. The DTMRs were extended to a wider range of line (block) sizes in [Smi87] and to varying degrees of associativity in [HS89]. In each case, the extrapolation method used "ratios of miss ratios." In that technique, a set of simulations were used to measure the ratio of the miss ratio for cache configuration X to that for configuration Y (i.e., RMR(X/Y)=MR(X)/MR(Y)) for those simulations, and then DTMR(X) was computed from DTMR(Y) using RMR(X/Y). The assumption was that although the absolute value of the miss ratios for a given configuration would depend strongly on the workload, the ratio of miss ratios could be expected to be much more stable, which was borne out by the measurements.

The ratios of miss ratios for the multiprogrammed workload are used here to extend the results from [Smi87] to provide DTMRs for sector caches. These DTMRs are used to estimate the design target traffic ratios (Table 21 in Appendix A) and design target memory delay for single (Tables 10 and 11, Tables 23 and 24 in Appendix A) and two–level caches (Tables 13, 14, 15, 16).

Several approximations were made to provide DTMRs outside the range of cache and block sizes in [Smi87]. For cache sizes above 32K bytes (the largest cache size in [Smi87]), we use a rule of thumb that the miss ratios drops roughly as $cache\ size^{-0.5}$ as suggested in [Smi82], which is borne out by our regression for the data miss ratio (Table 3), and is consistent with results reported in [Prz90b]. Our regression results show that the instruction cache miss ratio falls almost directly proportionally to the inverse of the cache size, but we use the more conservative inverse square root of the cache size for our approximations. It is known that instruction miss ratios are much higher for OS and database workloads [Jow98].

The new sector DTMRs are derived from the established DTMRs and the ratio of miss ratios using the formula:

$$DTMR_{sect}(cache, sect, subsect) = DTMR_o(cache, block) * \frac{MR(cache, sect, subsect)}{MR(cache, block, block)} \quad (7)$$

where $MR(cache, sect, subsect)$ are the miss ratio values established in the previous section by our simulations, $DTMR_{sect}$ is the new sector cache DTMR, and $DTMR_o$ is from the DTMR values established in [Smi87].

The DTMRs for sector caches generated from the method described above can be found in Table 6. The data points where the subsector and sector sizes are equal have the same value as the DTMRs in [Smi87] for caches from 4K to 32K bytes with block sizes from 16 to 128 bytes. The overall comparison of the simulated miss ratios (Table 4) with the DTMRs shows that the DTMRs are somewhat more pessimistic (i.e., higher), particularly for unified and instruction caches. This is consistent with observations in [GHPS93], where it was observed that the sorts of workloads used for trace analysis and benchmarking frequently had significantly lower miss ratios than workloads observed in practice, especially in commercial environments.

3.4 Traffic

One important function of the cache, besides speeding up the average memory access time, is to reduce the traffic on the memory bus. This is clearly a very important function for multiprocessor systems [Goo83], and also allows I/O and writes better access to the memory bus in uniprocessor systems. There are several different statistics that can be used to measure traffic, such as bus fetch traffic (average number of bytes fetched per memory reference) and the traffic ratio (the ratio of traffic for a system with a cache vs. a system

Design Target Miss Ratios for Unified, Instruction, and Data Sector Caches																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.37677	0.41477	0.42263	0.47032	0.60207	0.74457	0.35029	0.38482	0.42682	0.49652	0.57524	0.66655	0.28674	0.30122	0.27939	0.26541	0.30502	0.40310
	8	0.20604	0.22779	0.23261	0.26092	0.33494	0.41939	0.18443	0.20346	0.22716	0.26484	0.30771	0.35764	0.16214	0.17179	0.15971	0.15314	0.18072	0.24698
	16	0.12000	0.13342	0.13877	0.15709	0.20302	0.26030	0.10000	0.11109	0.12578	0.14721	0.17170	0.20087	0.10000	0.10746	0.10495	0.10306	0.12432	0.18220
	32		0.08200	0.08778	0.10073	0.13146	0.17253		0.06300	0.07220	0.08492	0.09940	0.11743		0.07000	0.07302	0.07554	0.09288	0.14267
	64			0.05900	0.06872	0.09074	0.12281			0.04300	0.05082	0.05999	0.07148			0.05400	0.05864	0.07369	0.11836
	128				0.05000	0.06736	0.09416				0.03200	0.03807	0.04561				0.04800	0.06137	0.10304
	256					0.05258	0.07558					0.02476	0.03006					0.05296	0.09124
	512						0.06430						0.02068						0.08302
8K	4	0.25721	0.26577	0.27044	0.26079	0.34112	0.43029	0.21360	0.23558	0.24644	0.27200	0.33554	0.40665	0.23798	0.24714	0.25489	0.20531	0.25645	0.28598
	8	0.13905	0.14411	0.14736	0.14279	0.18766	0.23949	0.11176	0.12359	0.12995	0.14389	0.17829	0.21661	0.13169	0.13766	0.14362	0.11681	0.14661	0.16835
	16	0.08000	0.08311	0.08540	0.08451	0.11202	0.14520	0.06000	0.06662	0.07057	0.07853	0.09807	0.11970	0.08000	0.08422	0.08910	0.07748	0.09847	0.11552
	32		0.05000	0.05182	0.05299	0.07132	0.09415		0.03700	0.03958	0.04435	0.05618	0.06919		0.05300	0.05738	0.05438	0.07245	0.08628
	64			0.03300	0.03513	0.04836	0.06519			0.02300	0.02600	0.03338	0.04140			0.03900	0.04067	0.05685	0.06875
	128				0.02500	0.03520	0.04830				0.01600	0.02101	0.02636				0.03200	0.04695	0.05740
	256					0.02698	0.03757					0.01353	0.01726					0.04045	0.05000
	512						0.03073						0.01186						0.04465
16K	4	0.19695	0.19896	0.20106	0.20483	0.24296	0.30738	0.18123	0.19175	0.20632	0.22449	0.25304	0.30583	0.18255	0.19203	0.19048	0.18158	0.20519	0.24607
	8	0.10545	0.10689	0.10848	0.11101	0.13260	0.16857	0.09414	0.09982	0.10781	0.11778	0.13338	0.16192	0.09985	0.10550	0.10544	0.10185	0.11607	0.14000
	16	0.06000	0.06104	0.06223	0.06393	0.07831	0.10054	0.05000	0.05319	0.05778	0.06350	0.07241	0.08850	0.06000	0.06374	0.06425	0.06308	0.07690	0.09491
	32		0.03600	0.03704	0.03836	0.04891	0.06457		0.02900	0.03176	0.03522	0.04055	0.05006		0.03900	0.03994	0.04028	0.05372	0.07077
	64			0.02300	0.02413	0.03244	0.04426			0.01800	0.02018	0.02352	0.02943			0.02600	0.02713	0.04010	0.05618
	128				0.01600	0.02307	0.03270				0.01200	0.01419	0.01813				0.01900	0.03151	0.04683
	256					0.01726	0.02550					0.00877	0.01140					0.02614	0.04081
	512						0.02093						0.00753						0.03671
32K	4	0.13152	0.13176	0.12189	0.11889	0.15418	0.17994	0.10940	0.11173	0.11222	0.13036	0.14058	0.16400	0.12638	0.12372	0.12706	0.12721	0.15499	0.16970
	8	0.07104	0.07144	0.06631	0.06473	0.08385	0.09834	0.05676	0.05830	0.05889	0.06859	0.07421	0.08696	0.06956	0.06877	0.07090	0.07134	0.08639	0.09624
	16	0.04000	0.04045	0.03772	0.03693	0.04839	0.05854	0.03000	0.03113	0.03173	0.03711	0.04033	0.04758	0.04000	0.04022	0.04172	0.04231	0.05274	0.06431
	32		0.02400	0.02255	0.02216	0.02904	0.03688		0.01700	0.01754	0.02062	0.02253	0.02683		0.02500	0.02620	0.02689	0.03327	0.04552
	64			0.01400	0.01387	0.01823	0.02471			0.01000	0.01182	0.01300	0.01568			0.01700	0.01777	0.02202	0.03435
	128				0.00900	0.01199	0.01766				0.00700	0.00776	0.00951				0.01200	0.01514	0.02725
	256					0.00822	0.01332					0.00474	0.00591					0.01097	0.02284
	512						0.01063						0.00379						0.01992
64K	4	0.09518	0.09697	0.08969	0.08761	0.09491	0.10584	0.07641	0.07961	0.08614	0.10294	0.12448	0.14417	0.08928	0.08992	0.08883	0.09234	0.09645	0.10341
	8	0.05102	0.05223	0.04864	0.04775	0.05191	0.05817	0.03986	0.04155	0.04478	0.05366	0.06547	0.07625	0.04925	0.04992	0.04996	0.05216	0.05473	0.05899
	16	0.02828	0.02917	0.02748	0.02720	0.02972	0.03353	0.02121	0.02213	0.02371	0.02855	0.03538	0.04157	0.02828	0.02898	0.02966	0.03118	0.03295	0.03578
	32		0.01697	0.01626	0.01628	0.01791	0.02039		0.01202	0.01279	0.01550	0.01961	0.02331		0.01768	0.01872	0.01989	0.02124	0.02335
	64			0.00990	0.01008	0.01120	0.01294			0.00707	0.00863	0.01121	0.01350			0.01202	0.01300	0.01413	0.01582
	128				0.00636	0.00719	0.00849				0.00495	0.00662	0.00809				0.00849	0.00949	0.01094
	256					0.00477	0.00579					0.00400	0.00494					0.00662	0.00792
	512						0.00411						0.00309						0.00595
128K	4	0.06566	0.06698	0.06336	0.06474	0.07019	0.08293	0.05251	0.05373	0.05517	0.06518	0.07247	0.08032	0.06445	0.06715	0.07250	0.07859	0.08230	0.08722
	8	0.03564	0.03644	0.03460	0.03544	0.03857	0.04582	0.02774	0.02839	0.02918	0.03451	0.03838	0.04261	0.03530	0.03692	0.04005	0.04361	0.04592	0.04915
	16	0.02000	0.02052	0.01961	0.02018	0.02211	0.02654	0.01500	0.01536	0.01581	0.01873	0.02085	0.02320	0.02000	0.02105	0.02302	0.02526	0.02690	0.02923
	32		0.01200	0.01159	0.01201	0.01328	0.01620		0.00850	0.00877	0.01041	0.01160	0.01294		0.01250	0.01385	0.01540	0.01667	0.01854
	64			0.00700	0.00733	0.00822	0.01021			0.00500	0.00595	0.00664	0.00744			0.00850	0.00962	0.01062	0.01210
	128				0.00450	0.00514	0.00653				0.00350	0.00391	0.00440				0.00600	0.00680	0.00794
	256					0.00329	0.00430					0.00234	0.00264					0.00447	0.00541
	512						0.00291						0.00163						0.00379
256K	4	0.04719	0.04897	0.04786	0.05075	0.05393	0.05772	0.03700	0.03772	0.03882	0.04657	0.05370	0.06105	0.04707	0.05084	0.05732	0.06565	0.06785	0.07072
	8	0.02545	0.02646	0.02592	0.02755	0.02934	0.03149	0.01957	0.01997	0.02057	0.02465	0.02847	0.03235	0.02544	0.02755	0.03119	0.03588	0.03725	0.03901
	16	0.01414	0.01475	0.01450	0.01548	0.01653	0.01783	0.01061	0.01083	0.01116	0.01336	0.01547	0.01758	0.01414	0.01538	0.01753	0.02031	0.02124	0.02242
	32		0.00849	0.00839	0.00901	0.00968	0.01053		0.00601	0.00620	0.00741	0.00860	0.00978		0.00884	0.01018	0.01194	0.01263	0.01351
	64			0.00495	0.00536	0.00579	0.00636			0.00354	0.00423	0.00492	0.00559			0.00601	0.00714	0.00766	0.00832
	128				0.00318	0.00347	0.00386				0.00247	0.00289	0.00329				0.00424	0.00462	0.00511
	256					0.00210	0.00238					0.00172	0.00196					0.00282	0.00320
	512						0.00149						0.00120						0.00204
512K	4	0.03420	0.03648	0.03663	0.03988	0.04198	0.04449	0.02606	0.02640	0.02713	0.03277	0.03889	0.04797	0.03457	0.03841	0.04550	0.05452	0.05627	0.05869
	8	0.01827	0.01952	0.01964	0.02144	0.02261	0.02401	0.01382	0.01402	0.01441	0.01737	0.02062	0.02541	0.01839	0.02052	0.02437	0.02931	0.03034	0.03178
	16	0.01000	0.01071	0.01082	0.01186	0.01255	0.01336	0.00750	0.00763	0.00785	0.00944	0.01121	0.01379	0.01000	0.01122	0.01338	0.01619	0.01685	0.01776
	32		0.00600	0.00610	0.00674	0.00716	0.00766		0.00425	0.00437	0.00524	0.00624	0.00766		0.00625	0.00750	0.00916	0.00961	0.01024
	64			0.00350	0.00389	0.00416	0.00447			0.00250	0.00299	0.00357	0.00437			0.00425	0.00525	0.00555	0.00597
	128				0.00225	0.00241	0.00260				0.00175	0.00209	0.00255				0.00300	0.00319	0.00346
	256					0.00141	0.00153					0.00123	0.00151					0.00186	0.00203
	512						0.00092						0.00091						0.00122

Table 6: Design target miss ratios for single level sector caches. Cache, sector and subsector (SS) sizes are in bytes.

without a cache). We use the traffic ratio, which incorporates write–back traffic, utilizing write–back information from our simulations (discussed in the next section).

3.4.1 Eviction Traffic

Fraction of Sector Written Before Eviction							
Sector Size (Bytes)							
Size		Unified Cache			Data Cache		
Cache	SS	16	64	256	16	64	256
4K	4	0.13113	0.05905	0.02009	0.35851	0.14718	0.04047
	16	0.18793	0.08944	0.03373	0.49086	0.20732	0.06146
	64		0.18646	0.07977		0.36598	0.13176
	256			0.23112			0.34968
8K	4	0.15740	0.08931	0.02572	0.37225	0.21086	0.04411
	16	0.21798	0.12561	0.03878	0.51941	0.29053	0.06294
	64		0.21326	0.08186		0.46202	0.12854
	256			0.21407			0.32683
16K	4	0.18729	0.11038	0.03714	0.38235	0.22769	0.06309
	16	0.25427	0.15165	0.05260	0.53491	0.31927	0.08792
	64		0.24027	0.09008		0.49191	0.14310
	256			0.18931			0.27952
32K	4	0.22688	0.14043	0.06619	0.45457	0.26755	0.11524
	16	0.29505	0.18391	0.08946	0.57857	0.34984	0.15915
	64		0.29027	0.13992		0.54452	0.24711
	256			0.25706			0.42594
64K	4	0.27203	0.17293	0.08450	0.53618	0.30795	0.14854
	16	0.34288	0.22556	0.11085	0.66617	0.40216	0.19440
	64		0.34882	0.17523		0.61786	0.30373
	256			0.29254			0.48679
128K	4	0.36441	0.24073	0.13127	0.55999	0.36482	0.19358
	16	0.45070	0.30381	0.16789	0.68570	0.45446	0.24519
	64		0.44111	0.25192		0.64258	0.36107
	256			0.38692			0.54838
256K	4	0.41307	0.28352	0.17775	0.60161	0.42157	0.26077
	16	0.49646	0.34634	0.22036	0.71349	0.50702	0.31955
	64		0.47464	0.31116		0.67203	0.44175
	256			0.44330			0.61999
512K	4	0.47531	0.34498	0.23192	0.66215	0.50041	0.34630
	16	0.55376	0.40675	0.27744	0.75547	0.58025	0.40815
	64		0.51997	0.36517		0.71822	0.52220
	256			0.48213			0.67359

Table 7: Fraction dirtiness of evicted sectors, sampled over the cache parameter space.

The total memory traffic includes not only fetches, but also the write–back of replaced blocks and subsectors which are dirty. It was estimated in [Smi85] that approximately 50 percent of data cache blocks evicted are dirty (with large variation over the workloads). It was also noted that the percentage of dirty blocks was expected to increase with cache size, since a larger residence time in the cache should increase the probability of a write occurring to a block before it is evicted. In the seven program workloads examined in [TS89], it was found that 51.9 percent of all evicted (replaced) data blocks needed to be written back during the execution of the program (with wide variation, averaged over a number of cache sizes and workloads). With cache flushed every 20,000 references to simulate multiprogramming behavior, this number dropped to 38.9 percent. It was also found that increasing the cache size to some extent causes an increasing percentage of the evicted blocks to be dirty, but it

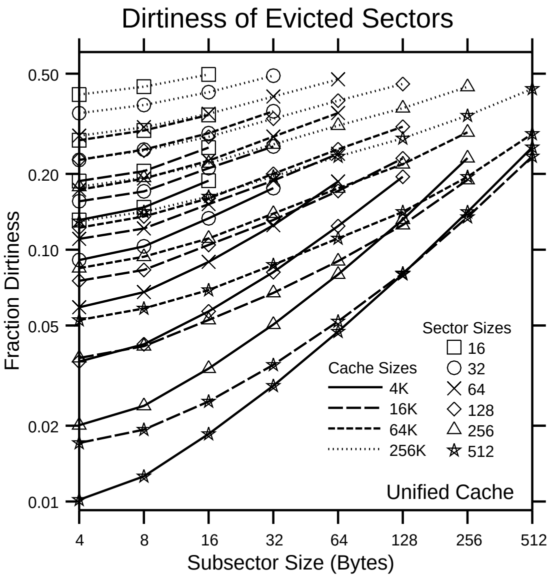


Figure 4: Fraction of modified sectors evicted from a unified cache.

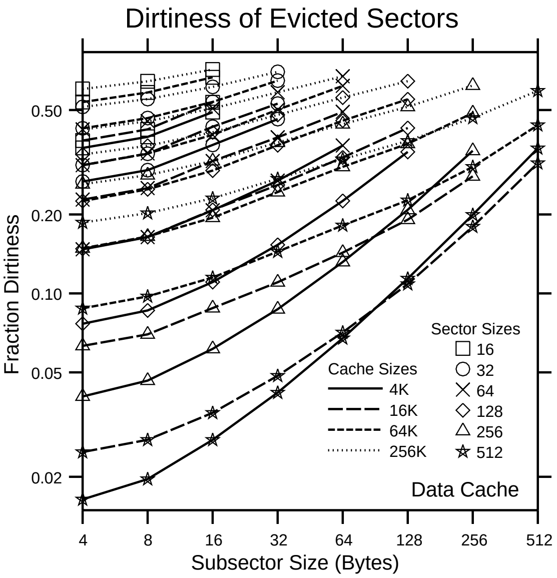


Figure 5: Fraction of modified sectors evicted from a data cache.

was not uniformly true. Both of these studies used 16–byte blocks.

Table 7 (full size in Table 20 in Appendix A) shows the fraction of evicted subsectors that are dirty, and thus must be written back. The percentage of sectors that have had some portion of each written can be determined by examining the points where the sector and subsector sizes are equal. Our study shows that the percent of dirty evicted blocks using 16–byte blocks for a data cache varies from 49.1 percent for a 4K cache to 75.5 percent for 512K cache; for a unified cache the numbers are 18.8 and 55.4 percent, respectively. The average data cache dirtiness we found is higher than the averages reported in [Smi85, TS89], but our workload and cache sizes are much larger. Our study did confirm the general trend that larger cache sizes have a larger percentage of dirty evicted blocks.

Our results show that the average fraction of sector dirtiness falls with decreasing subsector size (Figure 4 for unified caches, Figure 5 for data caches). In a normal (non-sectored) cache, a write to any word of a block requires that the entire block be written back. For a sector cache, only the modified subsectors need to be written back. In some cases, the use of a sector cache can significantly reduce the write traffic. As an extreme case, consider a 4K unified cache with a 512–byte sector; in this case, the sector has a 25.5 percent chance of some part of it being written during its short duration in the cache, but when it is possible to detect writes with single word granularity, on the average only a very small portion of the dirty sectors are actually dirty (3.98 percent or 5.09 words out of 128 words possible).

With the values for the eviction of dirty data determined here and the DTMRs developed in the previous section, we create the design target traffic ratios in the next section.

3.4.2 Design Target Traffic Ratio

Design Target Traffic Ratios for Unified, Instruction, and Data Sector Caches										
Sector Size (Bytes)										
Size		Unified Cache			Instruction Cache			Data Cache		
Cache	SS	16	64	256	16	64	256	16	64	256
4K	4	0.43971	0.47837	0.66968	0.35029	0.42682	0.57524	0.43014	0.40656	0.44220
	16	0.57021	0.63952	0.92557	0.40000	0.50313	0.68678	0.59633	0.59894	0.70558
	64		1.12001	1.72019		0.68800	0.95983		1.18020	1.62563
	256			4.14295			1.58479			4.57462
8K	4	0.30758	0.31760	0.38552	0.21360	0.24644	0.33554	0.35708	0.38646	0.37063
	16	0.38975	0.40794	0.51505	0.24000	0.28227	0.39229	0.48619	0.53767	0.55683
	64		0.64060	0.91506		0.36800	0.53413		0.91228	1.24242
	256			2.09616			0.86577			3.43469
16K	4	0.24190	0.24168	0.28399	0.18123	0.20632	0.25304	0.27429	0.28518	0.31075
	16	0.30102	0.30473	0.37136	0.20000	0.23112	0.28962	0.36834	0.38977	0.45466
	64		0.45642	0.61856		0.28800	0.37625		0.62060	0.88094
	256			1.31381			0.56107			2.14069
32K	4	0.16781	0.15334	0.18900	0.10940	0.11222	0.14058	0.19905	0.19980	0.23586
	16	0.20719	0.19208	0.24063	0.12000	0.12694	0.16131	0.25249	0.26201	0.32267
	64		0.28901	0.36525		0.16000	0.20804		0.42004	0.52580
	256			0.66149			0.30317			1.00085
64K	4	0.12592	0.11706	0.12070	0.07641	0.08614	0.12448	0.14979	0.14798	0.15937
	16	0.15188	0.14562	0.15272	0.08485	0.09483	0.14151	0.18832	0.19588	0.21413
	64		0.21360	0.23271		0.11314	0.17929		0.31101	0.35466
	256			0.39462			0.25595			0.63004
128K	4	0.09471	0.09026	0.09779	0.05251	0.05517	0.07247	0.10899	0.12194	0.13756
	16	0.11593	0.11240	0.12373	0.06000	0.06325	0.08340	0.13454	0.15366	0.17758
	64		0.16129	0.18446		0.08000	0.10622		0.22307	0.27305
	256			0.29187			0.14967			0.44253
256K	4	0.07034	0.07018	0.07776	0.03700	0.03882	0.05370	0.08063	0.09748	0.11468
	16	0.08440	0.08527	0.09566	0.04243	0.04464	0.06186	0.09637	0.11840	0.14235
	64		0.11656	0.13434		0.05657	0.07871		0.16019	0.20194
	256			0.19396			0.11018			0.29194
512K	4	0.05276	0.05564	0.06273	0.02606	0.02713	0.03889	0.06015	0.07868	0.09679
	16	0.06163	0.06568	0.07501	0.03000	0.03139	0.04484	0.06919	0.09198	0.11516
	64		0.08465	0.09915		0.04000	0.05707		0.11562	0.14988
	256			0.13354			0.07902			0.19781

Table 8: Design target traffic ratios. This shows the effectiveness of the cache in reducing bus traffic (fetch and write–back).

The traffic ratio measures the effectiveness of the cache in reducing the processor's bus bandwidth demands. Table 21 in Appendix A (sampled here in Table 8) shows the design target traffic ratios over the experimental cache space. These ratios include the traffic brought into the cache on a cache miss (demand fetches) as well as write–back traffic caused by the eviction of dirty sectors from the cache. The traffic ratio is calculated by multiplying the miss ratio by the number of words that are transferred to fulfill a cache miss and multiplying by a value to factor in the write–back traffic caused by evicted subsectors.

Our measurements assume that each unbuffered reference would fetch a four–byte word from memory and each cache miss fetches a subsector from main memory. A traffic ratio less than 1.0 indicates that the cache aids in reducing memory traffic, meaning that subsectors reside in the cache long enough to overcome the penalty for reading (and writing back of dirty subsectors) from memory in multi–word units. As would be expected, single word subsectors generate the least amount of traffic for all cache sizes. Only the words actually demanded are brought into the cache, and dirtiness can be measured with extreme precision for write–backs to memory.

The average number of times that the subsector is accessed must be more than the number of words in the subsector, or the cache is fetching too many useless bytes and degrading system performance. Large sectors in conjunction with small cache sizes (regardless of subsector size) result in extremely poor traffic ratios, which may not be directly obvious by examination of the miss ratios.

Large fetch (subsector) sizes for smaller caches are particularly ineffective, because the information brought into the cache cannot be properly exploited before it is ejected to make room for other demanded information. This is analogous to the thrashing that takes place when the working set of pages for a process in virtual memory does not fit in physical memory. In the case of sector caches, the inflexible assignment of multiple subsectors to an address tag exacerbates the problem.

The lowest amount of fetch traffic for a given sector size always corresponds to the smallest subsector size. The traffic ratio increases with both sector and subsector size and decreases with cache size. However, the lowest levels of fetch traffic per memory reference do not necessarily correspond to the best system performance. The spatial locality of reference and the overhead of initiating bus access operations must be taken into account and balanced with the time to fetch the subsector.

As will be seen more clearly below, small fetch sizes are best for systems with low memory access latency or for information with poor spatial locality (such as data). As the bus latency increases, larger fetch sizes show better performance. Large fetches are preferred for information streams with good spatial locality (e.g., instructions).

4 Implementation–Dependent Results

Given the implementation–independent statistics calculated in the previous section, we develop further statistics that depend on memory bus timing, such as design target bus utilization and memory delay to aid in characterizing the performance of sector caches. After looking at the basic characteristics of sector caches, we pose and answer the following question: under what circumstances are sector caches useful? How would a designer evaluate the trade–offs for the various cache organizations? Using memory delay as the primary metric, we evaluate the best designs for single and two level cache systems given a certain transistor budget.

4.1 Bus Utilization

Design Target Bus Utilization for Unified, Instruction and Data Sector Caches										
Sector Size (Bytes)										
Size		Unified Cache			Instruction Cache			Data Cache		
Cache	SS	16	64	256	16	64	256	16	64	256
4K	4	0.89083	0.89352	0.92543	0.85122	0.87455	0.90381	0.86108	0.84529	0.86052
	16	0.77844	0.78644	0.85087	0.67033	0.71891	0.77734	0.74499	0.73197	0.77317
	64		0.76611	0.85592		0.60973	0.68550		0.73484	0.81346
	256			0.97767			0.71301			0.94744
8K	4	0.84030	0.83393	0.86484	0.77722	0.80100	0.84569	0.82888	0.82631	0.82919
	16	0.68884	0.67681	0.73606	0.54955	0.58931	0.66601	0.68663	0.68649	0.71183
	64		0.61452	0.71378		0.45524	0.54811		0.64896	0.74176
	256			0.85006			0.57578			0.88738
16K	4	0.79494	0.78465	0.81169	0.74748	0.77116	0.80518	0.78612	0.77621	0.78643
	16	0.61501	0.59782	0.64455	0.50413	0.54020	0.59551	0.62180	0.60661	0.64356
	64		0.51706	0.59278		0.39541	0.46074		0.54583	0.64222
	256			0.70845			0.46797			0.78107
32K	4	0.71274	0.68344	0.72933	0.64118	0.64701	0.69662	0.71060	0.69585	0.73120
	16	0.50468	0.46799	0.52504	0.37888	0.39219	0.45055	0.51467	0.49840	0.54763
	64		0.38776	0.44613		0.26650	0.32085		0.43882	0.49225
	256			0.52304			0.32216			0.58755
64K	4	0.63911	0.61236	0.62028	0.55516	0.58454	0.67031	0.63270	0.61493	0.62682
	16	0.41754	0.39062	0.40026	0.30135	0.32526	0.41839	0.43009	0.41571	0.42904
	64		0.31104	0.32678		0.20440	0.28934		0.35946	0.38193
	256			0.37790			0.28635			0.45637
128K	4	0.54950	0.52615	0.54590	0.46170	0.47401	0.54205	0.55376	0.56254	0.58695
	16	0.33875	0.31362	0.33077	0.23372	0.24329	0.29773	0.35182	0.35248	0.37770
	64		0.24168	0.26119		0.15374	0.19433		0.28153	0.31459
	256			0.29154			0.19005			0.35826
256K	4	0.46734	0.45534	0.47929	0.37666	0.38802	0.46728	0.47601	0.50282	0.53812
	16	0.26948	0.25288	0.26955	0.17741	0.18496	0.23923	0.28258	0.29289	0.32287
	64		0.18526	0.19969		0.11383	0.15163		0.21847	0.24837
	256			0.20885			0.14728			0.26100
512K	4	0.38968	0.38990	0.41704	0.29859	0.30708	0.38845	0.40075	0.44423	0.49036
	16	0.21108	0.20242	0.21903	0.13232	0.13759	0.18564	0.22254	0.24027	0.27372
	64		0.14027	0.15251		0.08327	0.11473		0.16661	0.19299
	256			0.15125			0.11023			0.18879

Table 9: Design target bus utilization. This assumes 15 cycle memory response overhead, then 3 cycles for each word transferred.

As seen in section 3.4, using the smallest subsectors minimizes the bus traffic, as measured by the traffic ratio. However, minimizing the data transferred over the bus does not necessarily minimize the number of bus cycles needed, as noted in [Smi87]. For each bus transaction there are a number of overhead (transaction startup) cycles during which the bus is occupied, regardless of the size of data transferred during the transaction.

Our model assumes a non–split–transaction bus using burst mode data transfers. Burst mode transactions (e.g., Futurebus [Can90]) allow a cluster of memory locations (subsectors) to be transferred from main memory by specifying only the address of the first location. Thus the transfer time of a subsector grows sub–linearly with the size of the cluster.

Bus transactions can be modeled as consisting of two phases: address and response. The address phase includes contending for the bus and sending the initial address of the burst transfer to main memory. The duration of the response phase is directly proportional to the size of data being moved, whereas the duration of the initial address phase is the same regardless of the transfer size. The bus is occupied and the processor is stalled until this last phase completes, assuming non–split–transaction bus accesses and a non–blocking cache.

Calculation of bus utilization (Table 9, full size in Table 22 in Appendix A, displayed in Figures 6 and 7 here and Figures 13–16 in Appendix A) requires the introduction of implementation–dependent values to determine the fraction of time the bus is occupied. We model bus utilization using the following assumptions: the bus is busy during the address phase (15 cycles) and the data transfer (one word every 3 cycles). This is equivalent to a 300 MHz processor on a 100 MHz memory bus accessing DRAMs with 5 memory bus cycles to receive the first word, and 1 cycle for each additional word (multiplied by 3 to express it in processor cycle times).

These values will of course vary depending on implementation but are used for purposes of illustration. The numbers from Table 9 (and Table 22) were calculated by dividing the total bus occupancy time by the total workload execution time (including cache miss induced processor stalls). The bus utilization includes the time to write back evicted subsectors to memory. Note that we assume a write–back buffer, so that the processor is not delayed while waiting for the copy–back of a dirty subsector ([TS95] supports this assumption). This assumption may be slightly optimistic for small caches with high miss ratios.

An examination of (Figures 6 and 7) shows that the lowest bus utilizations tend to be obtained with small subsector sizes, with the best subsector size increasing with cache size. Instruction caches perform best with larger subsectors due to the better spatial locality of the instruction stream. As noted in [Prz90a], the best subsector size of the instruction stream

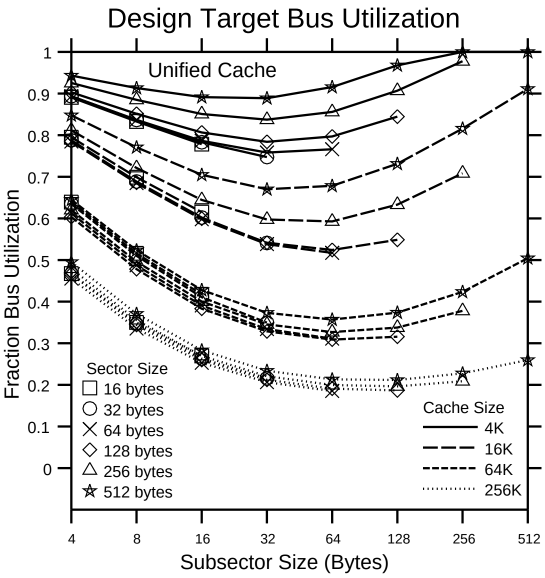


Figure 6: Unified cache design target bus utilization, cache sizes 4K, 16K, 64K, 256K; 15 cycle latency, 3 cycles per word.

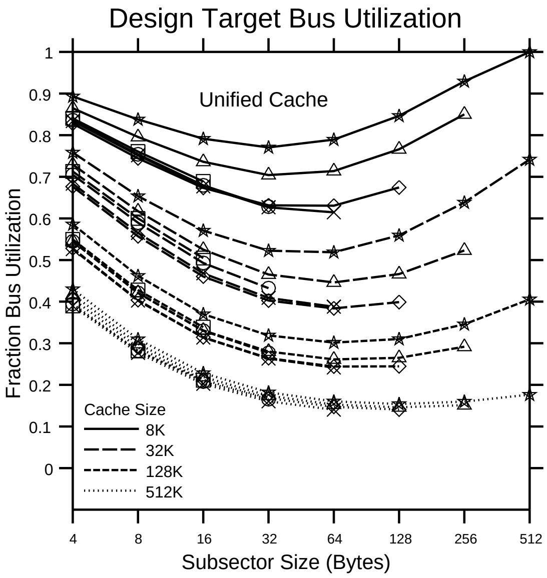


Figure 7: Unified cache design target bus utilization, cache sizes 8K, 32K, 128K, 512K; 15 cycle latency, 3 cycles per word.

is about two times that of the data stream, which is borne out here. The best unified cache fetch size for minimizing bus utilization falls in between that of instruction and data streams.

The bus utilization (Table 9, Figures 6 and 7) hits an extreme with the largest sectors (exceeding 97 percent in the worst case: 4K unified cache). For most reasonable choices of cache, sector and subsector size, the bus utilization ranges from 15 to 75 percent, generally larger for unified and data caches (instruction caches have no write–back traffic). For the best choices among the largest caches, the bus utilization ranges from about 14 to 20 percent. Using a subsector half the size of the sector works well; further reductions in subsector size often lead to an increase in the bus utilization because the fetch size was too small to take advantage of spatial locality of the information stream, causing repeated cache misses and fetches for pieces of the same sector.

Although bus data traffic is always minimized with the smallest subsector size, the best choice of subsector size for minimal bus utilization increases with the cache size due to the larger chance that the data will reside in the cache long enough to be referenced.

4.2 Design Target Delay

The real performance metric for a memory system is the effective memory access time, also called the average memory access delay. The access delay in an n–level cache hierarchy can be modeled as

$$t_{amad} = 1 + \sum_{i=1}^{n} t_i * mr_i(L_i) \tag{8}$$

with $mr_i(L_i)$ the miss ratio (a function of L_i, the transfer size) at cache level i with respect to the processor, i.e., that is the fraction of all memory references satisfied by a hit at that level, and t_i the penalty for fulfilling the miss from level i in the memory hierarchy. This is the generalized version of Equations 2 and 3 for a multilevel hierarchy. This timing assumes that a hit or determining that a miss has occurred in the on-chip cache takes 1 processor cycle. The miss penalty time is a function of the fetch size (subsector size) L_i, the latency a_i (cycles until the first word is ready in memory level $i+1$ for transfer), d_i the data width to the next level of memory, and c_i the additional time to transmit d_i bytes. The miss penalty is stated by:

$$t_i = a_i + c_i * (L_i/d_i). \tag{9}$$

These simple models ignore such features as queueing delays and stalls due to evicted sector write–backs, but these factors are small for uniprocessor systems [TS95]. This delay model assumes, as before, a burst transfer mode that allows a single address transaction followed by a burst transfer of the number of data words requested.

Memory access delay is a particularly good index of performance because it takes into account the miss ratio and the traffic induced by each miss. However, it is highly dependent on the time to transfer a new subsector into the cache. The subsector size that minimizes the delay for a given cache and sector size is dependent on the timing as well. As noted in [Smi87], the optimum delay is a function of the ratio a_i/c_i (but not of either term independently). When this ratio is small, smaller subsectors are preferable because the incremental cost of reading more bytes is large compared to the improved performance that spatial locality provides for larger fetches. When the ratio a_i/c_i is large, the marginal cost of each additional byte fetched is small, making large subsectors the better choice. An important point to note is that the subsector size that minimizes average delay (or bus utilization) is significantly

Design Target Delay (6 Cycles Overhead, 1 Cycle Per Word)										
Sector Size (Bytes)										
Size		**Unified Cache**			**Instruction Cache**			**Data Cache**		
Cache	**SS**	16	64	256	16	64	256	16	64	256
4K	4	3.63741	3.95844	5.21452	3.45206	3.98777	5.02667	3.00719	2.95576	3.13513
	16	2.20000	2.38772	3.03018	2.00000	2.25783	2.71696	2.00000	2.04953	2.24320
	64		2.29800	2.99618		1.94600	2.31977		2.18800	2.62118
	256			4.68068			2.73336			4.70716
16K	4	2.37867	2.40743	2.70071	2.26863	2.44425	2.77131	2.27785	2.33333	2.43635
	16	1.60000	1.62233	1.78314	1.50000	1.57780	1.72406	1.60000	1.64246	1.76895
	64		1.50600	1.71369		1.39600	1.51734		1.57200	1.88212
	256			2.20825			1.61367			2.82990
64K	4	1.66623	1.62780	1.66435	1.53486	1.60299	1.87136	1.62495	1.62182	1.67515
	16	1.28284	1.27480	1.29722	1.21213	1.23707	1.35378	1.28284	1.29657	1.32946
	64		1.21779	1.24643		1.15556	1.24653		1.26446	1.31076
	256			1.33395			1.27994			1.46357
256K	4	1.33032	1.33501	1.37753	1.25897	1.27174	1.37593	1.32951	1.40125	1.47494
	16	1.14142	1.14501	1.16531	1.10607	1.11161	1.15465	1.14142	1.17525	1.21239
	64		1.10889	1.12737		1.07778	1.10822		1.13223	1.16858
	256			1.14715			1.12050			1.19753

Table 10: Design target average memory access delay for unified, instruction, and data caches, sampled over the test space, 6 cycle memory overhead. SS is the subsector size in bytes.

smaller than the subsector size that minimizes the miss ratio. Designs based on minimizing the miss ratio are unlikely to perform very well. For example, 512–byte subsectors have the best miss ratios for all but the smallest caches. However, designs using 512–byte subsectors experience high levels of traffic and memory delay for the cache sizes under examination.

Tables 10 and 11 show a sampling of the various cache designs, using design target delays for 6– and 15–cycle startup latencies, with 1 cycle per word transfer rates (full size versions can be found in Tables 23 and 24 in Appendix A). These values can be computed using the information in Table 6 and Equations 8 and 9. For a given cache size, the best choice to reduce the memory delay is always a non–sector organization. However, there are a few choices that could greatly reduce the number of tag bits to control the cache with only a slight impact on performance. For example, in Table 10 a 64K byte unified cache with 256–byte sectors and 64-byte subsectors performs almost as well as a cache using a 64–byte block (delay 1.24 vs. 1.21), but uses approximately one fourth the number of tag bits. However, compared to the number of bits to implement the entire cache, the reduction is somewhat modest (saving 4.12K bytes or about 6 percent), while decreasing the speed by about 2.5

Design Target Delay (15 Cycles Overhead, 1 Cycle Per Word)										
Sector Size (Bytes)										
Size		**Unified Cache**			**Instruction Cache**			**Data Cache**		
Cache	**SS**	16	64	256	16	64	256	16	64	256
4K	4	7.02836	7.76215	10.63318	6.60470	7.82919	10.20381	5.58787	5.47032	5.88029
	16	3.28000	3.63666	4.85735	2.90000	3.38988	4.26222	2.90000	2.99411	3.36208
	64		2.82900	3.81280		2.33300	2.85968		2.67400	3.28440
	256			5.15391			2.95622			5.18380
16K	4	4.15125	4.21699	4.88734	3.89973	4.30114	5.04871	3.92081	4.04762	4.28309
	16	2.14000	2.18242	2.48797	1.95000	2.09782	2.37571	2.14000	2.22067	2.46101
	64		1.71300	2.00566		1.55800	1.72899		1.80600	2.24299
	256			2.36360			1.69257			3.06517
64K	4	2.52281	2.43497	2.51852	2.22254	2.37826	2.99168	2.42846	2.42129	2.54321
	16	1.53740	1.52213	1.56472	1.40305	1.45044	1.67218	1.53740	1.56349	1.62598
	64		1.30688	1.34724		1.21920	1.34738		1.37265	1.43789
	256			1.37689			1.31594			1.52317
256K	4	1.75502	1.76573	1.86291	1.59192	1.62111	1.85927	1.75316	1.91714	2.08558
	16	1.26870	1.27552	1.31408	1.20153	1.21206	1.29384	1.26870	1.33298	1.40355
	64		1.15344	1.17947		1.10960	1.15250		1.18632	1.23754
	256			1.16607			1.13600			1.22292

Table 11: Design target average memory access delay for unified, instruction, and data caches, sampled over the test space, 15 cycle memory overhead. SS is the subsector size in bytes.

percent.

Determining the best performance for a given number of bits (Tables 25 and 26 in Appendix A), the performance improvement from choosing a sector cache organization for a single level cache is generally on the order of 2 percent or less for reasonably sized caches. We can conclude that sector caches show very little advantage for on–chip caches. Designers were justified in abandoning sector caches for single level caches.

4.3 Two–Level Cache Designs

As shown above, sector caches seldom yield significant improvements for single level caches, but as we show in this section, they can be quite useful in a system with a two (or multi) level cache. In such a multilevel system, the first level is usually fairly small, and the second level can typically be made quite large. It is helpful in a multilevel cache to have the tags for the second level present at the first level, since this can save several cycles in processing a reference that misses at the first level, and then either hits or misses at the

Processor Configurations for Current Microprocessors					
Processor	Frequency	Memory Bus Freq.	DRAM Tech.	Burst Mode	Virtual Address
Alpha 21164	600MHz	100MHz	SDRAM	5-1-1-1	44 Bits
UltraSparc II	336MHz	66MHz	EDO	5-2-2-2	44 Bits
Pentium	266MHz	66MHz	EDO	5-2-2-2	32 Bits
Pentium II	400MHz	100MHz	SDRAM	5-1-1-1	32 Bits
MIPS R10000	250MHz	100MHz	SDRAM	5-1-1-1	44 Bits

Table 12: Processor configurations derived from vendor information.

second level. The useful feature of a sector cache in this case is that the data arrays for a large off-chip cache can be controlled by a relatively small number of on-chip tag bits. For example, the address tag and other overhead bits for a 512K instruction cache using 64–byte blocks requires 44K bytes to implement. By using a sectored organization with 512–byte sectors and 64–byte subsectors, only 6.03K bytes are needed to manage the cache. The savings in the number of bits required to manage a second level cache of a given size can then be used to increase the size of the first level cache; alternately, a larger second level cache can be used than could have been managed with the non-sector design. To evaluate two level cache systems, we will examine how to divide on–chip resources between the first level cache and the tags for the second level cache.

To compare various choices for two–level caches, we assume that the startup latency to access main memory is 24 (processor) cycles, with a bandwidth of one word (four bytes) every four cycles. This assumes that the processor speed is four times the memory bus speed, as it is for current systems such as the Pentium and Pentium II (Table 12). This value falls in between ratio of processor speed to memory bus speeds of the Alpha 21164 (6x) and the MIPS R10000 (2.5x) (Table 12).

A level-one miss causes an access to the level-two cache, requiring a startup time of one cycle, with two words (64 bits) transferred per cycle. This is a 2–1–1–1 burst mode transfer from the level-two cache to the level-one cache on a level-one cache miss (two cycles to receive the first word, one word each cycle after). This setup assumes that the second level cache is interposed between the first level cache and the memory bus (serial organization, as in Figure 8c). One other possible two–level cache organization (parallel cache), uses the memory bus for all L1 and L2 miss transactions, which is slower than the serial organization. The trade–offs between these two organizations can be found in [APB92].

Design Target Two Level Unified Cache Average Memory Access Delay						
On-Chip Bits Used (KBytes)	Tags Only On Chip Normal	Tags Only On Chip Sector	Single Level On-Chip Normal	Single Level On-Chip Sector	Two–level Normal	Two–level Sector
1.34	**4.336** c2 64K 256/256	**4.094** c2 64K 256/128	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.336** c2 64K 256/256	**4.094** c2 64K 256/128
1.69	**4.336** c2 64K 256/256	**3.898** c2 128K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.336** c2 64K 256/256	**3.898** c2 128K 512/64
2.13	**4.336** c2 64K 256/256	**3.898** c2 128K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.336** c2 64K 256/256	**3.898** c2 128K 512/64
2.69	**3.801** c2 256K 512/512	**3.587** c2 256K 512/128	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.801** c2 256K 512/512	**3.587** c2 256K 512/128
3.39	**3.801** c2 256K 512/512	**3.560** c2 256K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.801** c2 256K 512/512	**3.560** c2 256K 512/64
4.27	**3.801** c2 256K 512/512	**3.560** c2 256K 512/64	**8.600** c1 4K 128/128	**6.641** c1 4K 128/32	**3.801** c2 256K 512/512	**3.560** c2 256K 512/64
5.37	**3.491** c2 512K 512/512	**3.396** c2 512K 512/128	**5.592** c1 4K 32/32	**5.592** c1 4K 32/32	**3.236** c1 4K 128/128 c2 64K 256/256	**2.665** c1 4K 64/32 c2 64K 512/64
6.77	**3.491** c2 512K 512/512	**3.393** c2 512K 512/64	**5.592** c1 4K 32/32	**5.592** c1 4K 32/32	**2.391** c1 4K 64/64 c2 256K 512/512	**2.192** c1 4K 64/32 c2 256K 512/256
8.53	**3.491** c2 512K 512/512	**3.393** c2 512K 512/64	**4.800** c1 8K 128/128	**3.968** c1 8K 128/32	**2.281** c1 4K 16/16 c2 256K 512/512	**2.040** c1 4K 16/16 c2 256K 512/64
10.75	**3.396** c2 512K 256/256	**3.366** c2 512K 256/64	**3.800** c1 8K 32/32	**3.800** c1 8K 32/32	**1.971** c1 4K 16/16 c2 512K 512/512	**1.876** c1 4K 16/16 c2 512K 512/128
13.54	**3.396** c2 512K 256/256	**3.366** c2 512K 256/64	**3.800** c1 8K 32/32	**3.800** c1 8K 32/32	**1.821** c1 8K 64/64 c2 512K 512/512	**1.714** c1 8K 128/32 c2 512K 512/128
17.06	**3.396** c2 512K 256/256	**3.366** c2 512K 256/64	**3.432** c1 16K 128/128	**3.124** c1 16K 128/64	**1.791** c1 8K 32/32 c2 512K 512/512	**1.693** c1 8K 32/32 c2 512K 512/64
21.50	**3.342** c2 512K 128/128	**3.342** c2 512K 128/128	**3.016** c1 16K 32/32	**3.016** c1 16K 32/32	**1.696** c1 8K 32/32 c2 512K 256/256	**1.658** c1 16K 128/32 c2 512K 512/256
27.08	**3.342** c2 512K 128/128	**3.342** c2 512K 128/128	**3.016** c1 16K 32/32	**3.016** c1 16K 32/32	**1.626** c1 16K 64/64 c2 512K 256/256	**1.589** c1 16K 64/32 c2 512K 256/128
34.12	**3.342** c2 512K 128/128	**3.342** c2 512K 128/128	**2.368** c1 32K 128/128	**2.221** c1 32K 128/64	**1.612** c1 16K 32/32 c2 512K 256/256	**1.582** c1 16K 32/32 c2 512K 256/64
42.99	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**2.232** c1 32K 64/64	**2.221** c1 32K 128/64	**1.558** c1 32K 128/128 c2 512K 256/256	**1.500** c1 32K 128/32 c2 512K 256/128
54.17	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**2.232** c1 32K 64/64	**2.221** c1 32K 128/64	**1.482** c1 32K 64/64 c2 512K 128/128	**1.475** c1 32K 128/32 c2 512K 128/128
68.25	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.967** c1 64K 128/128	**1.887** c1 64K 128/64	**1.482** c1 32K 64/64 c2 512K 128/128	**1.475** c1 32K 128/32 c2 512K 128/128
85.98	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.871** c1 64K 64/64	**1.871** c1 64K 64/64	**1.448** c1 32K 64/64 c2 512K 64/64	**1.440** c1 64K 128/32 c2 512K 128/128
108.33	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.871** c1 64K 64/64	**1.871** c1 64K 64/64	**1.407** c1 64K 64/64 c2 512K 64/64	**1.406** c1 64K 64/32 c2 512K 64/64
136.49	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.684** c1 128K 128/128	**1.645** c1 128K 128/64	**1.407** c1 64K 64/64 c2 512K 64/64	**1.406** c1 64K 64/32 c2 512K 64/64
171.97	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.616** c1 128K 64/64	**1.616** c1 128K 64/64	**1.407** c1 64K 64/64 c2 512K 64/64	**1.380** c1 128K 128/32 c2 512K 64/64
216.67	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.616** c1 128K 64/64	**1.616** c1 128K 64/64	**1.378** c1 128K 64/64 c2 512K 64/64	**1.378** c1 128K 64/32 c2 512K 64/64
272.98	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.484** c1 256K 128/128	**1.471** c1 256K 128/64	**1.378** c1 128K 64/64 c2 512K 64/64	**1.378** c1 128K 64/32 c2 512K 64/64
343.94	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.436** c1 256K 64/64	**1.436** c1 256K 64/64	**1.357** c1 256K 64/64 c2 512K 64/64	**1.357** c1 256K 64/64 c2 512K 64/64
433.33	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.436** c1 256K 64/64	**1.436** c1 256K 64/64	**1.357** c1 256K 64/64 c2 512K 64/64	**1.357** c1 256K 64/64 c2 512K 64/64
545.97	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.342** c1 512K 128/128	**1.342** c1 512K 128/128	**1.357** c1 256K 64/64 c2 512K 64/64	**1.357** c1 256K 64/64 c2 512K 64/64
687.88	**3.308** c2 512K 64/64	**3.308** c2 512K 64/64	**1.308** c1 512K 64/64	**1.308** c1 512K 64/64	**1.343** c1 512K 64/64 c2 512K 64/64	**1.343** c1 512K 64/64 c2 512K 64/64

Table 13: Comparison of the best performing caches for various unified cache organizations. Each entry shows the following information: memory delay, first (c1) and/or second (c2) level cache size, and sector/subsector size, in bytes.

Design Target Two Level Instruction Cache Average Memory Access Delay						
On-Chip Bits Used (KBytes)	Tags Only On Chip Normal	Tags Only On Chip Sector	Single Level On-Chip Normal	Single Level On-Chip Sector	Two–level Normal	Two–level Sector
1.34	**3.873** c2 128K 512/512	**3.669** c2 128K 512/128	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.873** c2 128K 512/512	**3.669** c2 128K 512/128
1.69	**3.873** c2 128K 512/512	**3.654** c2 128K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.873** c2 128K 512/512	**3.654** c2 128K 512/64
2.13	**3.873** c2 128K 512/512	**3.654** c2 128K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.873** c2 128K 512/512	**3.654** c2 128K 512/64
2.69	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64
3.39	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64
4.27	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64	**5.864** c1 4K 128/128	**5.473** c1 4K 128/64	**3.641** c2 256K 512/512	**3.492** c2 256K 512/64
5.37	**3.482** c2 256K 256/256	**3.384** c2 512K 512/64	**4.528** c1 4K 32/32	**4.528** c1 4K 32/32	**2.449** c1 4K 128/128 c2 128K 512/512	**2.248** c1 4K 128/64 c2 128K 512/256
6.77	**3.482** c2 256K 256/256	**3.384** c2 512K 512/64	**4.528** c1 4K 32/32	**4.528** c1 4K 32/32	**2.071** c1 4K 64/64 c2 256K 512/512	**1.930** c1 4K 64/64 c2 256K 512/128
8.53	**3.482** c2 256K 256/256	**3.384** c2 512K 512/64	**3.432** c1 8K 128/128	**3.288** c1 8K 128/64	**2.019** c1 4K 32/32 c2 256K 512/512	**1.870** c1 4K 32/32 c2 256K 512/64
10.75	**3.346** c2 512K 256/256	**3.314** c2 512K 256/64	**3.024** c1 8K 64/64	**3.024** c1 8K 64/64	**1.860** c1 4K 32/32 c2 256K 256/256	**1.760** c1 8K 128/64 c2 256K 512/128
13.54	**3.346** c2 512K 256/256	**3.314** c2 512K 256/64	**3.024** c1 8K 64/64	**3.024** c1 8K 64/64	**1.711** c1 8K 32/32 c2 512K 512/512	**1.618** c1 8K 64/64 c2 512K 512/128
17.06	**3.346** c2 512K 256/256	**3.314** c2 512K 256/64	**2.824** c1 16K 128/128	**2.776** c1 16K 128/64	**1.634** c1 8K 128/128 c2 512K 256/256	**1.606** c1 8K 128/64 c2 512K 256/256
21.50	**3.266** c2 512K 128/128	**3.263** c2 512K 128/64	**2.584** c1 16K 64/64	**2.584** c1 16K 64/64	**1.568** c1 8K 32/32 c2 512K 256/256	**1.536** c1 8K 32/32 c2 512K 256/64
27.08	**3.266** c2 512K 128/128	**3.263** c2 512K 128/64	**2.584** c1 16K 64/64	**2.584** c1 16K 64/64	**1.488** c1 8K 32/32 c2 512K 128/128	**1.485** c1 8K 32/32 c2 512K 128/64
34.12	**3.266** c2 512K 128/128	**3.263** c2 512K 128/64	**2.064** c1 32K 128/128	**2.040** c1 32K 128/64	**1.482** c1 16K 128/128 c2 512K 128/128	**1.468** c1 16K 128/64 c2 512K 128/128
42.99	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.880** c1 32K 64/64	**1.880** c1 32K 64/64	**1.440** c1 16K 32/32 c2 512K 128/128	**1.432** c1 32K 128/64 c2 512K 256/64
54.17	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.880** c1 32K 64/64	**1.880** c1 32K 64/64	**1.366** c1 32K 64/64 c2 512K 128/128	**1.363** c1 32K 64/64 c2 512K 128/64
68.25	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.752** c1 64K 128/128	**1.752** c1 64K 128/128	**1.366** c1 32K 64/64 c2 512K 128/128	**1.338** c1 32K 128/64 c2 512K 64/64
85.98	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.622** c1 64K 64/64	**1.622** c1 64K 64/64	**1.320** c1 32K 64/64 c2 512K 64/64	**1.320** c1 32K 64/64 c2 512K 64/64
108.33	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.622** c1 64K 64/64	**1.622** c1 64K 64/64	**1.291** c1 64K 64/64 c2 512K 64/64	**1.291** c1 64K 64/64 c2 512K 64/64
136.49	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.532** c1 128K 128/128	**1.523** c1 128K 128/64	**1.291** c1 64K 64/64 c2 512K 64/64	**1.291** c1 64K 64/64 c2 512K 64/64
171.97	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.440** c1 128K 64/64	**1.440** c1 128K 64/64	**1.270** c1 128K 64/64 c2 512K 64/64	**1.270** c1 128K 64/64 c2 512K 64/64
216.67	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.440** c1 128K 64/64	**1.440** c1 128K 64/64	**1.270** c1 128K 64/64 c2 512K 64/64	**1.270** c1 128K 64/64 c2 512K 64/64
272.98	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.376** c1 256K 128/128	**1.372** c1 256K 128/64	**1.270** c1 128K 64/64 c2 512K 64/64	**1.270** c1 128K 64/64 c2 512K 64/64
343.94	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.311** c1 256K 64/64	**1.311** c1 256K 64/64	**1.255** c1 256K 64/64 c2 512K 64/64	**1.255** c1 256K 64/64 c2 512K 64/64
433.33	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.311** c1 256K 64/64	**1.311** c1 256K 64/64	**1.255** c1 256K 64/64 c2 512K 64/64	**1.255** c1 256K 64/64 c2 512K 64/64
545.97	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.266** c1 512K 128/128	**1.263** c1 512K 128/64	**1.255** c1 256K 64/64 c2 512K 64/64	**1.255** c1 256K 64/64 c2 512K 64/64
687.88	**3.220** c2 512K 64/64	**3.220** c2 512K 64/64	**1.220** c1 512K 64/64	**1.220** c1 512K 64/64	**1.245** c1 512K 64/64 c2 512K 64/64	**1.245** c1 512K 64/64 c2 512K 64/64

Table 14: Comparison of the best performing caches for various instruction cache organizations. Each entry shows the following information: memory delay, first (c1) and/or second (c2) level cache size, and sector/subsector size, in bytes.

Design Target Two Level Data Cache Average Memory Access Delay						
On-Chip Bits Used (KBytes)	Tags Only On Chip Normal	Tags Only On Chip Sector	Single Level On-Chip Normal	Single Level On-Chip Sector	Two–level Normal	Two–level Sector
1.34	**4.824** c2 32K 128/128	**4.308** c2 64K 512/32	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.824** c2 32K 128/128	**4.308** c2 64K 512/32
1.69	**4.824** c2 32K 128/128	**4.065** c2 128K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.824** c2 32K 128/128	**4.065** c2 128K 512/64
2.13	**4.824** c2 32K 128/128	**4.038** c2 128K 512/32	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.824** c2 32K 128/128	**4.038** c2 128K 512/32
2.69	**4.092** c2 256K 512/512	**3.777** c2 256K 512/128	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.092** c2 256K 512/512	**3.777** c2 256K 512/128
3.39	**4.092** c2 256K 512/512	**3.732** c2 256K 512/64	**28.000** c1 0K 0/0	**28.000** c1 0K 0/0	**4.092** c2 256K 512/512	**3.732** c2 256K 512/64
4.27	**4.092** c2 256K 512/512	**3.732** c2 256K 512/64	**8.296** c1 4K 128/128	**5.122** c1 4K 128/16	**4.092** c2 256K 512/512	**3.732** c2 256K 512/64
5.37	**3.652** c2 512K 512/512	**3.526** c2 512K 512/128	**4.920** c1 4K 32/32	**4.920** c1 4K 32/32	**3.652** c2 512K 512/512	**2.720** c1 4K 128/16
6.77	**3.652** c2 512K 512/512	**3.526** c2 512K 512/64	**4.920** c1 4K 32/32	**4.920** c1 4K 32/32	**2.632** c1 4K 64/64 c2 256K 512/512	**2.307** c1 4K 128/16 c2 256K 512/256
8.53	**3.652** c2 512K 512/512	**3.526** c2 512K 512/64	**4.920** c1 4K 32/32	**4.045** c1 8K 128/32	**2.492** c1 4K 16/16 c2 256K 512/512	**2.132** c1 4K 16/16 c2 256K 512/64
10.75	**3.521** c2 512K 256/256	**3.485** c2 512K 256/128	**3.968** c1 8K 32/32	**3.968** c1 8K 32/32	**2.052** c1 4K 16/16 c2 512K 512/512	**1.926** c1 4K 16/16 c2 512K 512/128
13.54	**3.521** c2 512K 256/256	**3.485** c2 512K 256/128	**3.968** c1 8K 32/32	**3.968** c1 8K 32/32	**1.941** c1 4K 32/32 c2 512K 256/256	**1.852** c1 8K 128/32 c2 512K 512/128
17.06	**3.521** c2 512K 256/256	**3.485** c2 512K 256/128	**3.888** c1 16K 128/128	**3.255** c1 16K 128/32	**1.921** c1 4K 16/16 c2 512K 256/256	**1.836** c1 8K 128/16 c2 512K 512/64
21.50	**3.456** c2 512K 128/128	**3.456** c2 512K 128/128	**3.184** c1 16K 32/32	**3.184** c1 16K 32/32	**1.839** c1 8K 32/32 c2 512K 256/256	**1.795** c1 8K 128/16 c2 512K 256/128
27.08	**3.456** c2 512K 128/128	**3.456** c2 512K 128/128	**3.184** c1 16K 32/32	**3.184** c1 16K 32/32	**1.774** c1 8K 32/32 c2 512K 128/128	**1.725** c1 16K 64/32 c2 512K 256/128
34.12	**3.456** c2 512K 128/128	**3.456** c2 512K 128/128	**2.824** c1 32K 128/128	**2.506** c1 32K 128/32	**1.755** c1 16K 32/32 c2 512K 256/256	**1.719** c1 16K 32/32 c2 512K 256/128
42.99	**3.374** c2 512K 64/64	**3.374** c2 512K 64/64	**2.400** c1 32K 32/32	**2.400** c1 32K 32/32	**1.690** c1 16K 32/32 c2 512K 128/128	**1.646** c1 32K 128/32 c2 512K 256/128
54.17	**3.374** c2 512K 64/64	**3.374** c2 512K 64/64	**2.400** c1 32K 32/32	**2.400** c1 32K 32/32	**1.608** c1 16K 32/32 c2 512K 64/64	**1.608** c1 16K 32/32 c2 512K 64/64
68.25	**3.374** c2 512K 64/64	**3.374** c2 512K 64/64	**2.290** c1 64K 128/128	**2.114** c1 64K 128/32	**1.606** c1 32K 32/32 c2 512K 128/128	**1.574** c1 32K 256/32 c2 512K 64/64
85.98	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.990** c1 64K 32/32	**1.990** c1 64K 32/32	**1.524** c1 32K 32/32 c2 512K 64/64	**1.524** c1 32K 32/32 c2 512K 64/64
108.33	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.990** c1 64K 32/32	**1.990** c1 64K 32/32	**1.494** c1 64K 64/64 c2 512K 64/64	**1.486** c1 64K 64/32 c2 512K 64/64
136.49	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.912** c1 128K 128/128	**1.847** c1 128K 128/64	**1.480** c1 64K 32/32 c2 512K 64/64	**1.477** c1 64K 256/32 c2 512K 32/32
171.97	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.700** c1 128K 32/32	**1.700** c1 128K 32/32	**1.456** c1 64K 32/32 c2 512K 32/32	**1.456** c1 64K 32/32 c2 512K 32/32
216.67	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.700** c1 128K 32/32	**1.700** c1 128K 32/32	**1.449** c1 128K 32/32 c2 512K 64/64	**1.433** c1 128K 64/32 c2 512K 32/32
272.98	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.645** c1 256K 128/128	**1.628** c1 256K 128/64	**1.425** c1 128K 32/32 c2 512K 32/32	**1.425** c1 128K 32/32 c2 512K 32/32
343.94	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.495** c1 256K 32/32	**1.495** c1 256K 32/32	**1.425** c1 128K 32/32 c2 512K 32/32	**1.422** c1 256K 128/32 c2 512K 32/32
433.33	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.495** c1 256K 32/32	**1.495** c1 256K 32/32	**1.403** c1 256K 32/32 c2 512K 32/32	**1.403** c1 256K 32/32 c2 512K 32/32
545.97	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.456** c1 512K 128/128	**1.456** c1 512K 128/128	**1.403** c1 256K 32/32 c2 512K 32/32	**1.403** c1 256K 32/32 c2 512K 32/32
687.88	**3.350** c2 512K 32/32	**3.350** c2 512K 32/32	**1.350** c1 512K 32/32	**1.350** c1 512K 32/32	**1.388** c1 512K 32/32 c2 512K 32/32	**1.388** c1 512K 32/32 c2 512K 32/32

Table 15: Comparison of the best performing caches for various data cache organizations. Each entry shows the following information: memory delay, first (c1) and/or second (c2) level cache size, and sector/subsector size, in bytes.

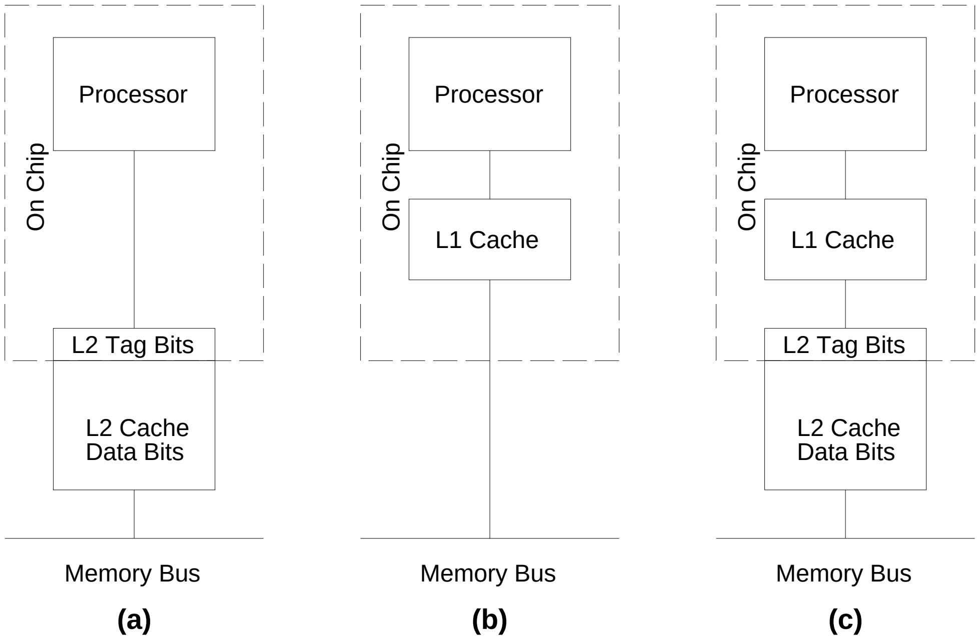


Figure 8: Diagrams of three two–level cache organizations: (a) off–chip only cache, (b) on–chip only cache, and (c) combination on– and off–chip caches.

Since there are literally thousands of possible combinations of first and second level caches over the range of cache sizes and designs that we have considered in this paper, we have selected a few representative cases. In each case, we present the best organizations for a given on–chip bit budget using the timing mentioned above in combination with design target miss ratios. For each type of cache (unified, instruction, data), we examined six different cache organizations under the same timing constraints. The different organizations consist of normal (non-sector) and sector versions of an off-chip cache (with the tags on-chip) (Figure 8a), a single level on-chip cache (Figure 8b), and a two–level cache with the level one cache and tags for the level two cache on-chip (Figure 8c). For a given number of bits, the best organization is in Tables 13–15. These tables assume a 48–bit virtual address and eight–way set–associativity for both levels of cache. For a given total on–chip cache bit budget, the best choices under the various constraints (on– or off–chip cache, sector vs. normal cache) are shown. The sector cache choices for each organization will be as good as or better than the normal cache, since the normal cache organizations are a proper subset of the sector cache choices. The two–level caches will be as good as or better than the single level off-chip cache as well as the single level on–chip cache, since those organizations are considered proper subsets of the two–level caches.

Given a choice of a small on-chip cache or a large off-chip cache, the off–chip cache is a winner. For example, given only 4.27K bytes of space to implement a data cache, the off-chip sector cache solution which controls a 256K byte cache with 512–byte sectors and 64–byte subsectors has less than one half of the delay of a 4K on-chip cache with 128–byte blocks.

Off-chip organizations (some with very small on-chip caches) were used by the early members of the PA-RISC 7000 series, although since they were able to run the off–chip cache at the same rate as the processor, cache access only required a single cycle [KCZD94]. Newer generations of processors are unlikely to be able to run off-chip caches at the same rate as the processor, due to the access time of large caches (which increases with cache size due to capacitive loading) and the delay caused by the off–chip interconnection wires, which have much higher inductive, capacitive, and resistive values than the much smaller on–chip interconnects.

From the tables, we can see that the sector cache organizations are the best choices when the number of on-chip bits is small, due to more flexibility in the organization. Tables 13–15 demonstrate how powerful the sector organization can be. Particularly for the data cache (Table 15), many of the best choices for a two level cache memory system utilize sectors for the first or second levels. The reduction in delay by using a sector cache organization is very significant for gross cache sizes of 20K bytes of space or less.

It can also be seen that for large numbers of on-chip cache bits, sector caches are sometimes useful, but even when useful, yield only small performance improvements. Note that although chip sizes are rapidly increasing, two level sector caches are still very useful for future designs. Optimal design is not making the biggest chip, but the best chip, trading off cost and performance. Making a smaller chip is much cheaper. Alternately, putting multiple CPUs on a large chip (thus reducing the area available for caches) may be a preferred approach. Thus the space available for on-chip cache storage will likely continue to be very limited, especially for embedded processors, whose designs are extremely cost sensitive.

4.3.1 Sample Organizations

Table 16 shows a number of real cache designs, and contrasts the level of performance with that possible from a sectored design (assuming the timings and design target miss ratios used earlier). In most cases, better performance, sometimes quite significantly so, can be had

Performance of Some Common Systems

Processor	Cache Type	Cache 1 Size	Sector 1 Size	Subsector 1 Size	Cache 2 Size	Sector 2 Size	Subsector 2 Size	On–Chip Bytes	Delay Time
Alpha	Data	8KB	32B	32B	128KB†	32B	32B	27.9KB	2.02
[Ben95]	Improved	16KB	32B	32B	512KB	256B	128B	27.8KB	**1.72**
UltraSparc	Data	16KB	32B	16B	0KB	0B	0B	18.6KB	3.55
[TO96]	Improved	8KB	128B	16B	512KB	256B	128B	17.7KB	**1.79**
Pentium	Inst	8KB	32B	32B	256KB	32B	32B	44.5KB	1.56
[Sai93]	Improved	32KB	64B	64B	512KB	256B	64B	43.7KB	**1.41**
PentiumII	Data	16KB	32B	32B	512KB	32B	32B	89.0KB	1.58
[CS95]	Improved	32KB	32B	32B	512KB	64B	64B	72.2KB	**1.52**
R10000	Inst	32KB	64B	64B	512KB	128B	128B	51.5KB	1.37
[Yea96]	Improved	32KB	64B	64B	512KB	128B	128B	51.5KB	**1.37**
PowerPC	Unified	32KB	64B	32B	0KB	0B	0B	34.6KB	2.26
[Moo93]	Improved	16KB	128B	32B	512KB	128B	128B	34.3KB	**1.57**

Table 16: Original and improved choices for real cache systems. († Alpha's second level cache is actually 96K bytes, but we use 128K bytes for comparison.)

with fewer or similar amounts of on–chip resources. For single level caches, it is worthwhile to divide the cache size in half in order to provide tags to control a large second level cache. Two–level caches can also be reorganized to control a larger cache with fewer tags. It should be noted that some of the processors already use sector caches. Some examples of this are the IBM 601 [Moo93] and the UltraSPARC [TO96] (shown in Table 16). Sector caches can be used to enhance performance in real systems, often at a smaller overall cost, measured in delay time and on–chip space.

5 Conclusions

In this paper, we have provided a thorough analysis of the design trade–offs for sector caches and have determined the circumstances under which they are better than normal, non-sectored caches. Using miss ratios and other statistics from the raw trace driven simulation of our multiprogrammed workload, design targets were developed. These design targets included miss ratios, traffic ratios, bus utilization and delay (average memory access time).

Using the number of bits in a cache and delay, we examined the best performing caches for a given number of bits. We found that for single level caches, sector caches are seldom advantageous. For multilevel cache designs with small amounts of storage at the first level caches, as would be the case for small on-chip caches, sector caches can yield significant performance improvements. For multilevel designs with large amounts of first level storage, sector caches provide relatively small improvements.

This work can be extended in several ways. One problem with sector caches is that an estimated 72 percent [HS84] of the subsectors are not referenced while a sector is present in the cache; our data shows the amount of unused subsectors ranges from 6 percent to over 90 percent. Research is required to more effectively utilize the data space in sector caches, such as along the lines of [Sez94, Sez97], which shows a method of dis–associating subsectors from sectors to reduce tag space without adversely impacting performance. More demanding workloads should be found to analyze caches and push them harder, as common workloads such as SPEC have been found lacking in utilizing larger caches [GHPS93]. Other work, in progress, is to examine the utility of sector caches for shared memory multiprocessors.

References

[ACY+83] Don Alpert, Dean Carberry, Mike Yamamura, Ying Chow and Phil Mak. 32–bit Processor Chip Integrates Major System Functions. *Electronics*, pp. 113-119, July 14 1983.

[APB92] Mani Azimi, Bindi Prasad, and Ketan Bhat. Two Level Cache Architectures. In *Thirty–Seventh IEEE Computer Society International Conference*, pp. 344–349, San Francisco, CA, February 24–28 1992.

[Ben95] B. Benschneider. An overview of the Alpha AXP 21164 microprocessor. *Proc. 38th Midwest Symposium on Circuits and Systems*, vol. 2, pp. 1131–34, Rio de Janeiro, Brazil, August 13–16 1995.

[BW88] Jean-Loup Baer and Wen-Hann Wang. On The Inclusion Properties For Multi–Level Cache Hierarchies. *Proc. 15th Annual International Symposium on Computer Architecture*, pp. 73–80, Honolulu, HI, May 30–June 2 1988.

[Can90] Jay Cantrell. Futurebus+ Cache Coherence. *WESCON/90 Conference Record*, pp. 90–94, Anaheim, CA, November 13–15 1990.

[CS95] Robert P. Colwell and Randy L. Steck. A 0.6μm BiCMOS Processor with Dynamic Execution. *IEEE International Solid State Circuits Conference*, pp. 176–177,361, San Francisco, CA February 15–17 1995.

[GHPS93] Jeffrey D. Gee, Mark D. Hill, Dionisios N. Pnevmatikatos, and Alan Jay Smith. Cache Performance of the SPEC92 Benchmark Suite. *IEEE Micro*, 13(4):17–27, August 1993.

[Goo83] James R. Goodman. Using Cache Memory to Reduce Processor-Memory Traffic. *Proc. 10th Annual International Symposium on Computer Architecture*, pp. 124–131, Stockholm, Sweden, June 13–16 1983.

[Goo87] James R. Goodman. Cache Memory Optimization to Reduce Processor/Memory Traffic. *Journal of VLSI and Computer Systems*, 2(1&2):61–86, 1987.

[HS84] Mark D. Hill and Alan Jay Smith. Experimental Evaluation of On-Chip Microprocessor Cache Memories. *Proc. 11th Annual International Symposium on Computer Architecture*, pp. 158–166, Ann Arbor, MI, June 5–7 1984.

[HS89] Mark D. Hill and Alan Jay Smith. Evaluating Associativity in CPU Caches. *IEEE Transactions on Computers*, 38(12):1612–1630, December 1989.

[Jow98] Lily Jow, Tandem Computers. *Private communication*, 1998.

[KCZD94] Gordon Kurpanek, Ken Chan, Jason Zheng, Eric DeLano, and William Bryg. PA7200: A PA–RISC Processor with Integrated High Performance MP Bus Interface. *Thirty–Ninth IEEE Computer Society International Conference*, pp. 375–382, San Francisco, CA, February 28–March 4 1994.

[Kob86] Makoto Kobayashi. An Empirical Study of Task Switching Locality in MVS. *IEEE Trans. on Computers*, C–35(8):720–731, August 1986.

[Lip68] J. S. Liptay. Structural Aspects of the System/360 Model 85, Part II: The Cache. *IBM Systems Journal*, vol. 7, pp. 15–21, 1968.

[Moo93] Charles R. Moore. The PowerPC 601 Microprocessor. *IEEE International Computer Society Conference*, pp. 109–116, San Francisco, CA, February 22-26 1993.

[Prz90a] Steven Przybylski. The Performance Impact of Block Sizes and Fetch Strategies. *Proc. 17th Annual International Symposium on Computer Architecture*, pp 160–169, Seattle, WA, May 28–31 1990.

[Prz90b] Steven A. Przybylski. *Cache and Memory Hierarchy Design: A Performance–Directed Approach.* San Mateo, CA. Morgan Kaufmann Publishers, 1990.

[Rot] Jeffrey B. Rothman. Multiprocessor Memory Reference Generation Using **Cerberus**. Technical Report, Computer Science Division, U.C. Berkeley, Berkeley, CA, in preparation.

[Sai93] Avtar Saini. Design of the Intel Pentium Processor. *Proc. IEEE International Conference on Computer Design*, pp. 258–261, Cambridge, MA, October 3–6 1993.

[Sez94] André Seznec. Decoupled Sectored Caches: conciliating low tag implementation cost and low miss ratio. *Proc. 21st Annual International Symposium on Computer Architecture*, pp. 384–393, Chicago, IL, April 18–21 1994.

[Sez97] André Seznec. Decoupled sectored caches. *IEEE Trans. on Computers*, 46(2):210–215, February 1997.

[Smi78] Alan Jay Smith. A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory. *IEEE Trans. on Software Engineering*, SE–4(2):121–130, March 1978.

[Smi82] Alan Jay Smith. Cache Memories. *ACM Computing Surveys*, 14(3):473–530, September 1982.

[Smi85] Alan Jay Smith. Cache Evaluation and the Impact of Workload Choice. *Proc. 12th Annual International Symposium on Computer Architecture*, pp. 64–73, Boston, MA, June, 1985.

[Smi87] Alan Jay Smith. Line (Block) Size Choice for CPU Cache Memories. *IEEE Trans. on Computers*, C-36(9):1063–1075, September 1987.

[Smi94] Alan Jay Smith. Trace Driven Simulation in Research on Computer Architecture and Operating Systems (invited paper). Proc. New Directions in Simulation for Manufacturing and Communications (SIM94), Tokyo, Japan, August 1-2, 1994, ed. Morito, Sakasegawa, Yoneda, Fushimi, Nakano, pp. 43-49.

[Smi91] Michael D. Smith, "Tracing with Pixie," Technical Report CSL-TR-91-497, Stanford University, Stanford CA, November 1991.

[SWG92] Jaswinder Pal Singh, Wolf–Dietrich Weber, and Anoop Gupta. SPLASH: Stanford Parallel Applications for Shared–Memory. *Computer Architecture News*, 20(1):5-44, March 1992.

[TO96] Marc Tremblay and J. Michael O'Connor. UltraSparc 1: A Four–Issue Processor Supporting Multimedia. *IEEE Micro*, 16(2):42–50, April 1996.

[TS89] James G. Thompson and Alan Jay Smith. Efficient (Stack) Algorithms for Analysis of Write–Back and Sector Memories. *ACM Transactions on Computer Systems*, 7(1):78–116, February 1989.

[TS95] John Tse and Alan Jay Smith. CPU Cache Prefetching: Timing Evaluation of Hardware Implementations. *IEEE Trans. on Computers*, 47(5):509–526, May 1998.

[Yea96] Kenneth C. Yeager. The MIPS R10000 Superscalar Microprocessor. *IEEE Micro*, 16(2):28–41, April 1996.

A Supplementary Figures and Tables

Figures 9, 10, 11 and 12 show the graphical representation of the miss ratios generated by our simulations from Table 4. These are companions to Figures 2 and 3 in the main text.

Tables 17 and 18 show the number of tag bits that each type of cache requires to control the various cache sizes and configurations, generated by Equation 4. Included in the number of tag bits are the state bits which determine whether a subsector is valid, clean, or dirty and 10 bits per set to implement a pseudo–LRU replacement policy for 8–way set–associative caches (as described in [Smi82]). Instruction caches need only 1 state bit per subsector, since subsectors can only be invalid or clean. Unified and data caches can have dirty subsectors, which indicates that the information is inconsistent with main memory and must be written back when the sector is evicted from the cache.

A number of tables in this appendix are full sized versions of data we generated, of which sampled versions were presented in the main body of the paper. These tables include subsector utilization (sampled: Table 5, full: Table 19), fraction dirtiness of evicted sectors (sampled: Table 7, full: Table 20), design target traffic ratios (sampled: Table 8, full: Table 21), and bus utilization (sampled: Table 9, full: Table 22). The companion figures for Table 22 are Figures 6 and 7 in Section 4.1 for unified caches and Figures 13–15 in this appendix. The bus utilization was calculated assuming a 15 cycle access overhead plus 3 cycles for each word transferred. DTMRs were used in combination with write–back information from the simulations to derive the bus utilization.

The design target delay for single level caches can be directly calculation from Table 6, so it was placed in this appendix in Tables 23 and 24, corresponding to 6 and 15 cycle access startup time, with one word per cycle after that. Tables 25 and 26 show the best memory delay for a give bit budget for level one caches. Only in rare cases are sector cache organizations better than normal caches, and the improvement is small, generally 2 percent or less.

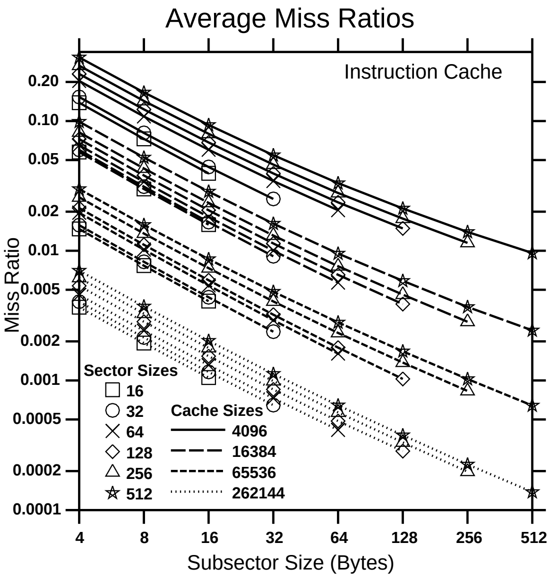


Figure 9: Instruction cache miss ratios, cache sizes 4K, 16K, 64K, 256K.

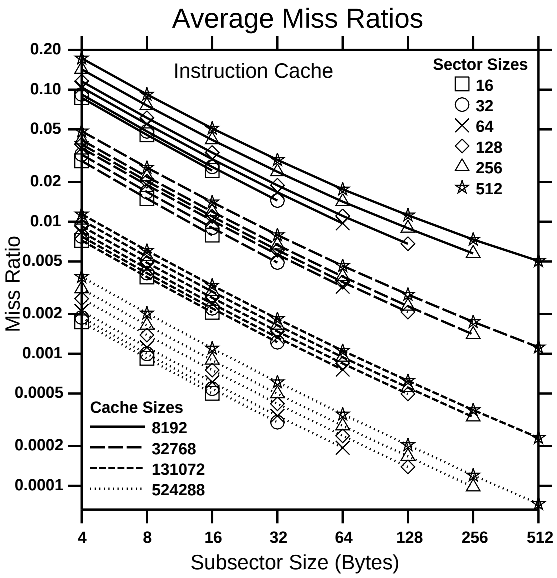


Figure 10: Instruction cache miss ratios, cache sizes 8K, 32K, 128K, 512K.

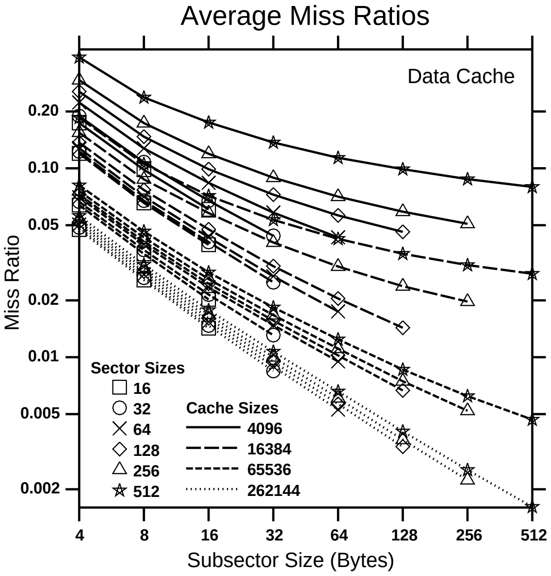


Figure 11: Data cache miss ratios, cache sizes 4K, 16K, 64K, 256K.

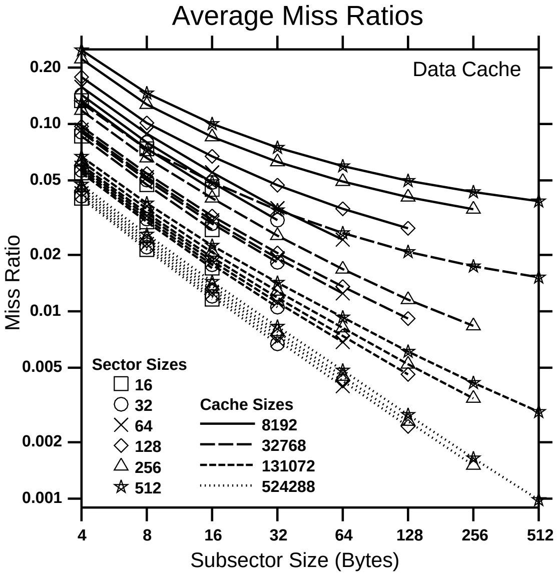


Figure 12: Data cache miss ratios, cache sizes 8K, 32K, 128K, 512K.

Tag Bits Required to Manage Instruction Cache							
Subsector	Cache	Sector Size (bytes)					
Size	Size	16	32	64	128	256	512
4 Bytes	4096	12608	6688	3792	2376	1684	1346
	8192	25216	13376	7584	4752	3368	2692
	16384	50432	26752	15168	9504	6736	5384
	32768	100864	53504	30336	19008	13472	10768
	65536	201728	107008	60672	38016	26944	21536
	131072	403456	214016	121344	76032	53888	43072
	262144	806912	428032	242688	152064	107776	86144
	524288	1613824	856064	485376	304128	215552	172288
8 Bytes	4096	12096	6176	3280	1864	1172	834
	8192	24192	12352	6560	3728	2344	1668
	16384	48384	24704	13120	7456	4688	3336
	32768	96768	49408	26240	14912	9376	6672
	65536	193536	98816	52480	29824	18752	13344
	131072	387072	197632	104960	59648	37504	26688
	262144	774144	395264	209920	119296	75008	53376
	524288	1548288	790528	419840	238592	150016	106752
16 Bytes	4096	11840	5920	3024	1608	916	578
	8192	23680	11840	6048	3216	1832	1156
	16384	47360	23680	12096	6432	3664	2312
	32768	94720	47360	24192	12864	7328	4624
	65536	189440	94720	48384	25728	14656	9248
	131072	378880	189440	96768	51456	29312	18496
	262144	757760	378880	193536	102912	58624	36992
	524288	1515520	757760	387072	205824	117248	73984
32 Bytes	4096		5792	2896	1480	788	450
	8192		11584	5792	2960	1576	900
	16384		23168	11584	5920	3152	1800
	32768		46336	23168	11840	6304	3600
	65536		92672	46336	23680	12608	7200
	131072		185344	92672	47360	25216	14400
	262144		370688	185344	94720	50432	28800
	524288		741376	370688	189440	100864	57600
64 Bytes	4096			2832	1416	724	386
	8192			5664	2832	1448	772
	16384			11328	5664	2896	1544
	32768			22656	11328	5792	3088
	65536			45312	22656	11584	6176
	131072			90624	45312	23168	12352
	262144			181248	90624	46336	24704
	524288			362496	181248	92672	49408
128 Bytes	4096				1384	692	354
	8192				2768	1384	708
	16384				5536	2768	1416
	32768				11072	5536	2832
	65536				22144	11072	5664
	131072				44288	22144	11328
	262144				88576	44288	22656
	524288				177152	88576	45312
256 Bytes	4096					676	338
	8192					1352	676
	16384					2704	1352
	32768					5408	2704
	65536					10816	5408
	131072					21632	10816
	262144					43264	21632
	524288					86528	43264
512 Bytes	4096						330
	8192						660
	16384						1320
	32768						2640
	65536						5280
	131072						10560
	262144						21120
	524288						42240

Table 17: Number of address and status tag bits needed to implement an instruction cache.

Tag Bits Required to Manage Unified or Data Cache							
Subsector	Cache	Sector Size (bytes)					
Size	Size	16	32	64	128	256	512
4 Bytes	4096	13632	7712	4816	3400	2708	2370
	8192	27264	15424	9632	6800	5416	4740
	16384	54528	30848	19264	13600	10832	9480
	32768	109056	61696	38528	27200	21664	18960
	65536	218112	123392	77056	54400	43328	37920
	131072	436224	246784	154112	108800	86656	75840
	262144	872448	493568	308224	217600	173312	151680
	524288	1744896	987136	616448	435200	346624	303360
8 Bytes	4096	12608	6688	3792	2376	1684	1346
	8192	25216	13376	7584	4752	3368	2692
	16384	50432	26752	15168	9504	6736	5384
	32768	100864	53504	30336	19008	13472	10768
	65536	201728	107008	60672	38016	26944	21536
	131072	403456	214016	121344	76032	53888	43072
	262144	806912	428032	242688	152064	107776	86144
	524288	1613824	856064	485376	304128	215552	172288
16 Bytes	4096	12096	6176	3280	1864	1172	834
	8192	24192	12352	6560	3728	2344	1668
	16384	48384	24704	13120	7456	4688	3336
	32768	96768	49408	26240	14912	9376	6672
	65536	193536	98816	52480	29824	18752	13344
	131072	387072	197632	104960	59648	37504	26688
	262144	774144	395264	209920	119296	75008	53376
	524288	1548288	790528	419840	238592	150016	106752
32 Bytes	4096		5920	3024	1608	916	578
	8192		11840	6048	3216	1832	1156
	16384		23680	12096	6432	3664	2312
	32768		47360	24192	12864	7328	4624
	65536		94720	48384	25728	14656	9248
	131072		189440	96768	51456	29312	18496
	262144		378880	193536	102912	58624	36992
	524288		757760	387072	205824	117248	73984
64 Bytes	4096			2896	1480	788	450
	8192			5792	2960	1576	900
	16384			11584	5920	3152	1800
	32768			23168	11840	6304	3600
	65536			46336	23680	12608	7200
	131072			92672	47360	25216	14400
	262144			185344	94720	50432	28800
	524288			370688	189440	100864	57600
128 Bytes	4096				1416	724	386
	8192				2832	1448	772
	16384				5664	2896	1544
	32768				11328	5792	3088
	65536				22656	11584	6176
	131072				45312	23168	12352
	262144				90624	46336	24704
	524288				181248	92672	49408
256 Bytes	4096					692	354
	8192					1384	708
	16384					2768	1416
	32768					5536	2832
	65536					11072	5664
	131072					22144	11328
	262144					44288	22656
	524288					88576	45312
512 Bytes	4096						338
	8192						676
	16384						1352
	32768						2704
	65536						5408
	131072						10816
	262144						21632
	524288						43264

Table 18: Number of address and status tag bits needed to implement a unified or data cache.

Fraction of Subsectors Used for Unified, Instruction, and Data Sector Caches																			
Sector Size (Bytes)																			
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.78494	0.63227	0.44771	0.29395	0.17891	0.09047	0.87573	0.76354	0.62038	0.48489	0.36297	0.25178	0.71685	0.53789	0.32337	0.17279	0.08999	0.03793
	8	0.85850	0.69450	0.49282	0.32615	0.19906	0.10192	0.92215	0.80740	0.66034	0.51726	0.38833	0.27019	0.81072	0.61354	0.36970	0.19940	0.10664	0.04648
	16	1.00000	0.81354	0.58802	0.39272	0.24132	0.12651	1.00000	0.88171	0.73130	0.57503	0.43336	0.30351	1.00000	0.76760	0.48589	0.26839	0.14672	0.06858
	32		1.00000	0.74391	0.50364	0.31252	0.16771		1.00000	0.83956	0.66345	0.50179	0.35485		1.00000	0.67614	0.39343	0.21923	0.10740
	64			1.00000	0.68719	0.43141	0.23874			1.00000	0.79413	0.60565	0.43199			1.00000	0.61085	0.34786	0.17821
	128				1.00000	0.64050	0.36611				1.00000	0.76865	0.55132				1.00000	0.57938	0.31026
	256					1.00000	0.58775					1.00000	0.72663					1.00000	0.54946
	512						1.00000						1.00000						1.00000
8K	4	0.80379	0.66442	0.51220	0.32598	0.19757	0.10938	0.88998	0.79589	0.66967	0.53125	0.38756	0.26784	0.74367	0.58287	0.40848	0.20050	0.09907	0.05004
	8	0.86909	0.72054	0.55818	0.35699	0.21739	0.12176	0.93132	0.83506	0.70627	0.56208	0.41187	0.28534	0.82307	0.64932	0.46030	0.22814	0.11327	0.05891
	16	1.00000	0.83106	0.64700	0.42253	0.25953	0.14764	1.00000	0.90025	0.76705	0.61350	0.45311	0.31538	1.00000	0.79456	0.57115	0.30268	0.15216	0.08085
	32		1.00000	0.78517	0.52993	0.33046	0.19147		1.00000	0.86049	0.69302	0.51913	0.36461		1.00000	0.73562	0.42486	0.22390	0.12077
	64			1.00000	0.70266	0.44813	0.26514			1.00000	0.81263	0.61695	0.43629			1.00000	0.63546	0.35141	0.19249
	128				1.00000	0.65231	0.39290				1.00000	0.77638	0.55553				1.00000	0.58035	0.32139
	256					1.00000	0.61131					1.00000	0.72753					1.00000	0.55990
	512						1.00000						1.00000						1.00000
16K	4	0.82064	0.69083	0.54636	0.40006	0.21993	0.11476	0.90617	0.82651	0.71639	0.58460	0.45100	0.31748	0.76063	0.61547	0.45788	0.29865	0.12265	0.05237
	8	0.87878	0.74226	0.58958	0.43364	0.24007	0.12586	0.94141	0.86051	0.74865	0.61342	0.47544	0.33618	0.83212	0.67629	0.50690	0.33504	0.13875	0.05959
	16	1.00000	0.84778	0.67644	0.49948	0.28357	0.15014	1.00000	0.91710	0.80250	0.66144	0.51620	0.36750	1.00000	0.81713	0.61775	0.41501	0.18385	0.08079
	32		1.00000	0.80531	0.59938	0.35423	0.19285		1.00000	0.88227	0.73381	0.57818	0.41576		1.00000	0.76802	0.52995	0.25690	0.12049
	64			1.00000	0.75421	0.46986	0.26441			1.00000	0.84095	0.67059	0.48884			1.00000	0.71406	0.38346	0.19128
	128				1.00000	0.66827	0.39066				1.00000	0.80939	0.60218				1.00000	0.60272	0.31888
	256					1.00000	0.60935					1.00000	0.75738					1.00000	0.55581
	512						1.00000						1.00000						1.00000
32K	4	0.82200	0.68623	0.54415	0.41281	0.29298	0.13226	0.91169	0.82154	0.70137	0.58196	0.46371	0.33787	0.78989	0.61858	0.46713	0.33128	0.22080	0.06654
	8	0.88795	0.74417	0.59206	0.44955	0.31869	0.14457	0.94594	0.85730	0.73611	0.61244	0.48956	0.35829	0.86954	0.68771	0.52131	0.37156	0.24616	0.07547
	16	1.00000	0.84262	0.67365	0.51291	0.36782	0.17210	1.00000	0.91559	0.79335	0.66269	0.53209	0.39210	1.00000	0.80442	0.61360	0.44068	0.30057	0.10087
	32		1.00000	0.80524	0.61545	0.44142	0.21684		1.00000	0.87697	0.73627	0.59441	0.44211		1.00000	0.77061	0.56013	0.37913	0.14280
	64			1.00000	0.77053	0.55418	0.29064			1.00000	0.84415	0.68622	0.51671			1.00000	0.74032	0.50201	0.21549
	128				1.00000	0.72898	0.41530				1.00000	0.81890	0.62673				1.00000	0.69013	0.34197
	256					1.00000	0.62678					1.00000	0.77858					1.00000	0.57311
	512						1.00000						1.00000						1.00000
64K	4	0.84124	0.71427	0.56623	0.43019	0.31084	0.20112	0.90048	0.82781	0.76139	0.64988	0.48635	0.36437	0.78912	0.63583	0.46186	0.34007	0.22756	0.13586
	8	0.90197	0.76948	0.61415	0.46895	0.34003	0.22107	0.93947	0.86414	0.79155	0.67762	0.51162	0.38544	0.87066	0.70598	0.51948	0.38420	0.25827	0.15502
	16	1.00000	0.85940	0.69399	0.53435	0.38938	0.25484	1.00000	0.92038	0.83818	0.72100	0.55289	0.42019	1.00000	0.81960	0.61679	0.45926	0.31093	0.18802
	32		1.00000	0.82131	0.63956	0.46913	0.30992		1.00000	0.90454	0.78284	0.61288	0.47125		1.00000	0.77855	0.58594	0.40100	0.24544
	64			1.00000	0.79192	0.58697	0.39332			1.00000	0.87210	0.70050	0.54593			1.00000	0.76611	0.53325	0.33249
	128				1.00000	0.75400	0.51643				1.00000	0.82792	0.65422				1.00000	0.71638	0.46011
	256					1.00000	0.70456					1.00000	0.79972					1.00000	0.66608
	512						1.00000						1.00000						1.00000
128K	4	0.82079	0.69771	0.56569	0.44959	0.33344	0.22243	0.87521	0.79016	0.68968	0.58199	0.48418	0.38523	0.80565	0.67148	0.53310	0.40932	0.28777	0.17992
	8	0.89099	0.75907	0.61785	0.49228	0.36642	0.24578	0.92459	0.83495	0.72947	0.61621	0.51288	0.40874	0.88258	0.73836	0.58897	0.45423	0.32114	0.20276
	16	1.00000	0.85520	0.70047	0.56056	0.42006	0.28478	1.00000	0.90349	0.79059	0.66889	0.55721	0.44515	1.00000	0.84208	0.67707	0.52631	0.37619	0.24117
	32		1.00000	0.82752	0.66709	0.50479	0.34761		1.00000	0.87670	0.74323	0.61977	0.49663		1.00000	0.81443	0.64159	0.46622	0.30591
	64			1.00000	0.81466	0.62458	0.43814			1.00000	0.84979	0.70965	0.57071			1.00000	0.80173	0.59432	0.39936
	128				1.00000	0.78109	0.56057				1.00000	0.83638	0.67550				1.00000	0.76069	0.52431
	256					1.00000	0.73870					1.00000	0.81111					1.00000	0.71437
	512						1.00000						1.00000						1.00000
256K	4	0.83418	0.72136	0.60430	0.49844	0.40087	0.30189	0.87199	0.78447	0.68624	0.58808	0.48745	0.39890	0.83213	0.71896	0.59606	0.48356	0.37569	0.27110
	8	0.89994	0.77968	0.65462	0.54118	0.43610	0.32937	0.92255	0.83075	0.72717	0.62263	0.51680	0.42281	0.89927	0.77920	0.64862	0.52849	0.41250	0.29909
	16	1.00000	0.86926	0.73242	0.60795	0.49149	0.37309	1.00000	0.90110	0.78919	0.67503	0.56148	0.45943	1.00000	0.87019	0.72896	0.59836	0.47043	0.34378
	32		1.00000	0.84750	0.70801	0.57541	0.44046		1.00000	0.87636	0.74880	0.62455	0.51101		1.00000	0.84706	0.70339	0.55970	0.41436
	64			1.00000	0.84146	0.68852	0.53255			1.00000	0.85360	0.71439	0.58459			1.00000	0.84169	0.67888	0.51041
	128				1.00000	0.82529	0.64617				1.00000	0.84026	0.68781				1.00000	0.81817	0.62694
	256					1.00000	0.79581					1.00000	0.81872					1.00000	0.78440
	512						1.00000						1.00000						1.00000
512K	4	0.85492	0.76003	0.65419	0.55383	0.46439	0.37941	0.86878	0.77659	0.67833	0.58526	0.49213	0.41051	0.86430	0.76823	0.66916	0.56792	0.47291	0.37702
	8	0.91347	0.81330	0.70131	0.59546	0.50024	0.40946	0.92126	0.82475	0.72056	0.62045	0.52201	0.43488	0.91970	0.82090	0.71686	0.61069	0.51002	0.40829
	16	1.00000	0.89223	0.77268	0.65898	0.55517	0.45581	1.00000	0.89804	0.78466	0.67394	0.56749	0.47190	1.00000	0.89760	0.78679	0.67460	0.56641	0.45648
	32		1.00000	0.87200	0.74867	0.63364	0.52282		1.00000	0.87435	0.74910	0.63161	0.52406		1.00000	0.88205	0.76366	0.64607	0.52616
	64			1.00000	0.86462	0.73537	0.60987			1.00000	0.85514	0.72223	0.59781			1.00000	0.87460	0.74606	0.61400
	128				1.00000	0.85427	0.71086				1.00000	0.84674	0.69964				1.00000	0.85813	0.71135
	256					1.00000	0.83474					1.00000	0.82603					1.00000	0.83459
	512						1.00000						1.00000						1.00000

Table 19: Fraction of subsectors touched while a sector is in the cache. SS is the subsector size in bytes.

Fraction of Sector Written Before Eviction for Unified and Data Sector Caches													
Sector Size (Bytes)													
Size		Unified Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.13113	0.09097	0.05905	0.03595	0.02009	0.01016	0.35851	0.26689	0.14718	0.07662	0.04047	0.01640
	8	0.14724	0.10309	0.06786	0.04213	0.02398	0.01259	0.39518	0.29491	0.16380	0.08609	0.04653	0.01960
	16	0.18793	0.13320	0.08944	0.05688	0.03373	0.01854	0.49086	0.36800	0.20732	0.11046	0.06146	0.02766
	32		0.17556	0.12504	0.08142	0.05043	0.02881		0.46162	0.26890	0.15312	0.08706	0.04185
	64			0.18646	0.12409	0.07977	0.04727			0.36598	0.22516	0.13176	0.06745
	128				0.19484	0.13167	0.08004				0.34615	0.20823	0.11391
	256					0.23112	0.14122					0.34968	0.19930
	512						0.25531						0.35747
8K	4	0.15740	0.12271	0.08931	0.04982	0.02572	0.01298	0.37225	0.29692	0.21086	0.09957	0.04411	0.02262
	8	0.17367	0.13587	0.09940	0.05610	0.02947	0.01532	0.41155	0.32742	0.23256	0.11040	0.04929	0.02582
	16	0.21798	0.17112	0.12561	0.07206	0.03878	0.02126	0.51941	0.41001	0.29053	0.13958	0.06294	0.03378
	32		0.21669	0.16008	0.09394	0.05455	0.03142		0.50582	0.36104	0.17740	0.08747	0.04769
	64			0.21326	0.12922	0.08186	0.04981			0.46202	0.23474	0.12854	0.07199
	128				0.18528	0.12760	0.08164				0.32222	0.19753	0.11260
	256					0.21407	0.14234					0.32683	0.18902
	512						0.25547						0.33087
16K	4	0.18729	0.15570	0.11038	0.07522	0.03714	0.01708	0.38235	0.30831	0.22769	0.14791	0.06309	0.02485
	8	0.20503	0.17043	0.12142	0.08309	0.04139	0.01930	0.42192	0.34059	0.25188	0.16401	0.06982	0.02766
	16	0.25427	0.21070	0.15165	0.10443	0.05260	0.02496	0.53491	0.43176	0.31927	0.20802	0.08792	0.03506
	32		0.25692	0.18805	0.13122	0.06736	0.03490		0.52887	0.39339	0.25888	0.11031	0.04853
	64			0.24027	0.17110	0.09008	0.05191			0.49191	0.32963	0.14310	0.07100
	128				0.23060	0.12533	0.08059				0.42698	0.19103	0.10850
	256					0.18931	0.13474					0.27952	0.17968
	512						0.23463						0.31380
32K	4	0.22688	0.18225	0.14043	0.10780	0.06619	0.02682	0.45457	0.35640	0.26755	0.18761	0.11524	0.03468
	8	0.25113	0.20212	0.15591	0.11860	0.07276	0.02975	0.49981	0.39585	0.29724	0.20864	0.12752	0.03862
	16	0.29505	0.23806	0.18391	0.13890	0.08946	0.03698	0.57857	0.46559	0.34984	0.24585	0.15915	0.04851
	32		0.29672	0.23017	0.17182	0.11048	0.04675		0.57930	0.43646	0.30793	0.19688	0.06132
	64			0.29027	0.21603	0.13992	0.06123			0.54452	0.38720	0.24711	0.07959
	128				0.27389	0.18160	0.08278				0.48199	0.31210	0.10566
	256					0.25706	0.12259					0.42594	0.15393
	512						0.19228						0.23904
64K	4	0.27203	0.22718	0.17293	0.12224	0.08450	0.05257	0.53618	0.42574	0.30795	0.22483	0.14854	0.08772
	8	0.29736	0.24963	0.19156	0.13554	0.09380	0.05845	0.58331	0.46588	0.34141	0.24935	0.16482	0.09742
	16	0.34288	0.29013	0.22556	0.15993	0.11085	0.06914	0.66617	0.53701	0.40216	0.29388	0.19440	0.11494
	32		0.35451	0.28121	0.20014	0.13912	0.08709		0.64716	0.50122	0.36712	0.24334	0.14431
	64			0.34882	0.25045	0.17523	0.11088			0.61786	0.45549	0.30373	0.18167
	128				0.30778	0.21869	0.14132				0.54955	0.37186	0.22644
	256					0.29254	0.19478					0.48679	0.30374
	512						0.28842						0.43799
128K	4	0.36441	0.30336	0.24073	0.18510	0.13127	0.07847	0.55999	0.46278	0.36482	0.27819	0.19358	0.11962
	8	0.39560	0.33027	0.26339	0.20294	0.14433	0.08696	0.60632	0.50255	0.39728	0.30362	0.21207	0.13207
	16	0.45070	0.37802	0.30381	0.23492	0.16789	0.10246	0.68570	0.57136	0.45446	0.34894	0.24519	0.15460
	32		0.45038	0.36677	0.28550	0.20574	0.12802		0.67403	0.54128	0.41928	0.29768	0.19148
	64			0.44111	0.34641	0.25192	0.15919			0.64258	0.50307	0.36107	0.23585
	128				0.41397	0.30385	0.19334				0.59792	0.43336	0.28445
	256					0.38692	0.25001					0.54838	0.36476
	512						0.34421						0.49708
256K	4	0.41307	0.34832	0.28352	0.22831	0.17775	0.12972	0.60161	0.51381	0.42157	0.33849	0.26077	0.18627
	8	0.44396	0.37547	0.30647	0.24737	0.19313	0.14152	0.64315	0.55040	0.45322	0.36535	0.28239	0.20227
	16	0.49646	0.42216	0.34634	0.28080	0.22036	0.16259	0.71349	0.61249	0.50702	0.41118	0.31955	0.23046
	32		0.49082	0.40554	0.33133	0.26194	0.19541		0.69900	0.58414	0.47905	0.37573	0.27360
	64			0.47464	0.39071	0.31116	0.23445			0.67203	0.55768	0.44175	0.32475
	128				0.45501	0.36485	0.27687				0.64401	0.51436	0.38149
	256					0.44330	0.34073					0.61999	0.46693
	512						0.43540						0.59225
512K	4	0.47531	0.41304	0.34498	0.28464	0.23192	0.18313	0.66215	0.58223	0.50041	0.42136	0.34630	0.27225
	8	0.50499	0.43957	0.36797	0.30451	0.24875	0.19707	0.69784	0.61655	0.53087	0.44843	0.36951	0.29143
	16	0.55376	0.48360	0.40675	0.33820	0.27744	0.22109	0.75547	0.67139	0.58025	0.49318	0.40815	0.32388
	32		0.54238	0.46024	0.38589	0.31853	0.25617		0.74350	0.64610	0.55443	0.46198	0.37004
	64			0.51997	0.43959	0.36517	0.29598			0.71822	0.62198	0.52220	0.42209
	128				0.49737	0.41484	0.33746				0.69572	0.58702	0.47732
	256					0.48213	0.39411					0.67359	0.55203
	512						0.46997						0.65074

Table 20: Fraction dirtiness of sectors evicted from unified and data caches.

Design Target Traffic Ratios for Unified, Instruction, and Data Sector Caches																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.43971	0.47444	0.47837	0.52784	0.66968	0.82820	0.35029	0.38482	0.42682	0.49652	0.57524	0.66655	0.43014	0.45068	0.40656	0.38311	0.44220	0.57735
	8	0.48275	0.52321	0.52928	0.58926	0.75056	0.94243	0.36886	0.40693	0.45431	0.52967	0.61541	0.71528	0.48235	0.50873	0.46094	0.43851	0.51914	0.70220
	16	0.57021	0.62106	0.63952	0.71937	0.92557	1.19379	0.40000	0.44438	0.50313	0.58883	0.68678	0.80348	0.59633	0.63593	0.59894	0.58191	0.70558	1.02268
	32		0.77116	0.82029	0.93610	1.22138	1.61740		0.50400	0.57762	0.67937	0.79523	0.93941		0.81850	0.81651	0.83949	1.03812	1.58614
	64			1.12001	1.29805	1.72019	2.35393			0.68800	0.81319	0.95983	1.14362			1.18020	1.28410	1.62563	2.61058
	128				1.91174	2.59850	3.67186				1.02400	1.21814	1.45953				2.06768	2.66951	4.50769
	256					4.14295	5.99956					1.58479	1.92362					4.57462	7.95711
	512						10.33147						2.64732						14.42580
8K	4	0.30758	0.31485	0.31760	0.30064	0.38552	0.48136	0.21360	0.23558	0.24644	0.27200	0.33554	0.40665	0.35708	0.37302	0.38646	0.30727	0.37063	0.41528
	8	0.33368	0.34256	0.34720	0.33047	0.42622	0.53926	0.22352	0.24718	0.25991	0.28778	0.35659	0.43321	0.39506	0.41413	0.43234	0.34666	0.42082	0.48428
	16	0.38975	0.40087	0.40794	0.39568	0.51505	0.66441	0.24000	0.26647	0.28227	0.31411	0.39229	0.47881	0.48619	0.51072	0.53767	0.45286	0.55683	0.65515
	32		0.48667	0.49909	0.49910	0.66473	0.87681		0.29600	0.31666	0.35483	0.44944	0.55356		0.63845	0.68430	0.61671	0.80602	0.96276
	64			0.64060	0.66550	0.91506	1.23897			0.36800	0.41606	0.53413	0.66239			0.91228	0.89108	1.24242	1.51146
	128				0.94823	1.34655	1.86678				0.51200	0.67217	0.84342				1.35395	2.01366	2.48020
	256					2.09616	2.96475					0.86577	1.10456					3.43469	4.28010
	512						4.93885						1.51823						7.60591
16K	4	0.24190	0.24380	0.24168	0.24334	0.28399	0.35314	0.18123	0.19175	0.20632	0.22449	0.25304	0.30583	0.27429	0.28820	0.28518	0.27150	0.31075	0.36284
	8	0.26011	0.26285	0.26165	0.26456	0.31093	0.38882	0.18828	0.19964	0.21561	0.23555	0.26676	0.32385	0.30094	0.31724	0.31563	0.30341	0.34894	0.40998
	16	0.30102	0.30483	0.30473	0.30920	0.37136	0.46903	0.20000	0.21277	0.23112	0.25399	0.28962	0.35401	0.36834	0.38962	0.38977	0.37879	0.45466	0.54437
	32		0.36199	0.36555	0.37406	0.46573	0.61003		0.23200	0.25409	0.28178	0.32440	0.40050		0.47697	0.48311	0.47959	0.61435	0.79425
	64			0.45642	0.47376	0.61856	0.84729			0.28800	0.32293	0.37625	0.47090			0.62060	0.63454	0.88094	1.23244
	128				0.63006	0.87668	1.26227				0.38400	0.45412	0.58008				0.86758	1.32798	2.00824
	256					1.31381	1.99307					0.56107	0.72959					2.14069	3.45600
	512						3.30699						0.96330						6.17345
32K	4	0.16781	0.16674	0.15334	0.14993	0.18900	0.21642	0.10940	0.11173	0.11222	0.13036	0.14058	0.16400	0.19905	0.19495	0.19980	0.19923	0.23586	0.25814
	8	0.18224	0.18168	0.16754	0.16362	0.20598	0.23715	0.11351	0.11659	0.11778	0.13719	0.14842	0.17392	0.21902	0.21666	0.22260	0.22277	0.26228	0.29096
	16	0.20719	0.20748	0.19208	0.18772	0.24063	0.28444	0.12000	0.12452	0.12694	0.14844	0.16131	0.19033	0.25249	0.25394	0.26201	0.26360	0.32267	0.38096
	32		0.24895	0.23192	0.22673	0.29042	0.35860		0.13600	0.14032	0.16492	0.18021	0.21461		0.31578	0.32826	0.33329	0.40429	0.52057
	64			0.28901	0.28412	0.36525	0.47870			0.16000	0.18909	0.20804	0.25082			0.42004	0.43292	0.52580	0.75253
	128				0.36687	0.47916	0.67762				0.22400	0.24827	0.30422				0.56902	0.70345	1.14158
	256					0.66149	1.01948					0.30317	0.37793					1.00085	1.85416
	512						1.62206						0.48541						3.15989
64K	4	0.12592	0.12779	0.11706	0.11249	0.12070	0.13350	0.07641	0.07961	0.08614	0.10294	0.12448	0.14417	0.14979	0.15001	0.14798	0.15333	0.15937	0.17015
	8	0.13565	0.13833	0.12760	0.12309	0.13245	0.14708	0.07972	0.08310	0.08955	0.10733	0.13095	0.15251	0.16434	0.16560	0.16549	0.17196	0.17928	0.19211
	16	0.15188	0.15603	0.14562	0.14137	0.15272	0.17048	0.08485	0.08851	0.09483	0.11420	0.14151	0.16626	0.18832	0.19171	0.19588	0.20443	0.21413	0.23056
	32		0.18385	0.17460	0.17098	0.18571	0.20892		0.09617	0.10234	0.12400	0.15687	0.18646		0.23277	0.24602	0.25869	0.27303	0.29661
	64			0.21360	0.21225	0.23271	0.26532			0.11314	0.13813	0.17929	0.21601			0.31101	0.33158	0.35466	0.39129
	128				0.26630	0.29697	0.34612				0.15839	0.21191	0.25886				0.42061	0.46114	0.52248
	256					0.39462	0.47325					0.25595	0.31643					0.63004	0.73806
	512						0.67798						0.39568						1.09437
128K	4	0.09471	0.09602	0.09026	0.09135	0.09779	0.11216	0.05251	0.05373	0.05517	0.06518	0.07247	0.08032	0.10899	0.11321	0.12194	0.13186	0.13756	0.14514
	8	0.10282	0.10449	0.09863	0.10006	0.10748	0.12404	0.05548	0.05678	0.05836	0.06902	0.07676	0.08522	0.11883	0.12385	0.13393	0.14535	0.15238	0.16225
	16	0.11593	0.11828	0.11240	0.11449	0.12373	0.14435	0.06000	0.06144	0.06325	0.07492	0.08340	0.09281	0.13454	0.14108	0.15366	0.16787	0.17758	0.19178
	32		0.13911	0.13367	0.13711	0.14953	0.17730		0.06800	0.07014	0.08324	0.09276	0.10354		0.16709	0.18411	0.20347	0.21831	0.24102
	64			0.16129	0.16711	0.18446	0.22267			0.08000	0.09518	0.10622	0.11899			0.22307	0.25026	0.27305	0.30780
	128				0.20351	0.22832	0.28103				0.11200	0.12518	0.14084				0.30649	0.34126	0.39191
	256					0.29187	0.36856					0.14967	0.16911					0.44253	0.52293
	512						0.50108						0.20849						0.72547
256K	4	0.07034	0.07244	0.07018	0.07389	0.07776	0.08246	0.03700	0.03772	0.03882	0.04657	0.05370	0.06105	0.08063	0.08674	0.09748	0.11127	0.11468	0.11912
	8	0.07579	0.07822	0.07597	0.08017	0.08456	0.08996	0.03914	0.03995	0.04114	0.04931	0.05694	0.06471	0.08675	0.09356	0.10555	0.12099	0.12521	0.13058
	16	0.08440	0.08745	0.08527	0.09036	0.09566	0.10234	0.04243	0.04333	0.04464	0.05346	0.06186	0.07031	0.09637	0.10433	0.11840	0.13665	0.14235	0.14956
	32		0.10095	0.09904	0.10567	0.11253	0.12148		0.04808	0.04957	0.05930	0.06881	0.07821		0.11956	0.13711	0.16005	0.16856	0.17918
	64			0.11656	0.12528	0.13434	0.14653			0.05657	0.06760	0.07871	0.08947			0.16019	0.18943	0.20194	0.21753
	128				0.14794	0.15994	0.17635				0.07920	0.09258	0.10526				0.22255	0.24013	0.26267
	256					0.19396	0.21714					0.11018	0.12530					0.29194	0.32595
	512						0.27423						0.15304						0.41475
512K	4	0.05276	0.05591	0.05564	0.06010	0.06273	0.06579	0.02606	0.02640	0.02713	0.03277	0.03889	0.04797	0.06015	0.06666	0.07868	0.09415	0.09679	0.10054
	8	0.05627	0.05972	0.05955	0.06451	0.06748	0.07094	0.02764	0.02804	0.02882	0.03475	0.04125	0.05082	0.06375	0.07096	0.08395	0.10080	0.10393	0.10835
	16	0.06163	0.06558	0.06568	0.07148	0.07501	0.07917	0.03000	0.03053	0.03139	0.03774	0.04484	0.05515	0.06919	0.07746	0.09198	0.11114	0.11516	0.12084
	32		0.07352	0.07419	0.08132	0.08578	0.09111		0.03400	0.03497	0.04195	0.04991	0.06124		0.08608	0.10282	0.12545	0.13093	0.13879
	64			0.08465	0.09349	0.09915	0.10595			0.04000	0.04789	0.05707	0.06986			0.11562	0.14246	0.14988	0.16046
	128				0.10734	0.11434	0.12261				0.05600	0.06691	0.08176				0.16143	0.17080	0.18410
	256					0.13354	0.14373					0.07902	0.09653					0.19781	0.21477
	512						0.17193						0.11686						0.25569

Table 21: Design target traffic ratios

Design Target Bus Utilization for Unified, Instruction and Data Sector Caches (15 Cycles Overhead, 0.333333333333333 Cycles Per Byte)																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	0.89083	0.89359	0.89352	0.90292	0.92543	0.94319	0.85122	0.86274	0.87455	0.89023	0.90381	0.91587	0.86108	0.85821	0.84529	0.83994	0.86052	0.90027
	8	0.83283	0.83659	0.83675	0.85140	0.88499	0.91332	0.76516	0.78235	0.80052	0.82391	0.84463	0.86336	0.79684	0.79319	0.77626	0.77142	0.80335	0.86201
	16	0.77844	0.78301	0.78644	0.80655	0.85087	0.89148	0.67033	0.69315	0.71891	0.74957	0.77734	0.80332	0.74499	0.74203	0.73197	0.73219	0.77317	0.85142
	32		0.74748	0.75824	0.78399	0.83725	0.88885		0.61788	0.64951	0.68550	0.71842	0.75087		0.71207	0.71512	0.72859	0.77582	0.86558
	64			0.76611	0.79688	0.85592	0.91568			0.60973	0.64871	0.68550	0.72199			0.73484	0.76217	0.81346	0.90638
	128				0.84451	0.90688	0.96781				0.64855	0.68703	0.72453				0.82605	0.87624	0.96262
	256					0.97767	1.00000					0.71301	0.75098					0.94744	1.00000
	512						1.00000						0.79339						1.00000
8K	4	0.84030	0.83594	0.83393	0.82774	0.86484	0.89319	0.77722	0.79372	0.80100	0.81627	0.84569	0.86914	0.82888	0.82596	0.82631	0.79196	0.82919	0.84621
	8	0.76074	0.75396	0.75150	0.74427	0.79572	0.83787	0.66380	0.68587	0.69659	0.71768	0.75903	0.79282	0.74967	0.74568	0.74764	0.70593	0.75476	0.78253
	16	0.68884	0.67945	0.67681	0.67340	0.73606	0.79146	0.54955	0.57530	0.58931	0.61490	0.66601	0.70879	0.68663	0.68196	0.68649	0.65522	0.71183	0.74781
	32		0.62816	0.62663	0.63142	0.70418	0.77072		0.48709	0.50396	0.53236	0.59049	0.63977		0.63947	0.64901	0.63635	0.70705	0.74795
	64			0.61452	0.63055	0.71378	0.78940			0.45524	0.48581	0.54811	0.60067			0.64896	0.65862	0.74176	0.78556
	128				0.67432	0.76618	0.84641				0.47989	0.54778	0.60316				0.71635	0.80729	0.84925
	256					0.85006	0.92929					0.57578	0.63392					0.88738	0.92342
	512						1.00000						0.68772						0.98816
16K	4	0.79494	0.78766	0.78465	0.78512	0.81169	0.84779	0.74748	0.75798	0.77116	0.78571	0.80518	0.83320	0.78612	0.78355	0.77621	0.76503	0.78643	0.81923
	8	0.69890	0.68853	0.68474	0.68579	0.72126	0.77087	0.62451	0.63814	0.65571	0.67540	0.70206	0.74098	0.69303	0.68850	0.67979	0.66844	0.69673	0.74020
	16	0.61501	0.60198	0.59782	0.59940	0.64455	0.70472	0.50413	0.51959	0.54020	0.56353	0.59551	0.64280	0.62180	0.61540	0.60661	0.59779	0.64356	0.69648
	32		0.54091	0.53813	0.54145	0.59744	0.67000		0.42672	0.44910	0.47481	0.50999	0.56235		0.56353	0.55707	0.55429	0.62183	0.69273
	64			0.51706	0.52415	0.59278	0.67827			0.39541	0.42307	0.46074	0.51675			0.54583	0.55169	0.64222	0.72847
	128				0.54859	0.63312	0.73123				0.40898	0.45006	0.51109				0.58580	0.69869	0.79476
	256					0.70845	0.81580					0.46797	0.53353					0.78107	0.87562
	512						0.91115						0.58286						0.94926
32K	4	0.71274	0.70477	0.68344	0.67635	0.72933	0.75843	0.64118	0.64601	0.64701	0.68043	0.69662	0.72817	0.71060	0.69719	0.69585	0.69203	0.73120	0.74919
	8	0.59992	0.58964	0.56509	0.55702	0.61737	0.65357	0.50067	0.50736	0.50989	0.54788	0.56729	0.60572	0.60325	0.58834	0.58607	0.58201	0.62554	0.65108
	16	0.50468	0.49291	0.46799	0.46038	0.52504	0.57069	0.37888	0.38762	0.39219	0.43007	0.45055	0.49174	0.51467	0.50120	0.49840	0.49503	0.54763	0.59588
	32		0.43235	0.40904	0.40262	0.46525	0.52249		0.30378	0.31043	0.34603	0.36635	0.40777		0.45272	0.45079	0.44956	0.49970	0.57567
	64			0.38776	0.38379	0.44613	0.51873			0.26650	0.30040	0.32085	0.36288			0.43882	0.44170	0.49225	0.59856
	128				0.39884	0.46622	0.55925				0.28758	0.30910	0.35410				0.46520	0.52124	0.65892
	256					0.52304	0.63822					0.32216	0.37205					0.58755	0.74773
	512						0.74165						0.41318						0.84088
64K	4	0.63911	0.63478	0.61236	0.60336	0.62028	0.64499	0.55516	0.56526	0.58454	0.62704	0.67031	0.70192	0.63270	0.62315	0.61493	0.61905	0.62682	0.64213
	8	0.51598	0.51013	0.48716	0.47811	0.49602	0.52340	0.41320	0.42332	0.44167	0.48667	0.53633	0.57395	0.51821	0.50676	0.50002	0.50410	0.51275	0.53021
	16	0.41754	0.41072	0.39062	0.38297	0.40026	0.42792	0.30135	0.31031	0.32526	0.36730	0.41839	0.45804	0.43009	0.41825	0.41571	0.41971	0.42904	0.44764
	32		0.34954	0.33400	0.32850	0.34527	0.37315		0.23579	0.24718	0.28460	0.33479	0.37431		0.36798	0.37239	0.37744	0.38825	0.40888
	64			0.31104	0.30884	0.32678	0.35695			0.20440	0.23878	0.28934	0.32910			0.35946	0.36773	0.38193	0.40669
	128				0.31574	0.33769	0.37367				0.22205	0.27634	0.31810				0.38165	0.40281	0.43541
	256					0.37790	0.42349					0.28635	0.33158					0.45637	0.49976
	512						0.50433						0.36465						0.59234
128K	4	0.54950	0.54474	0.52615	0.52790	0.54590	0.58566	0.46170	0.46741	0.47401	0.51566	0.54205	0.56745	0.55376	0.55121	0.56254	0.57779	0.58695	0.60005
	8	0.42785	0.42083	0.40250	0.40366	0.42127	0.46191	0.32887	0.33401	0.34015	0.37874	0.40408	0.42948	0.43681	0.43146	0.44146	0.45644	0.46653	0.48220
	16	0.33875	0.33001	0.31362	0.31435	0.33077	0.36991	0.23372	0.23798	0.24329	0.27579	0.29773	0.32055	0.35182	0.34424	0.35248	0.36646	0.37770	0.39584
	32		0.27676	0.26310	0.26418	0.28016	0.31882		0.17909	0.18369	0.21078	0.22936	0.24936		0.29425	0.30212	0.31637	0.32980	0.35196
	64			0.24168	0.24406	0.26119	0.30165			0.15374	0.17772	0.19433	0.21272			0.28153	0.29780	0.31459	0.34140
	128				0.24458	0.26499	0.31011				0.16794	0.18407	0.20242				0.30111	0.32326	0.35590
	256					0.29154	0.34558					0.19005	0.20956					0.35826	0.40099
	512						0.40563						0.23220						0.47177
256K	4	0.46734	0.46636	0.45534	0.46617	0.47929	0.49517	0.37666	0.38122	0.38802	0.43204	0.46728	0.49928	0.47601	0.48093	0.50282	0.53197	0.53812	0.54733
	8	0.35004	0.34623	0.33515	0.34408	0.35573	0.37068	0.25691	0.26082	0.26652	0.30341	0.33465	0.36370	0.36131	0.36176	0.38018	0.40723	0.41364	0.42346
	16	0.26948	0.26333	0.25288	0.25977	0.26955	0.28300	0.17741	0.18050	0.18496	0.21368	0.23923	0.26331	0.28258	0.27879	0.29289	0.31631	0.32287	0.33306
	32		0.21578	0.20634	0.21214	0.22088	0.23379		0.13365	0.13722	0.15985	0.18084	0.20058		0.23063	0.24218	0.26330	0.27087	0.28233
	64			0.18526	0.19095	0.19969	0.21318			0.11383	0.13308	0.15163	0.16886			0.21847	0.23920	0.24837	0.26188
	128				0.18669	0.19645	0.21163				0.12489	0.14298	0.15945				0.23383	0.24524	0.26203
	256					0.20885	0.22766					0.14728	0.16418					0.26100	0.28324
	512						0.25961						0.18166						0.32254
512K	4	0.38968	0.39427	0.38990	0.40649	0.41704	0.43009	0.29859	0.30132	0.30708	0.34867	0.38845	0.43931	0.40075	0.41149	0.44423	0.48443	0.49036	0.49988
	8	0.28142	0.28177	0.27659	0.28977	0.29846	0.30971	0.19623	0.19853	0.20293	0.23484	0.26706	0.30983	0.29293	0.29777	0.32344	0.35854	0.36400	0.37343
	16	0.21108	0.20780	0.20242	0.21225	0.21903	0.22831	0.13232	0.13436	0.13759	0.16097	0.18564	0.21895	0.22254	0.22207	0.24027	0.26878	0.27372	0.28276
	32		0.16552	0.16040	0.16818	0.17386	0.18202		0.09835	0.10089	0.11862	0.13803	0.16422		0.17828	0.19131	0.21501	0.21986	0.22912
	64			0.14027	0.14721	0.15251	0.16032			0.08327	0.09808	0.11473	0.13692			0.16661	0.18780	0.19299	0.20281
	128				0.14066	0.14612	0.15398				0.09167	0.10760	0.12842				0.17796	0.18359	0.19405
	256					0.15125	0.15984					0.11023	0.13144					0.18879	0.20066
	512						0.17639						0.14494						0.22010

Table 22: Design target bus utilization, 15 cycle latency, 3 cycles per word.

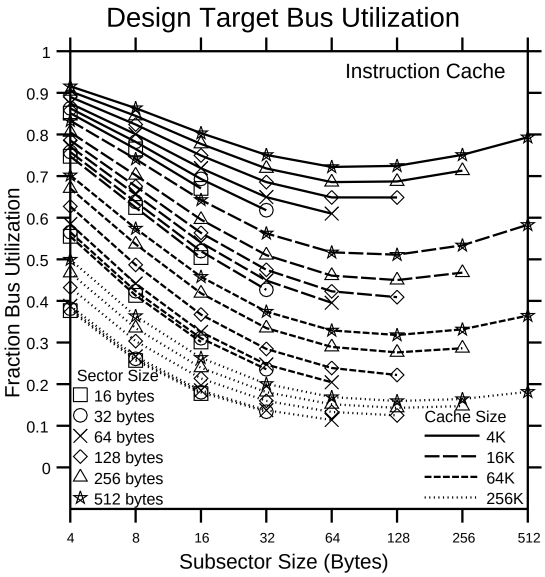


Figure 13: Instruction cache design target bus utilization, cache sizes 4K, 16K, 64K, 256K; 15 cycle latency, 3 cycles per word.

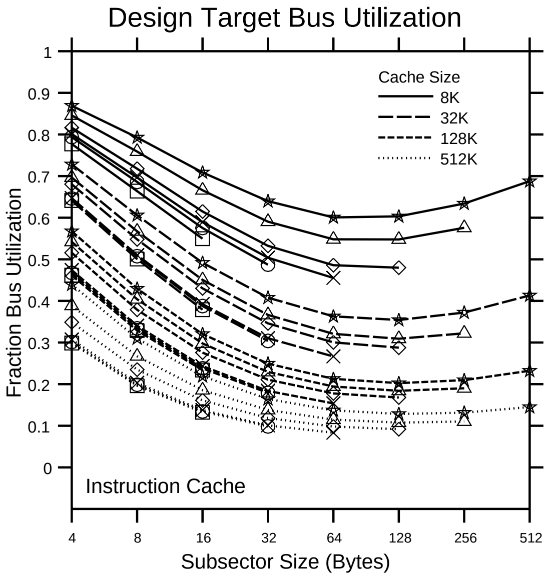


Figure 14: Instruction cache design target bus utilization, cache sizes 8K, 32K, 128K, 512K; 15 cycle latency, 3 cycles per word.

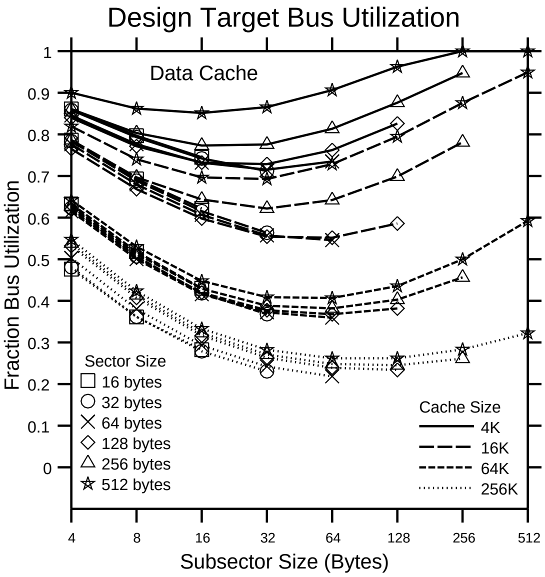


Figure 15: Data cache design target bus utilization, cache sizes 4K, 16K, 64K, 256K; 15 cycle latency, 3 cycles per word.

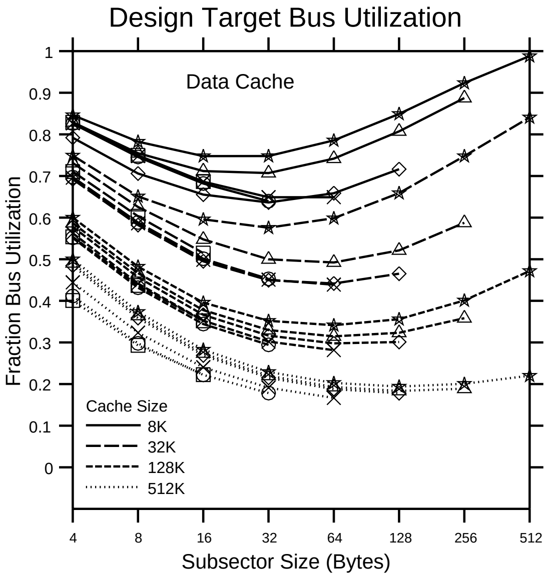


Figure 16: Data cache design target bus utilization, cache sizes 8K, 32K, 128K, 512K; 15 cycle latency, 3 cycles per word.

Average Memory Access Delay for Unified, Instruction, and Data Sector Caches (6 Cycles Overhead, 0.25 Cycles Per Byte)																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	2.36798	2.69732	2.95938	3.32169	3.97209	4.67553	1.96838	2.07226	2.42622	2.61416	2.87006	3.16689	2.21672	2.32284	2.56140	2.78474	3.05107	3.71059
	8	1.85495	2.06536	2.23247	2.47202	2.88958	3.36606	1.58269	1.64792	1.86747	1.98396	2.14324	2.32875	1.78631	1.86221	2.02004	2.17686	2.38883	2.89803
	16	1.62242	1.77998	1.91909	2.10780	2.43169	2.83565	1.39492	1.44222	1.60043	1.68365	1.79739	1.93287	1.60618	1.67419	1.83790	1.99005	2.19425	2.75025
	32		1.67113	1.81393	1.99447	2.29788	2.70340		1.35108	1.48252	1.55215	1.64631	1.76349		1.61482	1.81618	2.01589	2.24915	2.91876
	64			1.85967	2.06614	2.40771	2.90529			1.45158	1.51928	1.61292	1.73029			1.94845	2.23931	2.55736	3.50146
	128				2.33988	2.80499	3.52329				1.56473	1.67180	1.80493				2.75219	3.24015	4.76119
	256					3.59563	4.73108					1.80501	1.97712					4.56121	7.13514
	512						7.07605						2.28710						11.68713
8K	4	1.92774	2.00994	2.12780	2.39956	2.83070	3.30924	1.60830	1.64413	1.72824	1.80915	1.99816	2.20970	1.93179	2.00103	2.10340	2.24578	2.55604	2.73522
	8	1.57320	1.62585	1.70230	1.87581	2.15101	2.46886	1.36375	1.38619	1.43888	1.48920	1.60616	1.73642	1.58930	1.63723	1.71051	1.81000	2.01667	2.16739
	16	1.41222	1.45116	1.50879	1.64789	1.85884	2.11317	1.24411	1.26021	1.29791	1.33372	1.41678	1.50870	1.44748	1.48735	1.55100	1.67165	1.85357	2.00134
	32		1.38001	1.43221	1.56880	1.76550	2.01055		1.20233	1.23394	1.26389	1.33425	1.41168		1.42935	1.49677	1.65995	1.87920	2.04699
	64			1.43251	1.59258	1.81563	2.09951			1.21361	1.24312	1.31211	1.38706			1.53060	1.77556	2.08420	2.31112
	128				1.72833	2.02535	2.40715				1.25838	1.33921	1.42564				2.05404	2.54639	2.89060
	256					2.44779	3.01652					1.40242	1.51341					3.45423	4.03369
	512						4.15733						1.67545						6.18601
16K	4	1.64591	1.68477	1.79547	1.89019	2.05589	2.33587	1.40340	1.41944	1.45535	1.50782	1.57242	1.69183	1.83926	1.85913	1.89536	1.95804	2.08262	2.29828
	8	1.39524	1.42043	1.49051	1.55137	1.65861	1.83724	1.23948	1.24954	1.27191	1.30449	1.34482	1.41862	1.52465	1.53945	1.56642	1.61415	1.69985	1.84416
	16	1.28110	1.30012	1.35173	1.39693	1.48622	1.62422	1.15899	1.16622	1.18217	1.20520	1.23399	1.28600	1.39406	1.40737	1.43142	1.47546	1.57958	1.71538
	32		1.24781	1.29312	1.33343	1.42516	1.56124		1.12687	1.14019	1.15936	1.18346	1.22650		1.34898	1.37546	1.42500	1.56691	1.74682
	64			1.28599	1.32965	1.44310	1.60461			1.12485	1.14349	1.16719	1.20924			1.38411	1.44994	1.66488	1.93153
	128				1.37748	1.54427	1.77148				1.14736	1.17427	1.22261				1.54419	1.90256	2.34116
	256					1.75015	2.10834					1.19831	1.25788					2.37924	3.15306
	512						2.74094						1.32589						4.70774
32K	4	1.39076	1.41731	1.44122	1.50849	1.65942	1.76960	1.20209	1.22569	1.25273	1.26958	1.29072	1.33915	1.60293	1.63245	1.65159	1.67863	1.82682	1.90530
	8	1.24120	1.25860	1.27432	1.31642	1.40986	1.48070	1.11982	1.13458	1.15157	1.16211	1.17539	1.20552	1.37927	1.40179	1.41552	1.43495	1.52672	1.58676
	16	1.16978	1.18300	1.19508	1.22564	1.29566	1.35765	1.07917	1.08983	1.10210	1.10963	1.11914	1.14057	1.27261	1.29373	1.30568	1.32241	1.40197	1.49011
	32		1.15203	1.16323	1.18953	1.24837	1.31544		1.06868	1.07900	1.08526	1.09316	1.11095		1.25561	1.26873	1.28686	1.35493	1.48572
	64			1.15927	1.18643	1.24500	1.33220			1.07078	1.07681	1.08451	1.10188			1.27399	1.29790	1.36925	1.57589
	128				1.20896	1.27833	1.40995				1.07858	1.08710	1.10672				1.34752	1.43840	1.78928
	256					1.35167	1.56985					1.09796	1.12212					1.58510	2.21834
	512						1.87022						1.15012						3.03472
64K	4	1.27925	1.29133	1.31663	1.35100	1.38025	1.42404	1.10302	1.11017	1.13727	1.14942	1.18069	1.20928	1.44703	1.46600	1.49125	1.50824	1.53088	1.56918
	8	1.17109	1.17934	1.19624	1.21864	1.23770	1.26634	1.06141	1.06572	1.08154	1.08903	1.10862	1.12650	1.28184	1.29566	1.31573	1.32812	1.34429	1.37110
	16	1.11855	1.12519	1.13860	1.15571	1.17012	1.19189	1.04086	1.04375	1.05397	1.05920	1.07336	1.08619	1.20232	1.21453	1.23430	1.24514	1.25906	1.28131
	32		1.10197	1.11482	1.13046	1.14348	1.16336		1.03327	1.04077	1.04500	1.05693	1.06767		1.18322	1.20702	1.21893	1.23387	1.25706
	64			1.10984	1.12692	1.14105	1.16289			1.03541	1.03939	1.05112	1.06159			1.20893	1.22491	1.24436	1.27360
	128				1.13842	1.15648	1.18471				1.03900	1.05218	1.06374				1.25354	1.28351	1.32699
	256					1.19115	1.23211					1.05805	1.07177					1.36451	1.43600
	512						1.31532						1.08590						1.62653
128K	4	1.17577	1.18306	1.19338	1.20513	1.22240	1.26275	1.05052	1.05408	1.05852	1.06471	1.07194	1.07973	1.38359	1.39569	1.40891	1.42236	1.44229	1.46873
	8	1.10903	1.11381	1.12069	1.12835	1.13966	1.16591	1.03050	1.03266	1.03537	1.03915	1.04355	1.04834	1.24013	1.24863	1.25815	1.26783	1.28204	1.30185
	16	1.07648	1.08014	1.08552	1.09134	1.10007	1.12015	1.02061	1.02209	1.02396	1.02656	1.02957	1.03291	1.17004	1.17722	1.18548	1.19395	1.20650	1.22439
	32		1.06559	1.07072	1.07609	1.08417	1.10266		1.01711	1.01860	1.02066	1.02302	1.02570		1.14732	1.15618	1.16551	1.17914	1.19924
	64			1.06715	1.07301	1.08183	1.10167			1.01667	1.01856	1.02071	1.02320			1.15067	1.16250	1.17943	1.20437
	128				1.07740	1.08838	1.11234				1.01886	1.02108	1.02372				1.17505	1.19834	1.23172
	256					1.10422	1.13635					1.02322	1.02623					1.24015	1.29079
	512						1.17667						1.03095						1.38962
256K	4	1.13872	1.14533	1.15296	1.16010	1.17013	1.18207	1.02568	1.02831	1.03192	1.03749	1.04323	1.04915	1.33188	1.34021	1.35244	1.36408	1.37627	1.39217
	8	1.08552	1.08976	1.09468	1.09933	1.10576	1.11352	1.01552	1.01713	1.01933	1.02268	1.02619	1.02977	1.20495	1.21070	1.21915	1.22738	1.23608	1.24724
	16	1.05939	1.06254	1.06621	1.06974	1.07449	1.08036	1.01052	1.01162	1.01311	1.01537	1.01779	1.02022	1.14244	1.14706	1.15393	1.16090	1.16827	1.17762
	32		1.05037	1.05363	1.05685	1.06105	1.06641		1.00902	1.01019	1.01193	1.01385	1.01574		1.11830	1.12521	1.13240	1.14014	1.14986
	64			1.04972	1.05309	1.05740	1.06309			1.00914	1.01069	1.01245	1.01415			1.11614	1.12448	1.13356	1.14504
	128				1.05449	1.05942	1.06611				1.01082	1.01264	1.01438				1.12773	1.13901	1.15386
	256					1.06631	1.07499					1.01386	1.01576					1.15649	1.17730
	512						1.09020						1.01843						1.21635
512K	4	1.11029	1.11544	1.12198	1.12779	1.13454	1.14258	1.01220	1.01316	1.01474	1.01829	1.02170	1.02677	1.28085	1.28832	1.29838	1.30914	1.31905	1.33278
	8	1.06734	1.07059	1.07473	1.07851	1.08282	1.08793	1.00739	1.00799	1.00895	1.01108	1.01315	1.01620	1.17077	1.17605	1.18266	1.18995	1.19662	1.20593
	16	1.04607	1.04840	1.05146	1.05430	1.05744	1.06118	1.00502	1.00543	1.00609	1.00752	1.00894	1.01099	1.11605	1.12031	1.12530	1.13115	1.13647	1.14390
	32		1.03797	1.04065	1.04319	1.04589	1.04912		1.00424	1.00475	1.00585	1.00696	1.00854		1.09383	1.09833	1.10392	1.10897	1.11610
	64			1.03663	1.03919	1.04185	1.04502			1.00427	1.00525	1.00625	1.00766			1.08759	1.09352	1.09887	1.10645
	128				1.03914	1.04199	1.04532				1.00530	1.00633	1.00774				1.09234	1.09821	1.10652
	256					1.04527	1.04901					1.00689	1.00842					1.10541	1.11510
	512						1.05620						1.00975						1.13200

Table 23: Design target delay with 6 cycle bus access overhead, 1 cycle per word transferred.

Average Memory Access Delay for Unified, Instruction, and Data Sector Caches (15 Cycles Overhead, 0.25 Cycles Per Byte)																			
		Sector Size (Bytes)																	
Size		Unified Cache						Instruction Cache						Data Cache					
Cache	SS	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	4	4.12681	4.87959	5.47857	6.30671	7.79334	9.40121	3.21343	3.45089	4.25993	4.68951	5.27441	5.95290	3.78107	4.02363	4.56891	5.07941	5.68815	7.19563
	8	2.81677	3.26389	3.61899	4.12804	5.01537	6.02789	2.23821	2.37682	2.84337	3.09091	3.42938	3.82359	2.67090	2.83220	3.16759	3.50083	3.95126	5.03332
	16	2.18260	2.48197	2.74626	3.10481	3.72021	4.48773	1.75036	1.84021	2.14082	2.29894	2.51503	2.77246	2.15174	2.28097	2.59201	2.88109	3.26908	4.32547
	32		2.10256	2.33717	2.63378	3.13223	3.79845		1.57678	1.79271	1.90710	2.06180	2.25430		2.01006	2.34087	2.66897	3.05218	4.15225
	64			2.21135	2.50229	2.98359	3.68472			1.63631	1.73171	1.86367	2.02904			2.33645	2.74631	3.19446	4.52479
	128				2.65723	3.23249	4.12091				1.69848	1.83091	1.99557				3.16718	3.77071	5.65200
	256					3.92935	5.21079					1.90851	2.10275					5.01908	7.92394
	512						7.48414						2.37354						12.40492
8K	4	3.12054	3.30844	3.57783	4.19899	5.18446	6.27826	2.39040	2.47229	2.66455	2.84948	3.28152	3.76503	3.12980	3.28806	3.52207	3.84749	4.55665	4.96621
	8	2.21806	2.32994	2.49239	2.86109	3.44589	4.12132	1.77296	1.82065	1.93263	2.03955	2.28809	2.56489	2.25226	2.35410	2.50983	2.72125	3.16043	3.48070
	16	1.78321	1.85719	1.96670	2.23098	2.63180	3.11502	1.46380	1.49440	1.56602	1.63407	1.79188	1.96654	1.85022	1.92596	2.04690	2.27613	2.62179	2.90255
	32		1.62430	1.71006	1.93445	2.25760	2.66019		1.33240	1.38433	1.43353	1.54913	1.67633		1.70536	1.81612	2.08420	2.44439	2.72005
	64			1.60944	1.83499	2.14930	2.54931			1.30099	1.34258	1.43980	1.54540			1.74766	2.09283	2.52774	2.84749
	128				1.90083	2.26820	2.74042				1.31958	1.41955	1.52645				2.30368	2.91265	3.33837
	256					2.63393	3.27579					1.45416	1.57943					3.76978	4.42373
	512						4.36939						1.72082						6.53433
16K	4	2.47636	2.56519	2.81821	3.03471	3.41347	4.05341	1.92206	1.95872	2.04080	2.16072	2.30838	2.58132	2.91830	2.96372	3.04655	3.18981	3.47455	3.96751
	8	1.83989	1.89341	2.04234	2.17167	2.39955	2.77913	1.50890	1.53027	1.57782	1.64703	1.73275	1.88957	2.11489	2.14634	2.20363	2.30507	2.48719	2.79385
	16	1.53409	1.57023	1.66829	1.75418	1.92381	2.18602	1.30208	1.31582	1.34612	1.38988	1.44458	1.54341	1.74872	1.77400	1.81970	1.90338	2.10121	2.35922
	32		1.40711	1.48155	1.54777	1.69849	1.92204		1.20843	1.23032	1.26180	1.30140	1.37210		1.57332	1.61682	1.69822	1.93136	2.22693
	64			1.40298	1.46451	1.62437	1.85194			1.17593	1.20219	1.23558	1.29484			1.54125	1.63400	1.93688	2.31261
	128				1.46688	1.67318	1.95420				1.18226	1.21555	1.27533				1.67308	2.11632	2.65881
	256					1.84660	2.25084					1.22381	1.29103					2.55657	3.42988
	512						2.85786						1.34778						4.95677
32K	4	1.89316	1.95385	2.00850	2.16227	2.50724	2.75908	1.46193	1.51586	1.57767	1.61617	1.66450	1.77521	2.37813	2.44559	2.48935	2.55116	2.88987	3.06925
	8	1.51256	1.54952	1.58294	1.67240	1.87096	2.02148	1.25462	1.28598	1.32209	1.34449	1.37269	1.43673	1.80596	1.85380	1.88299	1.92426	2.11928	2.24687
	16	1.32257	1.34771	1.37065	1.42872	1.56175	1.67954	1.15042	1.17068	1.19399	1.20830	1.22636	1.26708	1.51796	1.55809	1.58079	1.61259	1.76375	1.93121
	32		1.24976	1.26816	1.31136	1.40804	1.51822		1.11283	1.12979	1.14008	1.15306	1.18227		1.41992	1.44148	1.47128	1.58309	1.79797
	64			1.22443	1.26270	1.34523	1.46809			1.09974	1.10823	1.11908	1.14356			1.38608	1.41977	1.52031	1.81149
	128				1.25845	1.34425	1.50704				1.09719	1.10772	1.13200				1.42983	1.54223	1.97622
	256					1.39688	1.64312					1.11055	1.13782					1.66032	2.37498
	512						1.92866						1.16021						3.17138
64K	4	1.63829	1.66589	1.72372	1.80230	1.86915	1.96924	1.23546	1.25181	1.31375	1.34153	1.41301	1.47835	2.02177	2.06514	2.12285	2.16170	2.21344	2.30098
	8	1.36357	1.38110	1.41702	1.46462	1.50511	1.56597	1.13051	1.13964	1.17328	1.18918	1.23081	1.26882	1.59890	1.62828	1.67093	1.69725	1.73161	1.78859
	16	1.22525	1.23786	1.26333	1.29585	1.32323	1.36459	1.07763	1.08312	1.10254	1.11249	1.13939	1.16377	1.38440	1.40761	1.44517	1.46576	1.49221	1.53449
	32		1.16752	1.18863	1.21432	1.23571	1.26838		1.05466	1.06698	1.07392	1.09352	1.11117		1.30101	1.34011	1.35967	1.38421	1.42231
	64			1.15478	1.17885	1.19875	1.22953			1.04990	1.05550	1.07204	1.08679			1.29440	1.31691	1.34432	1.38553
	128				1.17120	1.19354	1.22846				1.04824	1.06454	1.07884				1.31359	1.35066	1.40444
	256					1.21572	1.26195					1.06552	1.08100					1.41138	1.49206
	512						1.33650						1.09167						1.66861
128K	4	1.40176	1.41843	1.44201	1.46887	1.50835	1.60058	1.11547	1.12362	1.13376	1.14791	1.16444	1.18225	1.87678	1.90443	1.93465	1.96539	2.01095	2.07137
	8	1.23169	1.24184	1.25647	1.27274	1.29677	1.35256	1.06481	1.06940	1.07516	1.08319	1.09254	1.10273	1.51027	1.52833	1.54857	1.56913	1.59934	1.64144
	16	1.14531	1.15226	1.16249	1.17355	1.19012	1.22828	1.03917	1.04196	1.04552	1.05047	1.05618	1.06252	1.32309	1.33672	1.35241	1.36851	1.39234	1.42634
	32		1.10776	1.11619	1.12501	1.13828	1.16865		1.02811	1.03055	1.03394	1.03782	1.04222		1.24202	1.25657	1.27190	1.29430	1.32733
	64			1.09462	1.10288	1.11531	1.14326			1.02349	1.02615	1.02919	1.03269			1.21231	1.22897	1.25283	1.28797
	128				1.09573	1.10932	1.13894				1.02333	1.02608	1.02934				1.21650	1.24532	1.28661
	256					1.11762	1.15388					1.02620	1.02960					1.27103	1.32818
	512						1.18853						1.03303						1.41579
256K	4	1.31709	1.33218	1.34962	1.36594	1.38887	1.41617	1.05869	1.06472	1.07297	1.08570	1.09882	1.11233	1.75857	1.77763	1.80557	1.83218	1.86005	1.89639
	8	1.18173	1.19074	1.20120	1.21108	1.22474	1.24122	1.03299	1.03641	1.04108	1.04820	1.05566	1.06325	1.43551	1.44773	1.46570	1.48318	1.50166	1.52538
	16	1.11285	1.11883	1.12580	1.13251	1.14154	1.15269	1.01998	1.02207	1.02491	1.02920	1.03379	1.03841	1.27063	1.27942	1.29248	1.30570	1.31971	1.33747
	32		1.08274	1.08811	1.09340	1.10030	1.10910		1.01482	1.01674	1.01961	1.02275	1.02586		1.19435	1.20571	1.21751	1.23023	1.24619
	64			1.07006	1.07481	1.08088	1.08890			1.01288	1.01506	1.01754	1.01993			1.16366	1.17540	1.18819	1.20437
	128				1.06739	1.07349	1.08177				1.01338	1.01564	1.01778				1.15798	1.17193	1.19030
	256					1.07484	1.08464					1.01564	1.01779					1.17661	1.20010
	512						1.09626						1.01966						1.23088
512K	4	1.25209	1.26385	1.27882	1.29209	1.30753	1.32590	1.02789	1.03008	1.03370	1.04180	1.04960	1.06118	1.64195	1.65901	1.68201	1.70660	1.72925	1.76064
	8	1.14310	1.14999	1.15879	1.16684	1.17598	1.18685	1.01571	1.01697	1.01902	1.02354	1.02795	1.03443	1.36289	1.37411	1.38815	1.40365	1.41782	1.43761
	16	1.08754	1.09196	1.09777	1.10318	1.10914	1.11623	1.00953	1.01032	1.01157	1.01429	1.01698	1.02088	1.22050	1.22859	1.23806	1.24918	1.25930	1.27341
	32		1.06238	1.06678	1.07095	1.07540	1.08070		1.00696	1.00780	1.00961	1.01144	1.01403		1.15414	1.16154	1.17073	1.17902	1.19074
	64			1.05161	1.05522	1.05897	1.06344			1.00602	1.00740	1.00881	1.01079			1.12342	1.13177	1.13932	1.15000
	128				1.04841	1.05193	1.05605				1.00656	1.00783	1.00957				1.11421	1.12147	1.13174
	256					1.05109	1.05532					1.00777	1.00950					1.11897	1.12990
	512						1.05998						1.01041						1.14087

Table 24: Design target delay with 15 cycle bus access overhead, 1 cycle per word transferred.

Comparison of L1 Cache Organizations – 6 Cycle Overhead Memory Delay						
Budget	**Unified Cache**		**Instruction Cache**		**Data Cache**	
(Bytes)	**Normal**	**Sector**	**Normal**	**Sector**	**Normal**	**Sector**
5.26K	2.148 c4K 32/32	2.148 c4K 32/32	1.882 c4K 32/32	1.882 c4K 32/32	1.980 c4K 32/32	1.980 c4K 32/32
6.62K	2.148 c4K 32/32	2.148 c4K 32/32	1.882 c4K 32/32	1.882 c4K 32/32	1.980 c4K 32/32	1.980 c4K 32/32
8.35K	1.950 c8K 128/128	**1.773 c8K** **128/64**	1.608 c8K 128/128	**1.572 c8K** **128/64**	1.980 c4K 32/32	**1.895 c8K** **128/64**
10.52K	1.700 c8K 32/32	1.700 c8K 32/32	1.506 c8K 64/64	1.506 c8K 64/64	1.742 c8K 32/32	1.742 c8K 32/32
13.25K	1.700 c8K 32/32	1.700 c8K 32/32	1.506 c8K 64/64	1.506 c8K 64/64	1.742 c8K 32/32	1.742 c8K 32/32
16.69K	1.608 c16K 128/128	**1.531 c16K** **128/64**	1.456 c16K 128/128	**1.444 c16K** **128/64**	1.722 c16K 128/128	**1.597 c16K** **128/64**
21.03K	1.504 c16K 32/32	1.504 c16K 32/32	1.396 c16K 64/64	1.396 c16K 64/64	1.546 c16K 32/32	1.546 c16K 32/32
26.50K	1.504 c16K 32/32	1.504 c16K 32/32	1.396 c16K 64/64	1.396 c16K 64/64	1.546 c16K 32/32	1.546 c16K 32/32
33.39K	1.342 c32K 128/128	**1.305 c32K** **128/64**	1.266 c32K 128/128	**1.260 c32K** **128/64**	1.456 c32K 128/128	**1.391 c32K** **128/64**
42.06K	1.308 c32K 64/64	**1.305 c32K** **128/64**	1.220 c32K 64/64	1.220 c32K 64/64	1.350 c32K 32/32	1.350 c32K 32/32
53.00K	1.308 c32K 64/64	**1.305 c32K** **128/64**	1.220 c32K 64/64	1.220 c32K 64/64	1.350 c32K 32/32	1.350 c32K 32/32
66.77K	1.242 c64K 128/128	**1.222 c64K** **128/64**	1.188 c64K 128/128	1.188 c64K 128/128	1.322 c64K 128/128	**1.278 c64K** **128/32**
84.13K	1.218 c64K 64/64	1.218 c64K 64/64	1.156 c64K 64/64	1.156 c64K 64/64	1.247 c64K 32/32	1.247 c64K 32/32
105.99K	1.218 c64K 64/64	1.218 c64K 64/64	1.156 c64K 64/64	1.156 c64K 64/64	1.247 c64K 32/32	1.247 c64K 32/32
133.54K	1.171 c128K 128/128	**1.161 c128K** **128/64**	1.133 c128K 128/128	**1.131 c128K** **128/64**	1.228 c128K 128/128	**1.212 c128K** **128/64**
168.25K	1.154 c128K 64/64	1.154 c128K 64/64	1.110 c128K 64/64	1.110 c128K 64/64	1.175 c128K 32/32	1.175 c128K 32/32
211.98K	1.154 c128K 64/64	1.154 c128K 64/64	1.110 c128K 64/64	1.110 c128K 64/64	1.175 c128K 32/32	1.175 c128K 32/32
267.08K	1.121 c256K 128/128	**1.118 c256K** **128/64**	1.094 c256K 128/128	**1.093 c256K** **128/64**	1.161 c256K 128/128	**1.157 c256K** **128/64**
336.50K	1.109 c256K 64/64	1.109 c256K 64/64	1.078 c256K 64/64	1.078 c256K 64/64	1.124 c256K 32/32	1.124 c256K 32/32
423.96K	1.109 c256K 64/64	1.109 c256K 64/64	1.078 c256K 64/64	1.078 c256K 64/64	1.124 c256K 32/32	1.124 c256K 32/32
534.16K	1.085 c512K 128/128	1.085 c512K 128/128	1.067 c512K 128/128	**1.066 c512K** **128/64**	1.114 c512K 128/128	1.114 c512K 128/128
673.00K	1.077 c512K 64/64	1.077 c512K 64/64	1.055 c512K 64/64	1.055 c512K 64/64	1.087 c512K 32/32	1.087 c512K 32/32

Table 25: Best L1 cache for a given total (gross) bit budget, 6 cycle main memory latency, 1 word per cycle transfer bandwidth. Each entry consists of memory delay, cache size, sector/subsector size. Sector cache organization are shown in **bold**.

Comparison of L1 Cache Organizations – 15 Cycle Overhead Memory Delay						
Budget	**Unified Cache**		**Instruction Cache**		**Data Cache**	
(Bytes)	**Normal**	**Sector**	**Normal**	**Sector**	**Normal**	**Sector**
5.26K	2.829 c4K 64/64	2.829 c4K 64/64	2.333 c4K 64/64	2.333 c4K 64/64	2.610 c4K 32/32	2.610 c4K 32/32
6.62K	2.829 c4K 64/64	2.829 c4K 64/64	2.333 c4K 64/64	2.333 c4K 64/64	2.610 c4K 32/32	2.610 c4K 32/32
8.35K	2.175 c8K 128/128	**2.089 c8K** **128/64**	1.752 c8K 128/128	1.752 c8K 128/128	2.504 c8K 128/128	**2.261 c8K** **128/64**
10.52K	2.023 c8K 64/64	2.023 c8K 64/64	1.713 c8K 64/64	1.713 c8K 64/64	2.209 c8K 64/64	2.209 c8K 64/64
13.25K	2.023 c8K 64/64	2.023 c8K 64/64	1.713 c8K 64/64	1.713 c8K 64/64	2.209 c8K 64/64	2.209 c8K 64/64
16.69K	1.752 c16K 128/128	**1.748 c16K** **128/64**	1.564 c16K 128/128	1.564 c16K 128/128	1.893 c16K 128/128	**1.841 c16K** **128/64**
21.03K	1.713 c16K 64/64	1.713 c16K 64/64	1.558 c16K 64/64	1.558 c16K 64/64	1.806 c16K 64/64	1.806 c16K 64/64
26.50K	1.713 c16K 64/64	1.713 c16K 64/64	1.558 c16K 64/64	1.558 c16K 64/64	1.806 c16K 64/64	1.806 c16K 64/64
33.39K	1.423 c32K 128/128	1.423 c32K 128/128	1.329 c32K 128/128	1.329 c32K 128/128	1.564 c32K 128/128	**1.551 c32K** **128/64**
42.06K	1.423 c32K 128/128	1.423 c32K 128/128	1.310 c32K 64/64	1.310 c32K 64/64	1.527 c32K 64/64	1.527 c32K 64/64
53.00K	1.423 c32K 128/128	1.423 c32K 128/128	1.310 c32K 64/64	1.310 c32K 64/64	1.527 c32K 64/64	1.527 c32K 64/64
66.77K	1.299 c64K 128/128	1.299 c64K 128/128	1.233 c64K 128/128	1.233 c64K 128/128	1.399 c64K 128/128	1.399 c64K 128/128
84.13K	1.299 c64K 128/128	1.299 c64K 128/128	1.219 c64K 64/64	1.219 c64K 64/64	1.373 c64K 64/64	1.373 c64K 64/64
105.99K	1.299 c64K 128/128	1.299 c64K 128/128	1.219 c64K 64/64	1.219 c64K 64/64	1.373 c64K 64/64	1.373 c64K 64/64
133.54K	1.212 c128K 128/128	1.212 c128K 128/128	1.165 c128K 128/128	1.165 c128K 128/128	1.282 c128K 128/128	1.282 c128K 128/128
168.25K	1.212 c128K 128/128	1.212 c128K 128/128	1.155 c128K 64/64	1.155 c128K 64/64	1.264 c128K 64/64	1.264 c128K 64/64
211.98K	1.212 c128K 128/128	1.212 c128K 128/128	1.155 c128K 64/64	1.155 c128K 64/64	1.264 c128K 64/64	1.264 c128K 64/64
267.08K	1.150 c256K 128/128	1.150 c256K 128/128	1.116 c256K 128/128	1.116 c256K 128/128	1.199 c256K 128/128	1.199 c256K 128/128
336.50K	1.150 c256K 128/128	1.150 c256K 128/128	1.110 c256K 64/64	1.110 c256K 64/64	1.186 c256K 64/64	1.186 c256K 64/64
423.96K	1.150 c256K 128/128	1.150 c256K 128/128	1.110 c256K 64/64	1.110 c256K 64/64	1.186 c256K 64/64	1.186 c256K 64/64
534.16K	1.106 c512K 128/128	1.106 c512K 128/128	1.082 c512K 128/128	1.082 c512K 128/128	1.141 c512K 128/128	1.141 c512K 128/128
673.00K	1.106 c512K 128/128	1.106 c512K 128/128	1.077 c512K 64/64	1.077 c512K 64/64	1.132 c512K 64/64	1.132 c512K 64/64

Table 26: Best L1 cache for a given total (gross) bit budget, 15 cycle main memory latency, 1 word per cycle transfer bandwidth. Each entry consists of memory delay, cache size, sector/subsector size. Sector cache organization are shown in **bold**.

B Individual Workload Miss Ratios

Our first attempts to measure miss ratios for sector caches involved averaging the miss ratios of each of the individual workloads. Most of the workloads have trace lengths of 10 million instruction references, and 2 to 4 million data references. Some of the traces reference on the order of 100K bytes of distinct data locations, and many fewer distinct instruction locations. Table 27 shows the instruction and data miss ratios of the workloads over the cache sizes we examined (4–512K bytes) with 16–byte blocks. This table shows that with increasing cache size many of the workloads show no improvement in the miss ratio at relatively small cache sizes (which we refer to as *saturation*). Using these individual workloads to measure performance results in inaccurate measurement of the larger caches, so that many of the workloads only measure cold start misses, and never sample capacity misses. To rectify this situation, we multiprogrammed simulation of the traces to create one long trace. The resulting trace accesses almost 8 million distinct data locations, and had in excess of 300 million total instruction and data references.

The rest of this appendix shows the miss ratio plots of groups of the programs used in this paper, split into different workloads. There is a wide variation in program miss ratios and the raw data is made available here for further examination.

Figures 17–25 show the variation in the miss ratios for the different workloads over a number of cache sizes, and for various types of caches. The block size is set at 32 bytes, which is generally an optimal size. These graphs provide insight into how each program we used in our workload varies from the other programs in terms of miss ratio over various cache sizes.

Workload 1 consists of the SPEC92 integer workloads cc1, compress, eqntott, espresso, and xlisp. Workload 2 is the floating–point members of the SPEC92 programs we analyzed: alvinn, doduc, fpppp, and tomcatv. In general, these first two sub-workloads were much bigger in text and data size than the remaining programs we looked at, except for workload 5.

Workload 3 consists of programs from the Stanford SPLASH collection: barnes, cholesky, locus, mp3d, water. These are all parallel programs, but were run in uniprocessor mode to characterize their behavior. Workload 4 is a few other parallel programs which were created at U.C. Berkeley. These programs are matrix, maxflow, and SOR.

Workload 5 consists of the traces of UNIX tools and other traces. These programs are an as1, cpp, fcom, troff, and yacc. The MVStrace contains user and kernel calls from an IBM 370 using the MVS operating system. These programs have the highest miss ratios of all groups.

Figures 26–35 show the data cache miss ratios over the various sub-workloads. Workloads 2 and 3 show little variation between the different sector sizes for the larger cache sizes, the strong factor being the subsector size. Workload 4, which consisted of two programs which stepped through arrays of data (SOR and matrix), show a huge drop in miss ratio between 32K and 64K, as well as strange behavior in sizes 4K and 8K. This is due to the matrix nature of the programs, which do not use the cache very well until the matrices fit reasonably well into the cache. The data would not be re-referenced unless the data fit into the cache. Matrix has a 65 x 65 array of doubles, requiring 33,800 bytes to store it. SOR has a 55 x 55 array of doubles, which requires 24,200 bytes. SOR involves operations in column major order, and needs 55 rows to be able to fit into the cache to take advantage of temporal locality. For smaller caches, increasing the subsector size does not improve the miss ratio, because the data fetched typically is not used before the sector is evicted from the cache. At 32K the miss ratios for all sector sizes is essentially the same except for the larger sectors, which perform rather poorly. These two matrix programs tend to improve in sudden jumps, or not at all, giving rise to strange looking graphs (Figures 32 and 33).

Figures 36–45 show the instruction cache miss ratios of the various sub-workloads used in the analyses in this paper. The workloads are broken into roughly equal pieces of programs that generally group together. We can see from these plots that workloads 1, 2, and 5 are much more substantial than workloads 3 and 4, being able to take advantage of additional cache memory beyond 32K.

The miss ratios for the unified cache are plotted in Figures 46–55. The same data access patterns which distort the graphs in Figures 32 and 33 also show up in Figures 52 and 53.

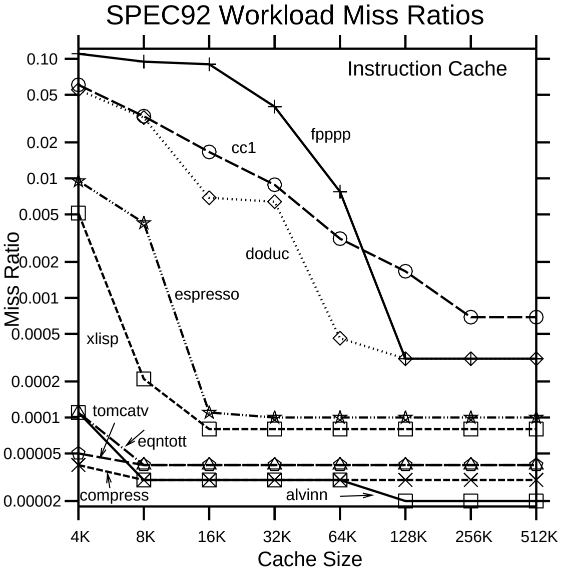


Figure 17: Instruction cache miss ratios for the SPEC92 workloads, 32–byte block size.

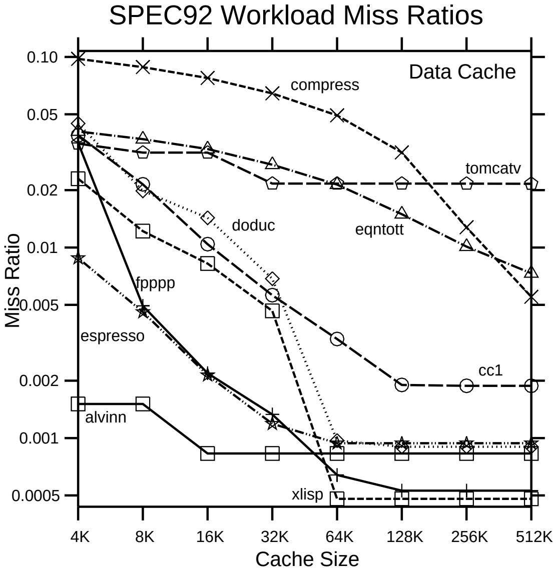


Figure 18: Data cache miss ratios for the SPEC92 workloads, 32–byte block size.

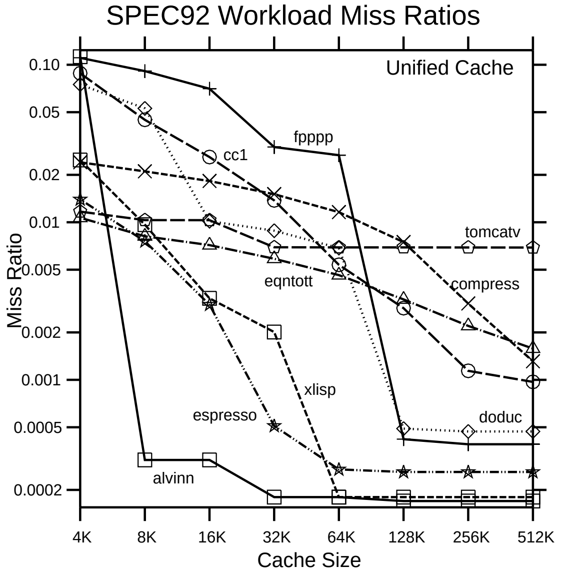


Figure 19: Unified cache miss ratios for the SPEC92 workloads, 32–byte block size.

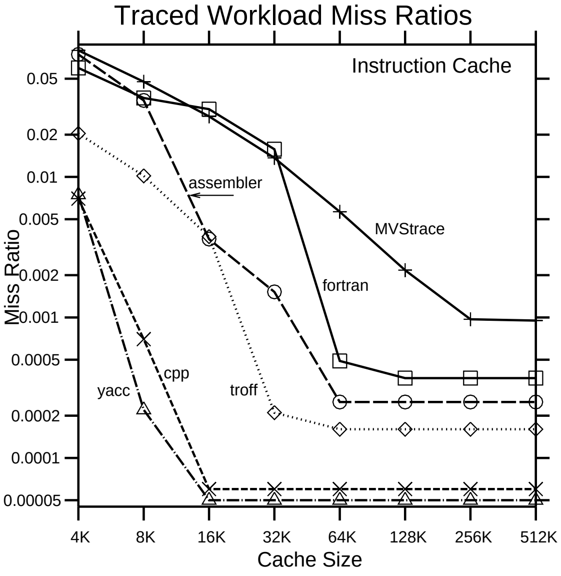


Figure 20: Instruction cache miss ratios for workloads derived from traces, 32–byte block size.

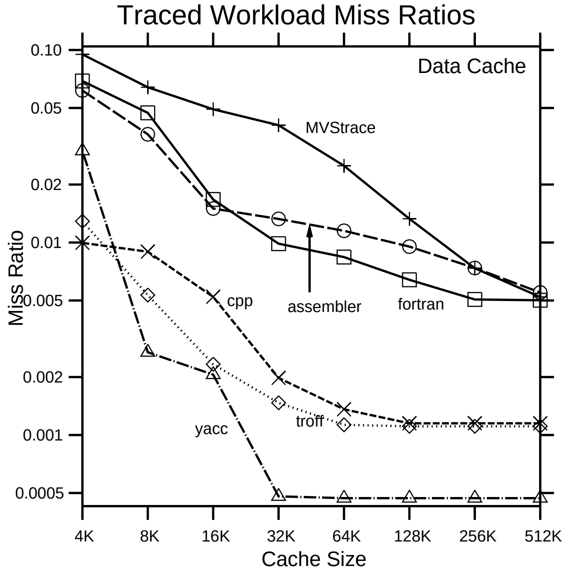


Figure 21: Data cache miss ratios for workloads derived from traces, 32–byte block size.

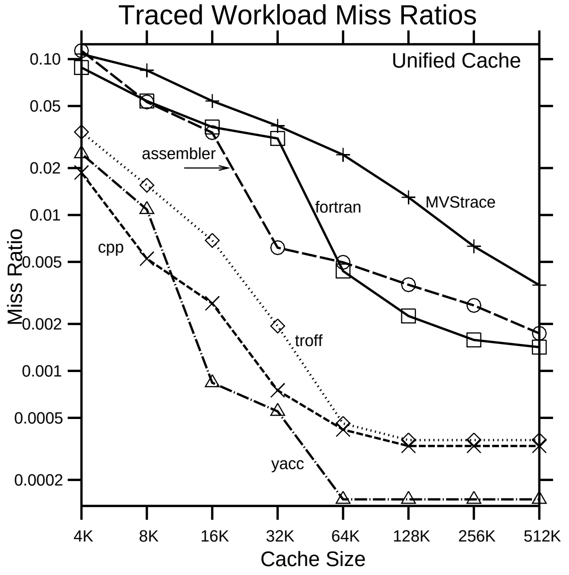


Figure 22: Unified cache miss ratios for the workloads derived from traces, 32–byte block size.

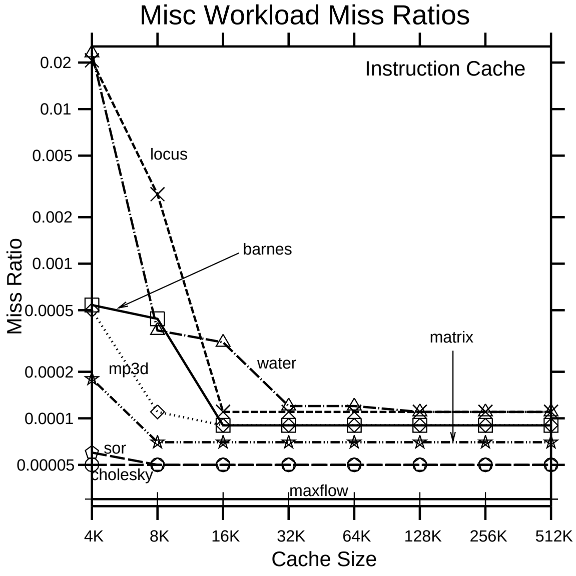


Figure 23: Instruction cache miss ratios for the SPLASH workload, 32–byte block size.

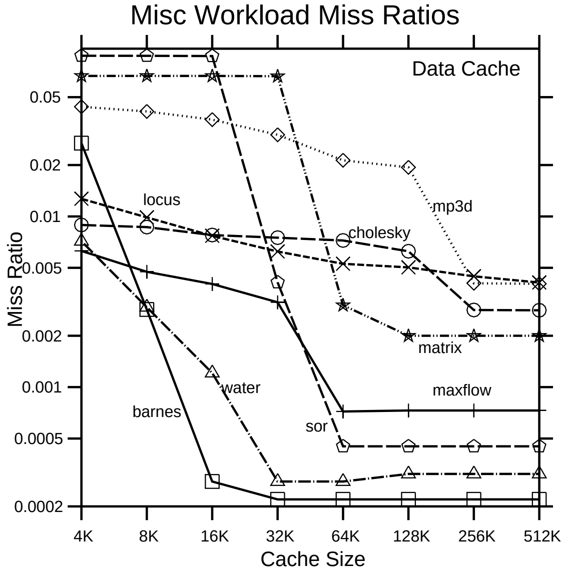


Figure 24: Data cache miss ratios for the SPLASH workload, 32–byte block size.

Miss Ratios of Individual Workloads with 16 Byte Blocks									
Workloads		**Cache Size** (KBytes)							
		4	8	16	32	64	128	256	512
barnes	inst	0.000880	0.000856	0.000169	0.000167	0.000167	0.000167	0.000167	0.000167
	data	0.035797	0.003668	0.000503	0.000426	0.000426	0.000426	0.000426	0.000426
cholesky	inst	0.000102	0.000097	0.000094	0.000094	0.000094	0.000094	0.000094	0.000094
	data	0.017379	0.017304	0.015518	0.015172	0.014903	0.012085	0.005594	0.005578
locus	inst	0.018694	0.004797	0.000193	0.000193	0.000193	0.000193	0.000193	0.000193
	data	0.019309	0.015378	0.011865	0.009814	0.008198	0.007505	0.006481	0.006374
mp3d	inst	0.000510	0.000202	0.000174	0.000173	0.000173	0.000173	0.000173	0.000173
	data	0.078257	0.074428	0.068169	0.057351	0.043272	0.036158	0.007208	0.007144
water	inst	0.042235	0.000743	0.000594	0.000215	0.000215	0.000215	0.000215	0.000215
	data	0.013299	0.005944	0.002552	0.000561	0.000561	0.000561	0.000561	0.000561
matrix	inst	0.000167	0.000120	0.000120	0.000120	0.000120	0.000120	0.000120	0.000120
	data	0.129119	0.129106	0.129028	0.128507	0.005823	0.003861	0.003859	0.003859
maxflow	inst	0.000050	0.000050	0.000050	0.000050	0.000050	0.000050	0.000050	0.000050
	data	0.009243	0.008863	0.007470	0.005350	0.001338	0.001338	0.001338	0.001338
sor	inst	0.000100	0.000088	0.000088	0.000088	0.000088	0.000088	0.000088	0.000088
	data	0.168515	0.168406	0.167823	0.007861	0.000877	0.000877	0.000877	0.000877
alvinn	inst	0.000044	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043	0.000043
	data	0.002942	0.002942	0.001588	0.001587	0.001587	0.001587	0.001587	0.001587
doduc	inst	0.102843	0.061476	0.013194	0.012288	0.000876	0.000596	0.000596	0.000596
	data	0.038600	0.030499	0.022745	0.002331	0.001962	0.001700	0.001700	0.001700
fpppp	inst	0.217355	0.187229	0.178766	0.079325	0.010795	0.000607	0.000607	0.000607
	data	0.031030	0.004447	0.002946	0.001845	0.001117	0.001006	0.001006	0.001006
tomcatv	inst	0.000086	0.000064	0.000064	0.000064	0.000064	0.000064	0.000064	0.000064
	data	0.069721	0.062686	0.062647	0.043094	0.043090	0.043081	0.043064	0.043030
cc1	inst	0.078809	0.045899	0.024318	0.011103	0.004722	0.002344	0.001244	0.001244
	data	0.041180	0.026345	0.014394	0.009290	0.005697	0.003561	0.003540	0.003540
compress	inst	0.000065	0.000061	0.000061	0.000061	0.000061	0.000061	0.000061	0.000061
	data	0.097531	0.088284	0.077386	0.065388	0.051291	0.034777	0.016130	0.010785
eqntott	inst	0.000109	0.000074	0.000074	0.000074	0.000074	0.000074	0.000074	0.000074
	data	0.061804	0.056848	0.051103	0.043791	0.035188	0.025626	0.019041	0.014296
espresso	inst	0.014938	0.007485	0.000187	0.000184	0.000184	0.000184	0.000184	0.000184
	data	0.014765	0.008331	0.003954	0.002159	0.001790	0.001790	0.001790	0.001790
xlisp	inst	0.007909	0.000162	0.000148	0.000140	0.000140	0.000140	0.000140	0.000140
	data	0.034660	0.019326	0.015735	0.008978	0.000912	0.000912	0.000912	0.000912
as1	inst	0.087669	0.050102	0.003530	0.002150	0.000458	0.000456	0.000456	0.000456
	data	0.106615	0.040030	0.027989	0.024913	0.022277	0.018388	0.014483	0.010891
cpp	inst	0.008586	0.000686	0.000114	0.000114	0.000114	0.000114	0.000114	0.000114
	data	0.017775	0.016292	0.008407	0.003314	0.002352	0.002069	0.002069	0.002069
fortran	inst	0.084499	0.056420	0.048730	0.008145	0.000721	0.000662	0.000662	0.000662
	data	0.079936	0.049418	0.021782	0.016998	0.014628	0.010863	0.009771	0.009714
troff	inst	0.031256	0.014325	0.005347	0.000300	0.000289	0.000289	0.000289	0.000289
	data	0.018224	0.007748	0.003852	0.002514	0.002092	0.002064	0.002064	0.002064
yacc	inst	0.011323	0.000278	0.000093	0.000093	0.000093	0.000093	0.000093	0.000093
	data	0.038315	0.004735	0.004024	0.000893	0.000890	0.000890	0.000890	0.000890
ostrace	inst	0.124766	0.070901	0.039585	0.019658	0.007956	0.002925	0.001703	0.001695
	data	0.110717	0.077476	0.063248	0.045293	0.028708	0.018312	0.012521	0.009712
Multi– prog.	inst	0.039492	0.024411	0.015899	0.007917	0.004086	0.002061	0.001052	0.000502
	data	0.060618	0.044748	0.039406	0.027261	0.020232	0.017004	0.014244	0.011605

Table 27: Miss ratio characteristics of individual workloads for instruction and data streams.

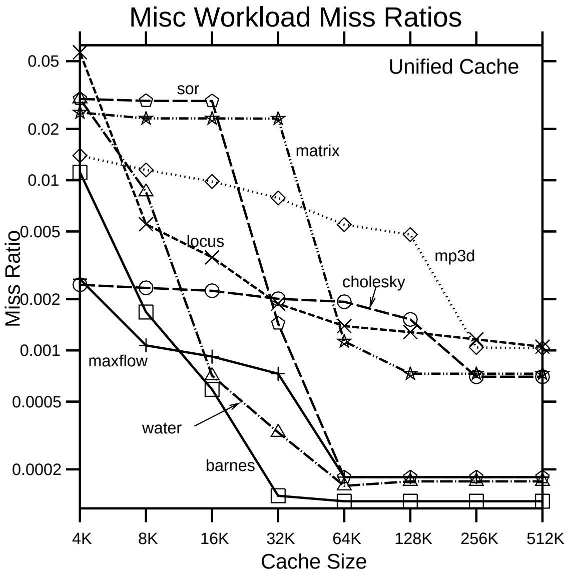


Figure 25: Unified cache miss ratios for the SPLASH workload, 32–byte block size.

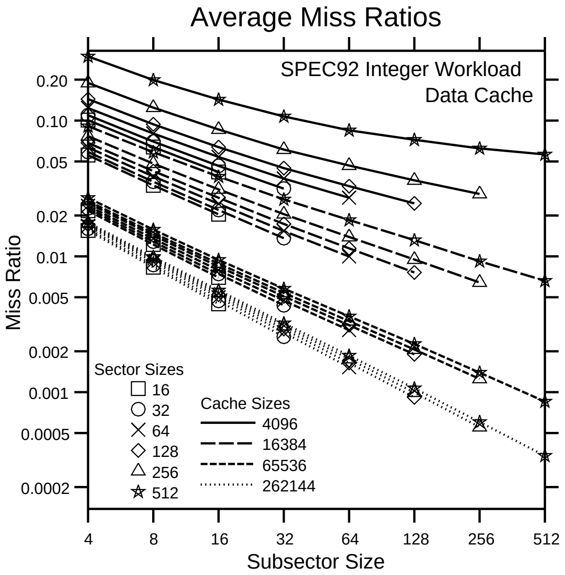


Figure 26: Data cache miss ratios for workload 1

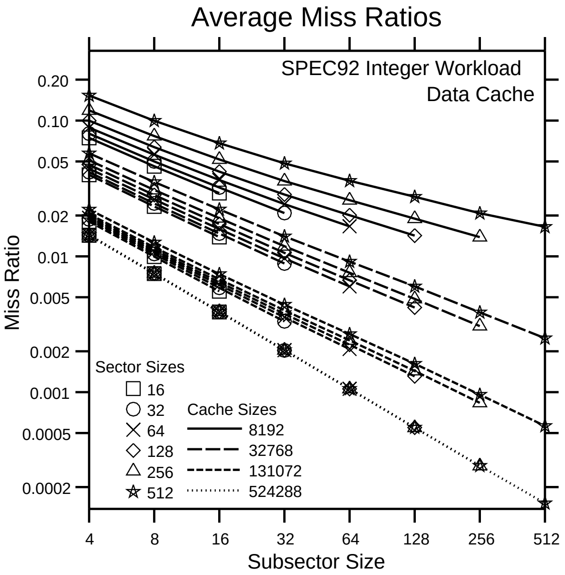


Figure 27: Data cache miss ratios for workload 1

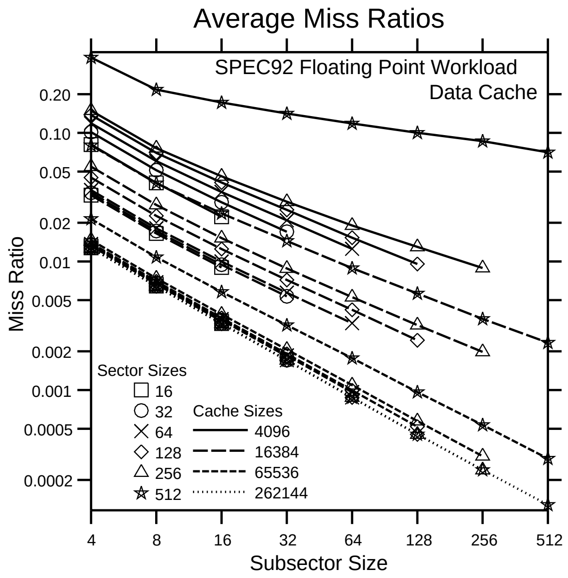


Figure 28: Data cache miss ratios for workload 2.

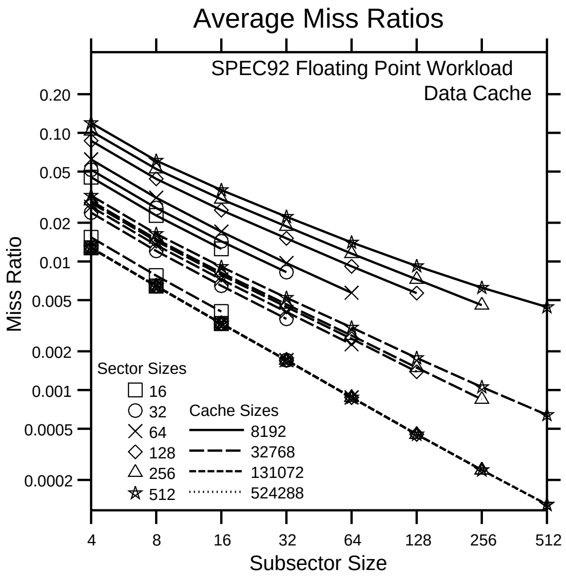


Figure 29: Data cache miss ratios for workload 2.

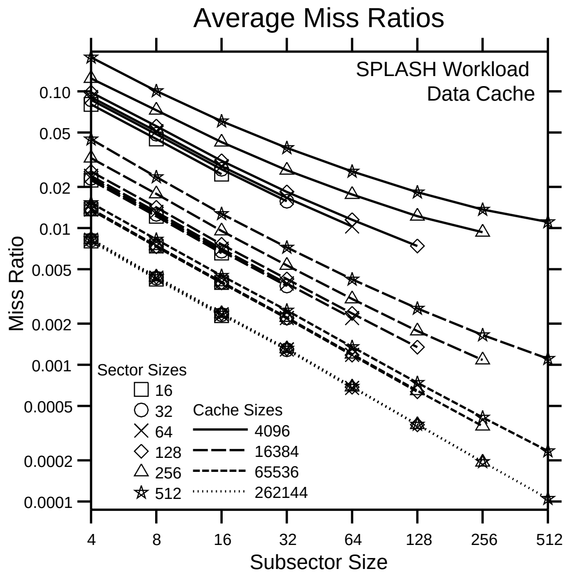


Figure 30: Data cache miss ratios for workload 3.

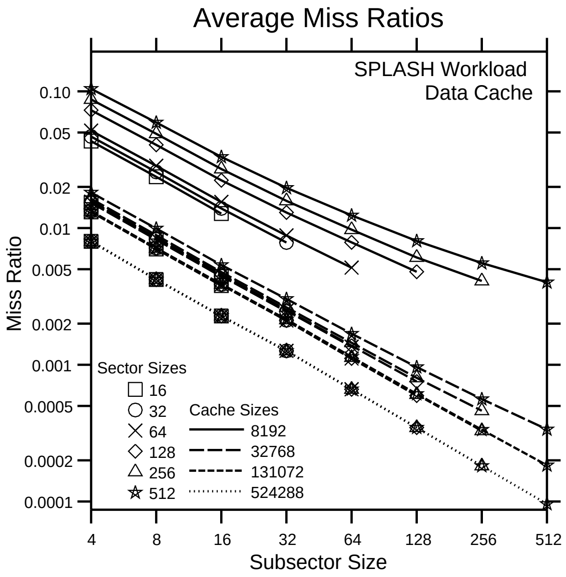


Figure 31: Data cache miss ratios for workload 3.

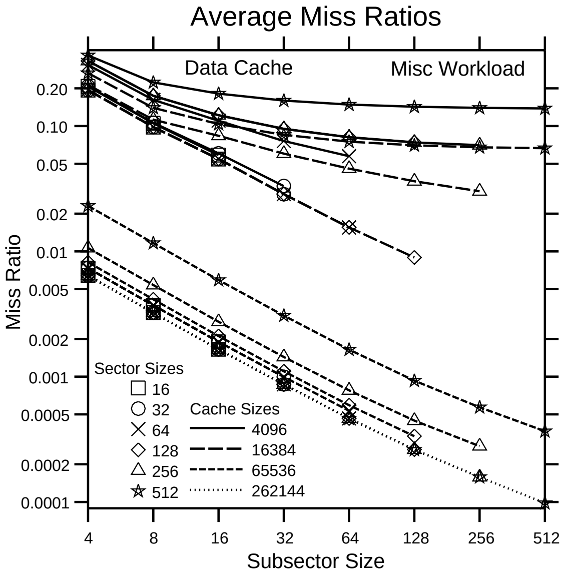


Figure 32: Data cache miss ratios for workload 4.

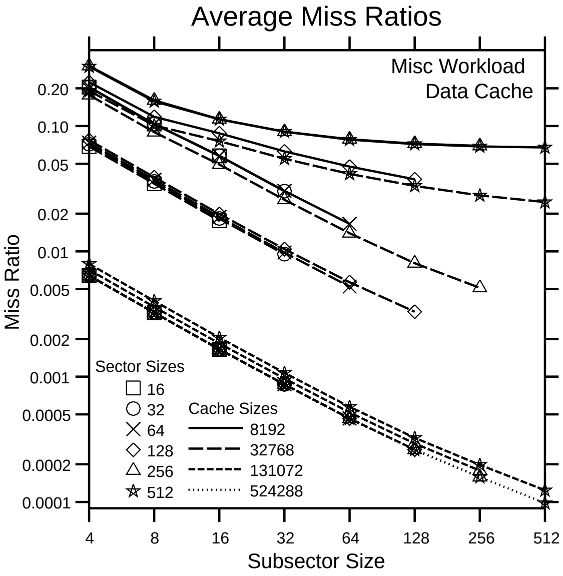


Figure 33: Data cache miss ratios for workload 4.

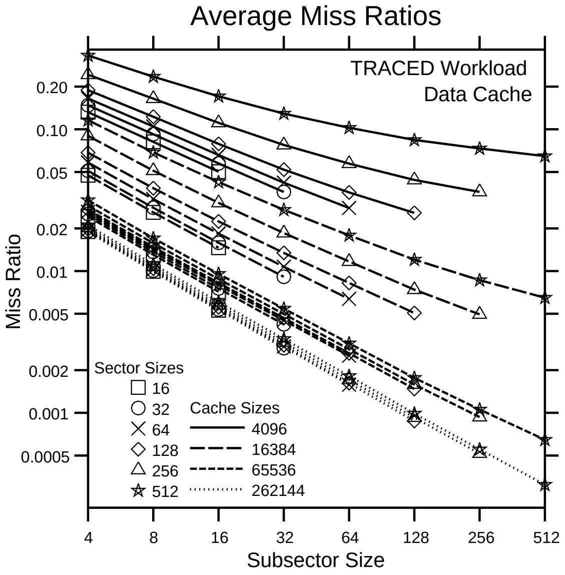


Figure 34: Data cache miss ratios for workload 5.

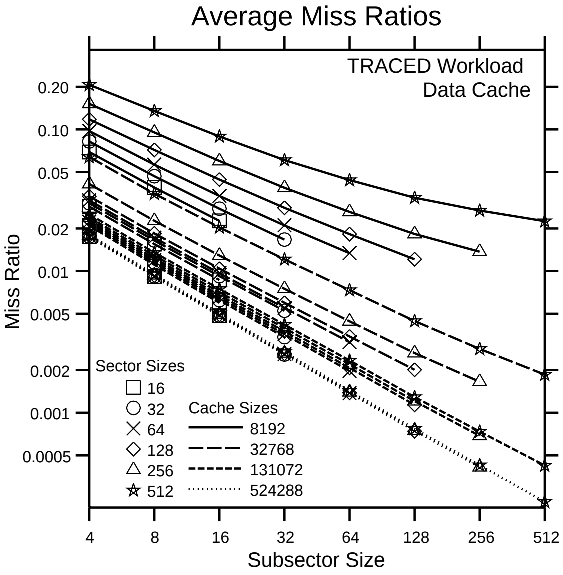


Figure 35: Data cache miss ratios for workload 5.

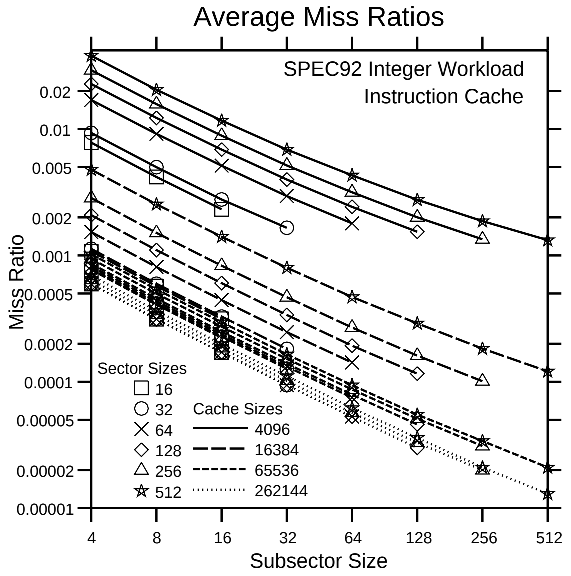


Figure 36: Instruction cache miss ratios for workload 1.

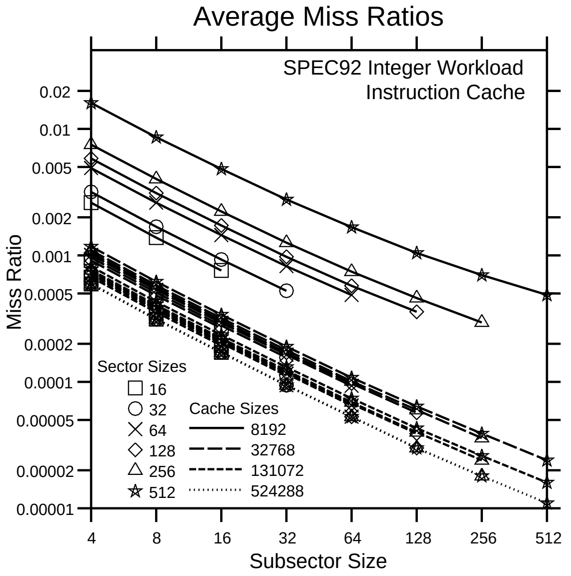


Figure 37: Instruction cache miss ratios for workload 1.

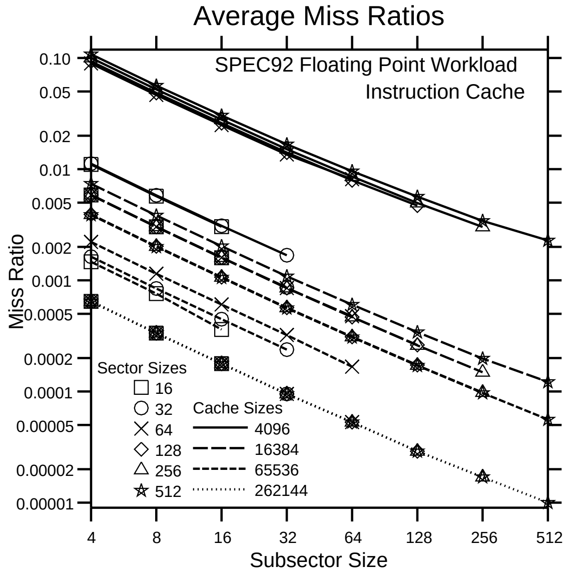


Figure 38: Instruction cache miss ratios for workload 2.

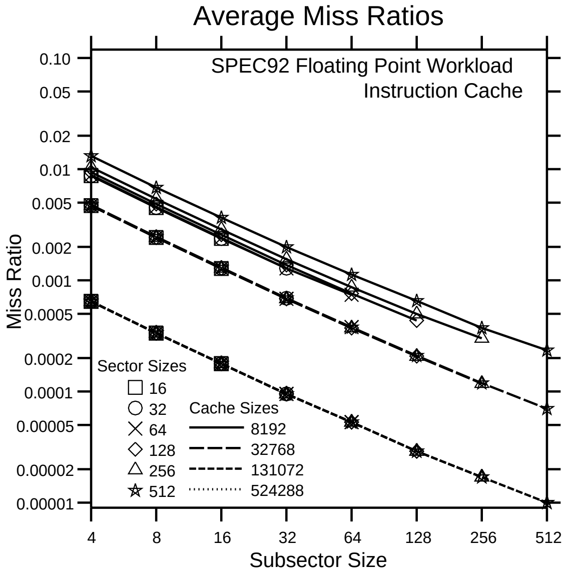


Figure 39: Instruction cache miss ratios for workload 2.

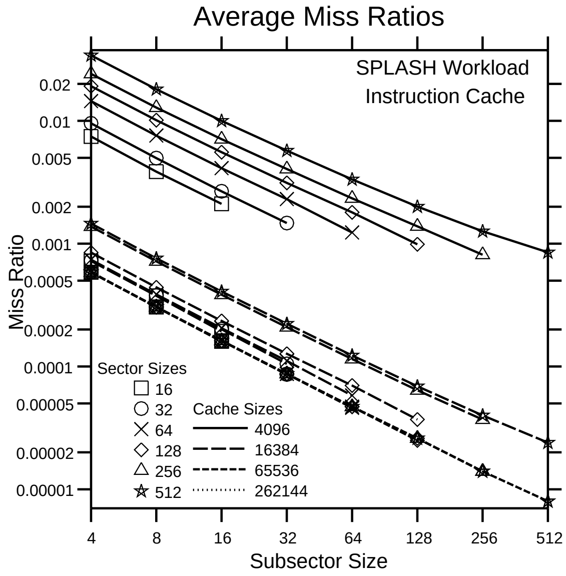


Figure 40: Instruction cache miss ratios for workload 3.

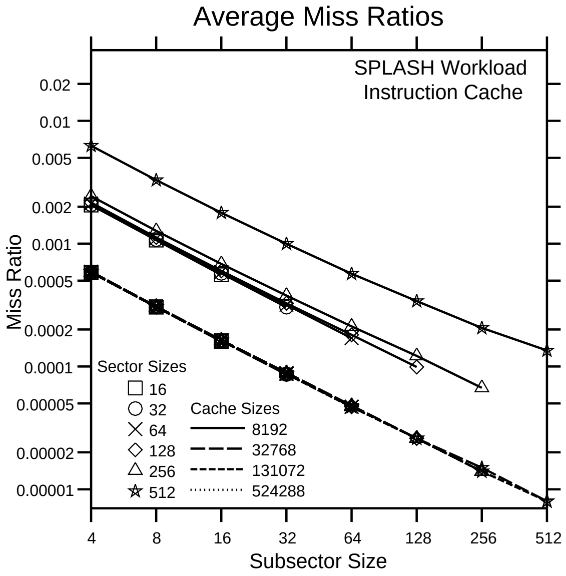


Figure 41: Instruction cache miss ratios for workload 3.

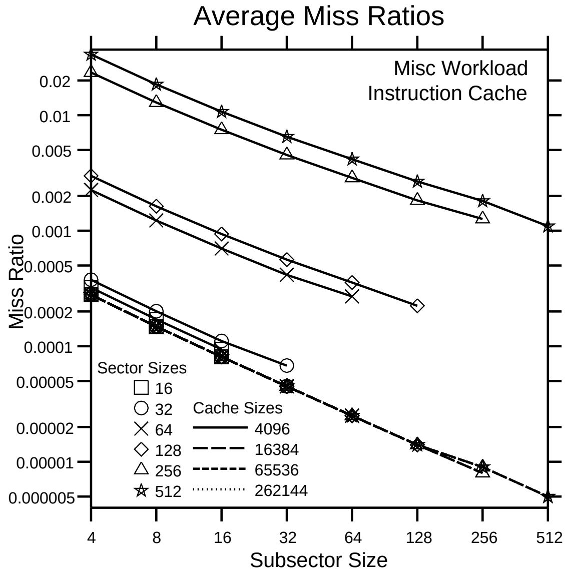


Figure 42: Instruction cache miss ratios for workload 4.

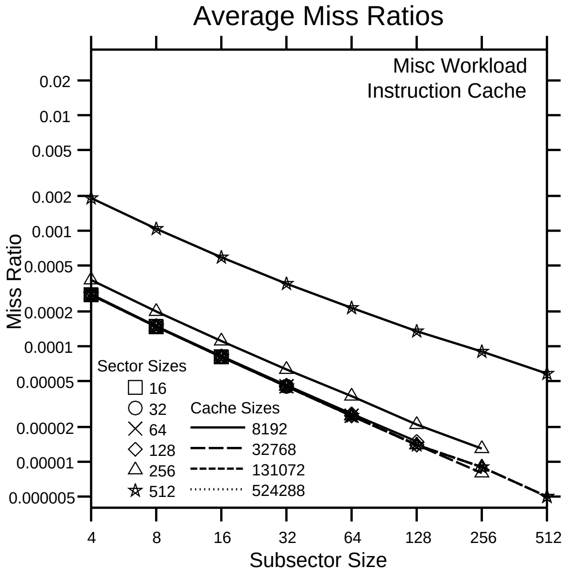


Figure 43: Instruction cache miss ratios for workload 4.

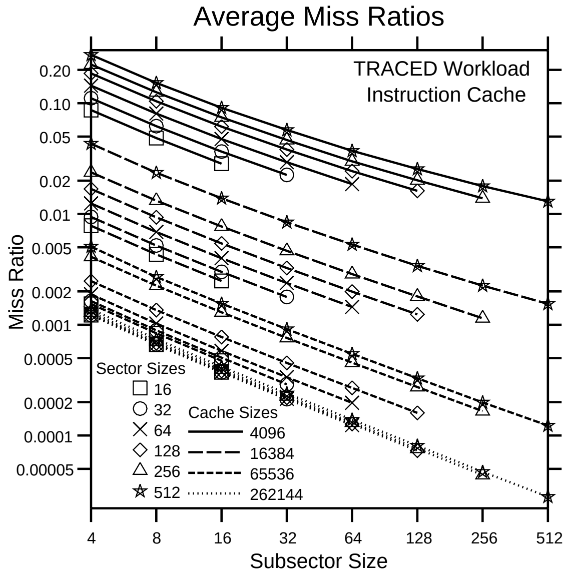


Figure 44: Instruction cache miss ratios for workload 5.

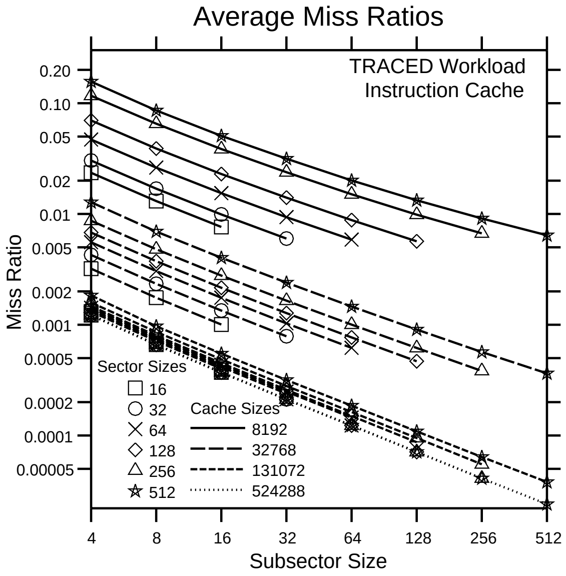


Figure 45: Instruction cache miss ratios for workload 5.

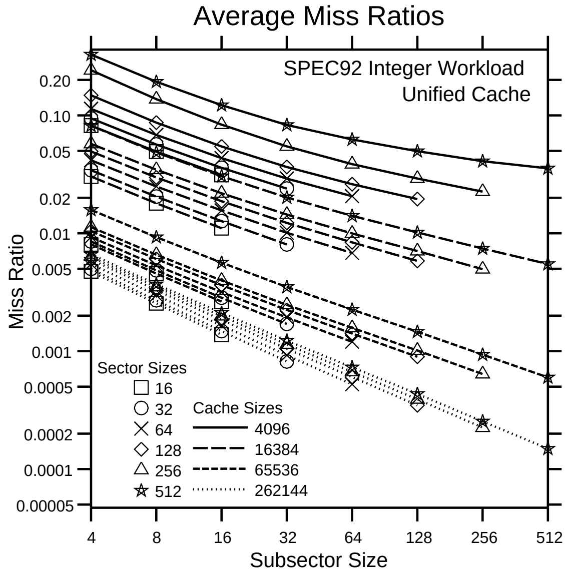


Figure 46: Unified cache miss ratios for workload 1.

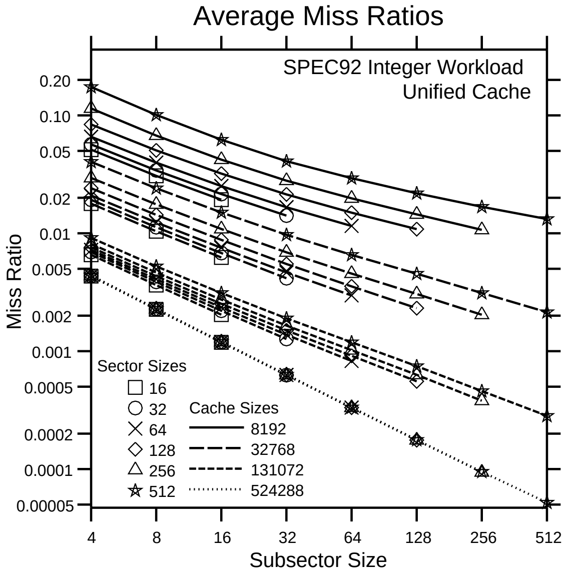


Figure 47: Unified cache miss ratios for workload 1.

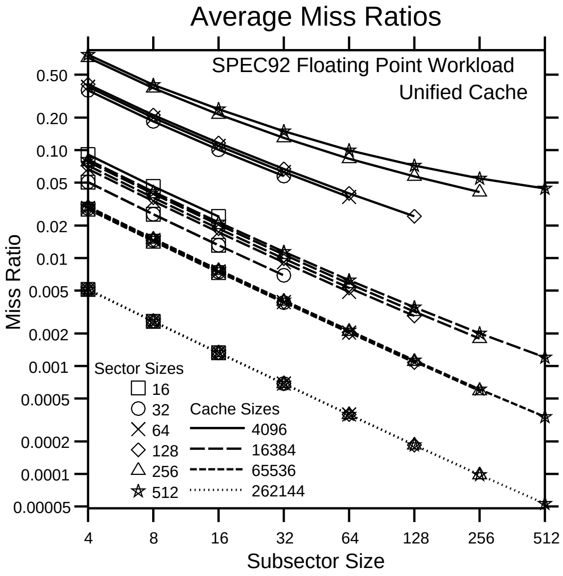


Figure 48: Unified cache miss ratios for workload 2.

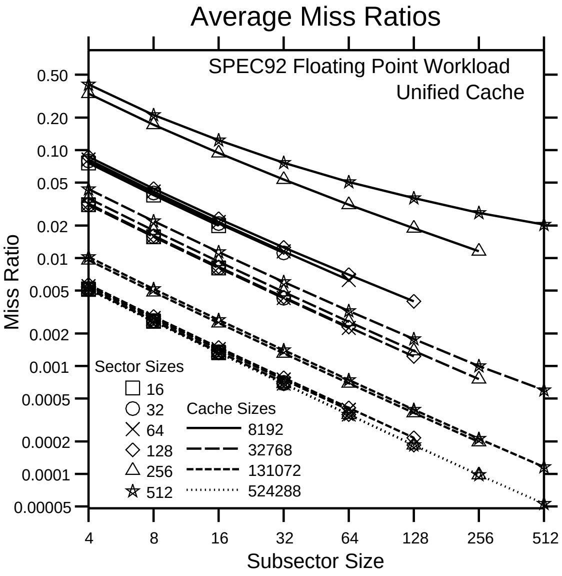


Figure 49: Unified cache miss ratios for workload 2.

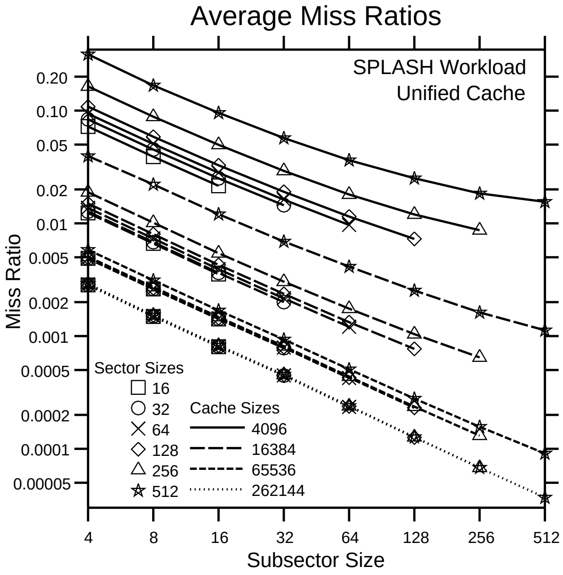


Figure 50: Unified cache miss ratios for workload 3.

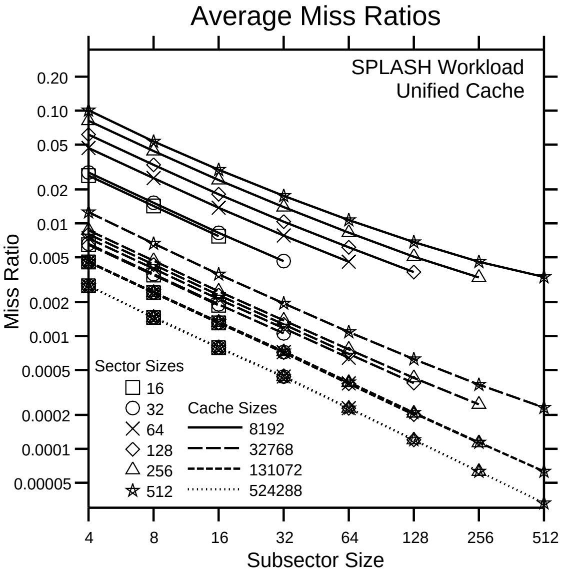


Figure 51: Unified cache miss ratios for workload 3.

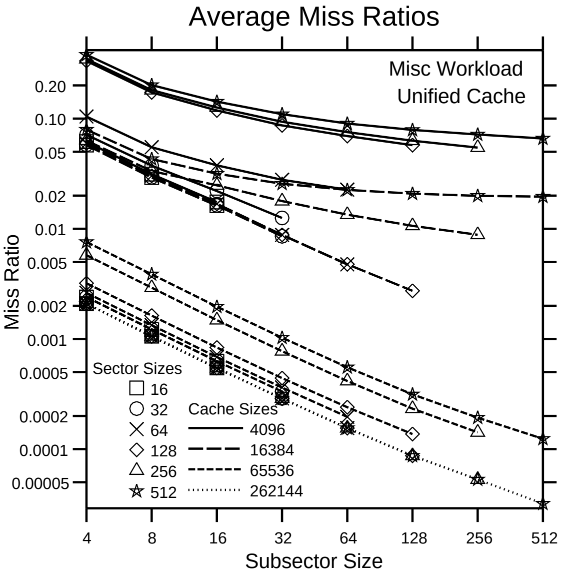


Figure 52: Unified cache miss ratios for workload 4.

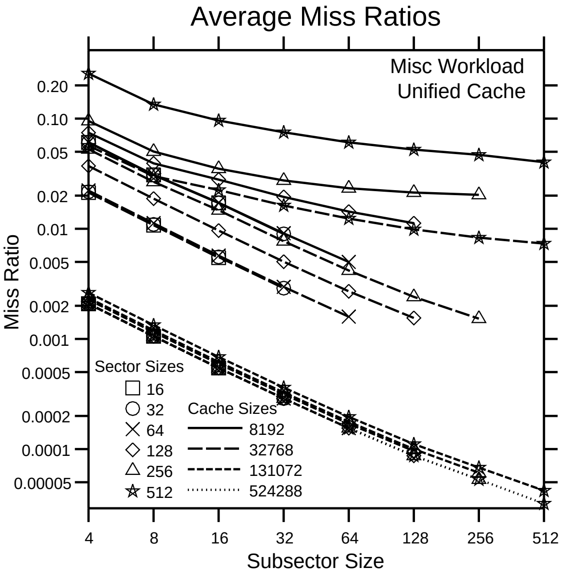


Figure 53: Unified cache miss ratios for workload 4.

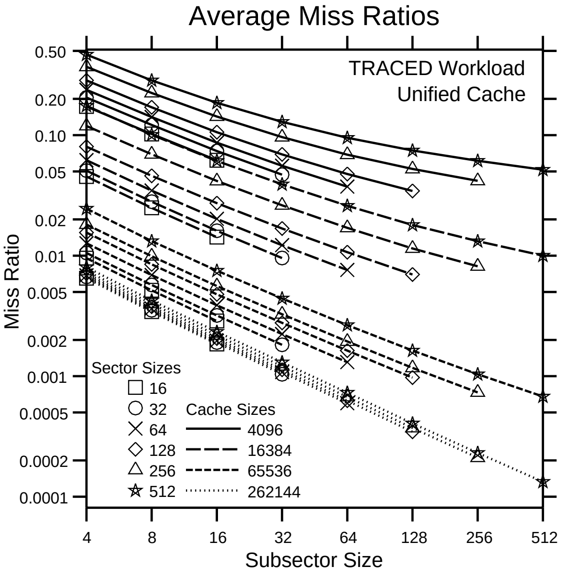


Figure 54: Unified cache miss ratios for workload 5.

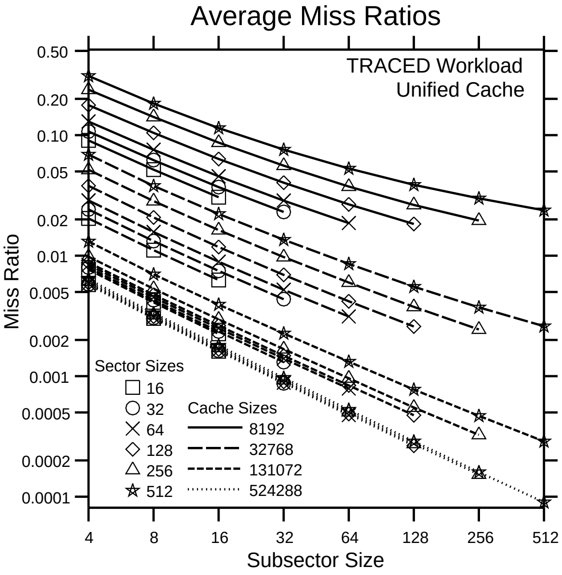


Figure 55: Unified cache miss ratios for workload 5.