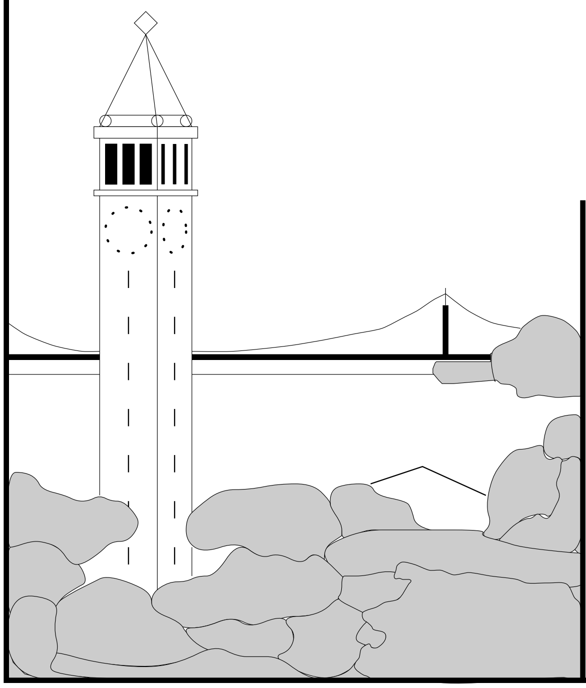
The Pool of Subsectors Cache Design

Jeffrey B. Rothman and Alan Jay Smith



Report No. UCB/CSD-99-1035

January 1999

Computer Science Division (EECS)

University of California

Berkeley, California 94720

The Pool of Subsectors Cache Design

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

January 1999

Abstract

Sector caches use an address tag to identify a sector, and valid bits to indicate whether each subsector is present in the cache. This design permits a small number of tag bits to control a large number of data bytes in the cache. Such a design is useful for single level caches when cache sizes are small and/or when the optimal line sizes are small. Sector caches are most useful when small first-level tag arrays are used to control large second level cache data arrays. The problem with sector caches is that because of the constrained mapping between address tags and data array space, frequently many of the subsectors are not filled before the sector is replaced.

In this paper, we propose a new design, called the **sector pool cache**, in which subsectors may be shared between sectors, so that the cache data arrays are used much more efficiently. We evaluate this design using a large multiprogrammed workload to provide a more realistic test of the various cache designs. Our results show that the sector pool cache can be an attractive solution for a first level on–chip cache when used in conjunction with a regular second level off–chip sector cache. Such a design provides improved performance by allowing a larger fraction of first level cache bits to be used for data rather than for tags.

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

1 Introduction

On-chip caches are using an increasing portion of die space on recent processors. Simultaneously there has been a disproportionate increase in the overhead of tag bits used to manage the buffer space in the cache due to the increase in the address space [WSY95, Sez94, Sez97]. One approach to minimizing the ratio of tag to data bits is to use a *sector* cache [Lip68, HS84, Prz90, Sez94, Sez97, RS99].

A cache's function is to keep frequently accessed portions of main memory close to the processor in a high speed buffer in order to reduce the delay of accessing memory. When a reference to memory does not find a copy in the cache (*cache miss*), a number of bytes are fetched from memory in a group, which can be referred to as a *cache line* or *block*. Address tags are associated with each cache line to keep track of the original location of the line in main memory. Caches are successful due to the observed property of memory reference locality, which consists of two forms. Temporal locality is observed when the same location is accessed repeated, such as instructions in a loop. Spatial locality occurs when words neighboring a recently accessed memory location are also referenced. Data arrays and the stream of instruction references in general show this property.

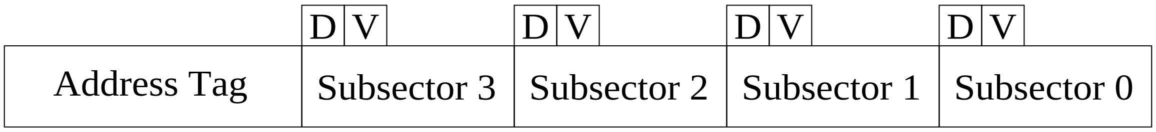


Figure 1: Symbolic view of a sector frame in a unified or data sector cache. Several subsectors are associated with one address tag. Each subsector has a dirty bit (D) and a valid bit (V). Each subsector can be independently present (V bit is on), invalid (V bit off), dirty (D bit on), or clean (D bit off).

Sector caches, which were the first commercial form of cache, were introduced with the IBM System/360 Model 85 [Lip68]. Sector caches differ from normal caches by allowing a number of cache lines (subsectors) to be associated with each address tag (Figure 1). As address tag sizes increase, sharing a single address tag among several subsectors can permit a significant savings in cache management bits. However, this savings comes at a cost of reducing the flexibility of mapping subsectors into the cache, generally causing an increase in the miss ratio. Each sector (address tag) has a power of two number of subsectors associated with it at a fixed offset within the sector frame (cache space set aside for all the subsectors belonging to a sector). Each subsector can be independently present or absent while its sector is in the cache.

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." When appropriate, we call a "sector" a "sector frame." The subsector is also called the "fetch size."

Various studies have shown sector caches to be useful. Sector caches can be used to reduce bus traffic with only a small increase in miss ratio, by using small subsectors [Goo83]. In some cases, particularly when the cache is small, the line size is small and/or the miss latency is small, sector caches improve performance in computers with single level caches [RS99, Prz90]; some recent processors use designs with two subsectors per sector [Moo93, TO96]. Sector caches more frequently yield benefits in two–level cache systems in which tags for the second level sector cache are placed at the first level, thus permitting a small tag store to control a very large cache [RS99].

The problem with sector caches is that the mapping between sectors and subsectors is constrained, with two negative results. First, in most cases a sector is replaced before all of its subsectors are occupied with data, causing the cache space to be used inefficiently. Second, when a sector is replaced, all of its subsectors must be replaced, even though only a single subsector is being fetched. For example, Hill [HS84] estimated that 72% of the data in the cache lines had not been touched, [RS99] found that the number of subsectors per sector not accessed while in the cache ranged between 6 to over 90 percent, depending on sector and cache size (reproduced in Table 1). Table 1 shows the fraction of subsectors accessed between the time a sector is read into the cache and the time it is evicted from the cache, demonstrating that for many cache configurations, a significant number of (four byte) words or subsectors are not touched while a sector resides in the cache. Traditional sector caches are partially able to take advantage of this property by not loading portions of a sector which are not demanded, saving memory bus bandwidth. However, the sector cache still under–utilizes its data buffer space, since each subsector space in the cache is reserved for exclusive use by its corresponding sector frame. What we propose in this paper is a more

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

dynamic form of sector cache, which allows the sector frames in each set to compete for subsectors. Less data space is used by the cache, which provides similar performance with less memory allocated to the data storage arrays.

1.1 Sector Pool Cache Organization

In the sector pool organization, subsectors are associated with a set of sectors, rather than with individual sectors (Figure 2). Assume an N–way set–associative sector cache, i.e., the sectors are organized in a set–associative manner, and then each sector consists of subsectors. In a "normal" design, each sector would contain K subsectors, and we could diagram the set as N rows, consisting of an address tag (denoting the sector) and K subsectors (with valid bits). The set would thus contain N*K subsectors. Since we've observed that many subsectors never get used, in the sector pool design each column of subsectors contains fewer than N elements, and the subsectors are dynamically assigned to the sectors as necessary. The hope is that we can save a significant amount of the data space without

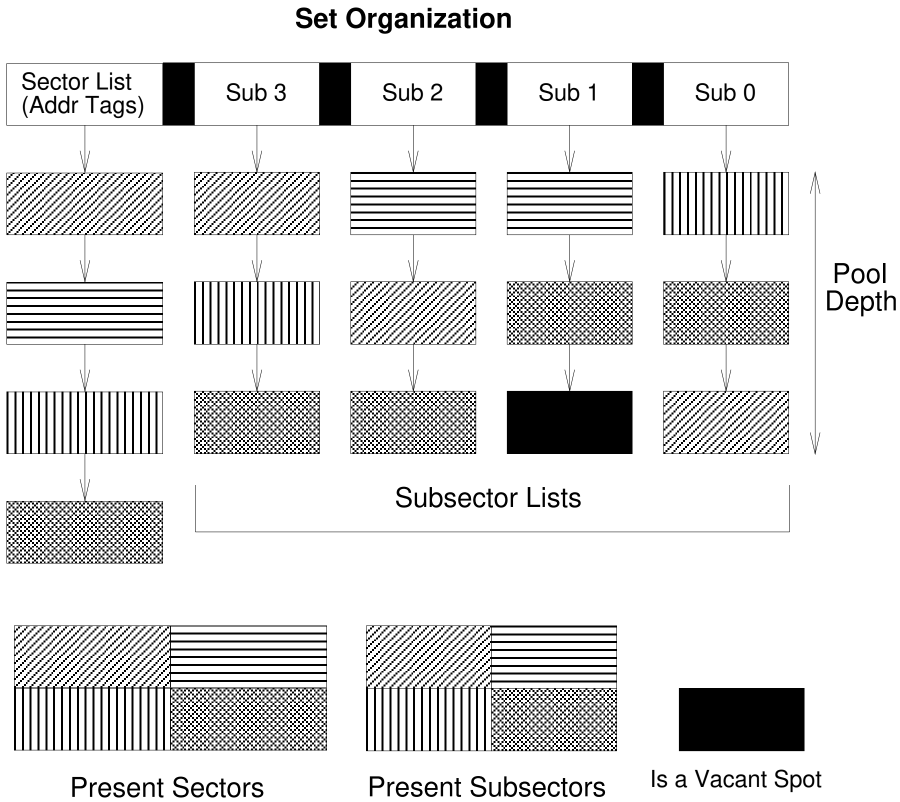


Figure 2: Diagram of a sector pool cache. Regions with the same colored shading are associated with the same sector frame.

significantly increasing the miss ratio. An example sector pool design (Figure 2) uses a 4–way set–associative configuration with 4 subsectors per sector and pool depth 3. There are 3 assignable subsectors for each subsector position [0..3]. This leads to a total of 12 assignable subsectors for the whole set, rather than the 16 subsectors that required by a traditional sector cache.

The implementation of the sector pool cache is straightforward. Figure 3 shows the subsector list pointer bits associated with a single sector address tag. This example contains 8 sectors in the set (one sector is shown) which must compete among themselves for 5 subsectors at each offset. Each offset within the sector frame contains pointer bits to a position within the list of subsectors associated with a set. A traditional 8–way set–associative sector cache effectively contains 8 subsectors per list; this example sector pool cache uses 5 subsectors in each list. Six valid values are maintained by the pointer bits: 0 for an invalid subsector, and 1 through 5 which are indices into the corresponding subsector list.

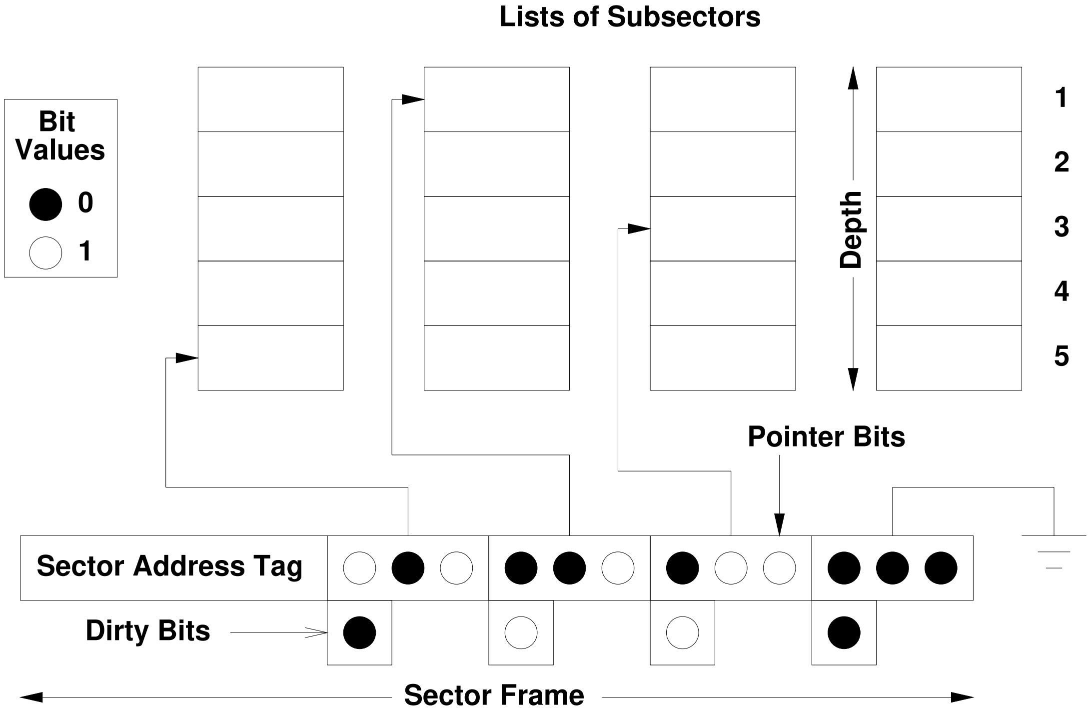


Figure 3: Schematic showing the pointer bits associated with a sector address tag for a sector pool cache with depth 5.

1.2 Terminology

The following is a list of terms used to describe sector pool cache configurations:

- **Degree of Sectoring**: number of subsectors per sector.
- **List**: the number of subsectors a set has reserved for each possible subsector position in a sector frame. If the degree of sectoring is four, there are four distinct lists per set, corresponding to the four subsectors positions possible for each sector frame. Each list has the possible design range of $[1..N]$ subsectors available for use, where N is the set–associativity of the cache.
- **Pool**: the lists of subsectors associated with a set.
- **Depth**: the fixed number of subsectors reserved by the set for each particular subsector offset. The depth quantifies the number of assignable subsectors available in a list for a particular configuration.

1.3 Related Work

Sector caches with a dynamic assignment of subsectors to sectors have been previously proposed. The purpose of these sector cache organizations was improve on traditional sector caches to reduce the number of address tag bits while minimizing the impact on miss ratio. This was achieved by increasing the flexibility of mapping subsectors to sectors.

In [Sez94, Sez97], an organization was proposed in which address tags (sectors) were grouped together to compete for subsectors. For each subsector position, only one sector out of N in the grouping can use that particular subsector offset at a time. Each subsector maintains a *selector tag*, which allows it to determine the sector to which it belongs. The grouping of sectors that share subsectors can be unrelated (direct–mapped) or belong to a set. The sectors and subsectors can have independent varying degrees of set–associativity.

In the simplest case, the subsectors are accessed just like the blocks in a direct–mapped cache using some of the upper address bits as an index into the cache. The sector address tags (for example, kept in pairs) are accessed using a smaller subset of the upper address bits. Associated with each subsector is a valid bit and a selector bit which points to one of the pair of sector address tags. Once the selection bit has been read and the sector address tag has been obtained, it is possible to determine whether the subsector access was a hit or a miss.

When compared to normal caches of the same associativity, the decoupled sector cache was reported to have worse behavior, with up to a 337 percent increase in the miss ratio, although with the increase falling mostly in the 20 to 80 percent range for the larger workloads. Increasing the set–associativity for the sectors and (independently) the subsectors was shown to improve the performance of this type of organization. For large caches (256K and 1M bytes), the miss ratio (using 32 byte subsectors) showed reasonable performance with some workloads, while reducing the number of tag bits up to 86 percent. However, some of the more robust workloads (such as doduc and gcc) showed much worse miss ratios (2 to 11 times higher) for certain configurations. The best overall performance generally occurred when the set–associativity of the tag array was close to the number of sector tags competing for each subsector. Invalidations appear to be somewhat complicated in this organization, due to the necessity of querying all the subsectors (up to 64 in some cases) in a region of the cache to determine which ones to evict when a sector is replaced.

An even more dynamic organization was described and evaluated in [WSY95, WSY97]. The authors noted that many address tags in a cache contain the same values, allowing the possibility of reducing tag space by creating a small address tag array (sector tags), with each cache line maintaining a pointer into the sector tag array. Like a sector cache, when an address tag is removed from the tag array, all associated cache lines (which behave like subsectors) must also be removed from the cache. What makes it different from a normal sector cache is that an arbitrary number of subsectors can be associated with each sector. The results show that for a 16K byte cache using 16 byte lines, a tag array with 32 entries has similar performance to a normal cache,but with a substantial savings in tag space. However, one important issue with this type of organization is the associative search required to remove subsectors on sector replacement.

2 Methodology

This research used trace driven simulation to derive miss ratio results for each point in the design space. We examined a large range of cache parameters, varying the cache size, the number of subsectors per sector, the size of subsectors, and the subsector pool depth associated with each set. To properly evaluate our design, we use a form of memory delay as well as the total number of bits required by each cache organization to measure the cost and benefit.

2.1 Traces

Our cache simulations used 24 separate traces, combined in a multiprogramming workload, as described below. A summary of each of the trace characteristics can be found in Table 2. Five of the workload traces come from the Stanford Splash applications [SWG92], which are scientific or engineering programs targeted for parallel computers. The programs are **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.

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 2: Description and characteristics of the input workloads.

The Berkeley workloads are small parallel programs (run with a single processor), consisting of test cases for particular algorithms. These applications are **matrix**, a matrix multiplication algorithm; **maxflow**, a calculation of the maximum flow possible through a graph; and **sor**, a solution to Laplace's equation using successive overrelaxation.

The SPEC92 Integer workloads include **cc1**, the gcc C compiler; **compress**, a file compression utility using an adaptive Lempel-Ziv algorithm; **eqntott**, which translates 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 were 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 measures the performance of a two electron integral derivative computation; and **tomcatv**, a vectorized

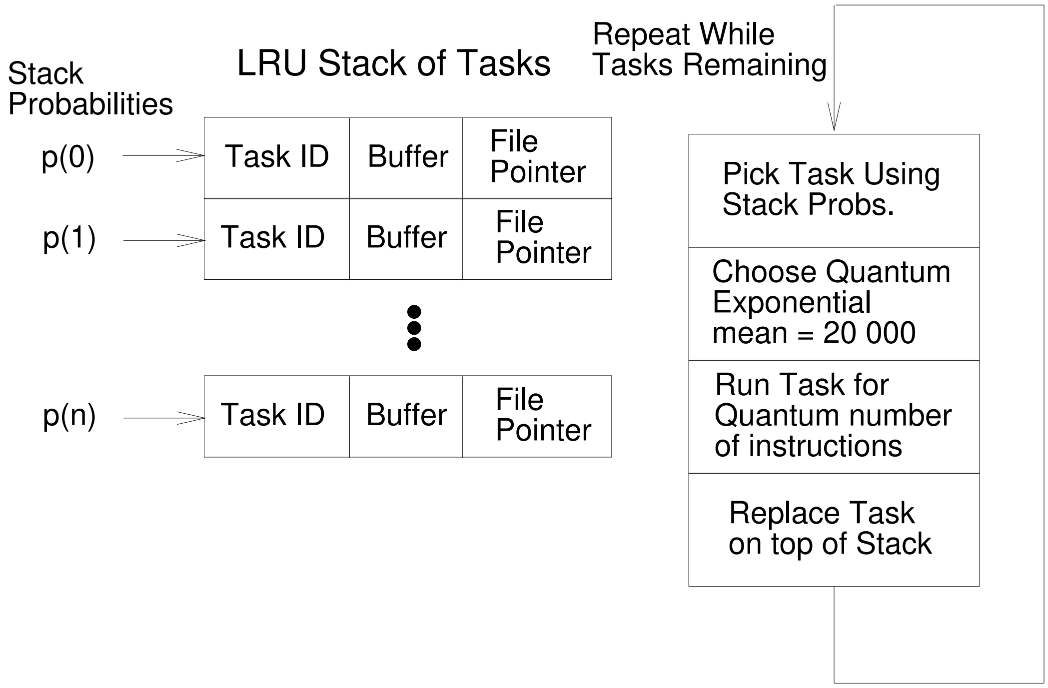


Figure 4: Creation of large multiprogrammed trace using a stack of workload tasks.

mesh generation program. All the codes except **alvinn** were written in FORTRAN; all use intensive Floating-Point math for calculations.

The UNIX group of applications consists of a few commonly used UNIX utilities: **cpp**, the pre-processor for the standard UNIX C compiler; **fortran**, a FORTRAN compiler; **troff**, a text typesetting program; and **yacc**, which generates an LR(1) parser from an input grammar specification.

The last trace **MVStrace** is a combination of user and supervisor memory references from an IBM/370 using the MVS operating system (a concatenation of traces MVS1 and MVS2 from [Smi85]). This trace was used to include operating system references to make the load more realistic.

2.2 Simulated Multiprogramming

An examination of Table 2 shows that in many cases the number of bytes accessed by these workloads is insufficient to stress a cache above 16K bytes (particularly instruction caches), let alone provide any meaningful results for the maximum size of caches under test (512K bytes). In addition, it was noted in [Smi85] that the usual method of exclusively utilizing user space references results in over-optimistic estimates of performance for several 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) the OS can be have particularly poor cache access characteristics and can adversely affect cache performance.

To address these issues, we combined all the traces into a single trace which simulates multiprogramming, using task switching probability information observed in real systems [Kob86]. Figure 4 details the process for generating the simulated multiprogramming trace. Our process is as follows: the LRU stack model is used to model process scheduling; the LRU stack consists of all of the processes in the system, ordered from most recently executed to least recently executed. At the time of a task switch, a new task is selected using the the stack program locality probabilities derived in [Kob86], "run" for a time quantum and then pushed back on the top of the stack. The time quanta have an exponential distribution with mean 20,000. "Running" a process means using sequential memory addresses from that process. Each address is tagged with the process ID, so the cache is not flushed on a task switch. This process is repeated until all the traces have been exhausted.

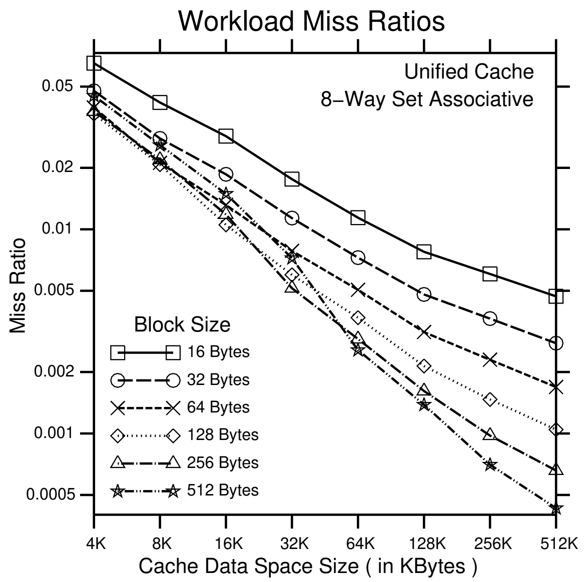


Figure 5: Miss ratios of the simulated multiprogrammed workload for a normal unified cache.

This multiprogrammed trace has two major advantages: it displays realistic multiprogramming behavior, and it references a large enough address space to fully exercise the caches under test. The last entry of Table 2 shows some of the characteristics of the multiprogrammed trace. This new trace references more than 9Mbytes of address space, which is sufficient to fully exercise any cache design under consideration in this paper.

Figure 5 shows the miss ratios for unified (combined instruction and data) normal (nonsectored) caches over a range of cache and block sizes. It demonstrates that the multiprogrammed workload succeeds in providing a reasonable set of references that have enough capacity misses to properly evaluate caches up to 512K bytes in size. The trend of improving miss ratios with cache size shows little sign of saturation for the largest caches under examination.

A comparison of these miss ratios shows that they fall within the range of values established by other research. The values here are lower than the design target miss ratios for 8–way set–associative caches generated in [HS89], generally about 50 percent of the value for unified caches, a little less than 50 percent for instruction caches, but very similar values for data miss ratios. Compared with the miss ratios for the SPEC92 workloads generated in [GHPS93], our instruction cache miss ratios are 50 to 100 percent higher, the data miss ratios are 30 to 50 percent lower, and the unified cache miss ratios have generally the same values.

2.3 Metrics

In this paper, we present two principal metrics for cache performance. Cache miss ratio is the normal metric, but in systems such as the ones we consider, with varying fetch sizes and multiple levels, not all misses are equal; some take much longer than others. Therefore, we also consider average memory access time (including that for cache misses).

3 Design Parameters and Resources

An important issue in evaluating the performance of any particular cache organization is the resources required to implement the design in hardware. Such a metric could be the number of transistors, the number of bits, the die area required to implement the design, etc. For simplicity, we use the total number of address tag and data bits. A more detailed evaluation would involve measuring the use of silicon for comparators, data lines, multiplexors, etc.; a study at that level is beyond the scope of this paper.

Calculating the number of data bits for a particular design is straight–forward. We also need to consider the number of bits needed for address tags, valid and dirty bits, replacement, and to point from sector tags to subsectors. Since we are focussing on future architectures, we assume 48-bit address tags, which we believe are reasonable for the newest generation of 64-bit machines.

3.1 Replacement Strategy

One influence on the performance of a cache as well as on the necessary bit resources is the choice of replacement policy. In the case of a sector pool cache, replacement is an issue both for the sectors and the subsectors in a set, rather than just the sectors (blocks) in normal caches. One possibility we investigated used LRU replacement independently for sectors and for each list of subsectors in a set, as well as a scheme which made subsector replacement dependent on the set's LRU information for the sector address tags. The results showed that both schemes had very similar performance, with the dependent LRU subsector replacement policy behaving very slightly better. Allowing the subsectors to use the set (sector) LRU information for subsector eviction requires fewer bits (no LRU bits for each list). The dependent subsector eviction works by determining which sector frame had any of its subsectors referenced the longest ago, (i.e., the LRU sector) and evicting the corresponding subsector within the pool needing replacement. This may take a few cycles to determine, but since write–back of a dirty subsector can be handled in the background while stalling for the fetch that caused the subsector eviction, it is not time–critical. Our calculation of bits for each type of organization uses the dependent subsector LRU design.

The trace driven simulations used a true LRU replacement policy for sectors in each set. Accurately implementing such a policy can be done theoretically with $\lceil log_2(n!) \rceil$ bits, and efficiently with $\frac{n*(n-1)}{2}$ bits, where n is the number of items to maintain in order. A good approximation to 8–way set–associativity is that used in the IBM 370/168–3, which requires only 10 bits per set. The eight sectors are divided up into pairs, using a bit to maintain LRU ordering for each pair. Then true LRU order is kept for the four pairs, which requires six bits. It is not believed that this algorithm has a significant effect on the miss ratio [Smi82]. We assumed 10 LRU bits per set for 8–way set–associative caches.

3.2 Set–Associativity and Pool Depth

Three possible set–associativities (2,4,8–way) were tested for the sector pool cache design. The results we present here are for 8–way, because of the range of sector pool designs that can be evaluated; 8 or larger degrees of associativity have been used in various mainframes, but are uncommon in microprocessor based systems. Various clever schemes exist to yield good performance despite the delays associated with high levels of associativity [PHS98].

Since subsectors are associated with the set instead of sectors in this design, pointers need be maintained to indicate where the subsector is found in the set's list of subsectors. This requires $\lceil log_2(depth+1)\rceil$ bits for each subsector in each sector frame; one value is used to indicate that the subsector is absent (not valid), the other values are used as a pointer to a location in the subsector list (Figure 3). When the cache is fully populated (i.e., the pool depth is equal to the set–associativity), only a single pointer bit is required (it becomes the *valid* bit), because the cache is a traditional sector cache and has fixed subsectors for each sector.

3.3 Tag Bit Calculations

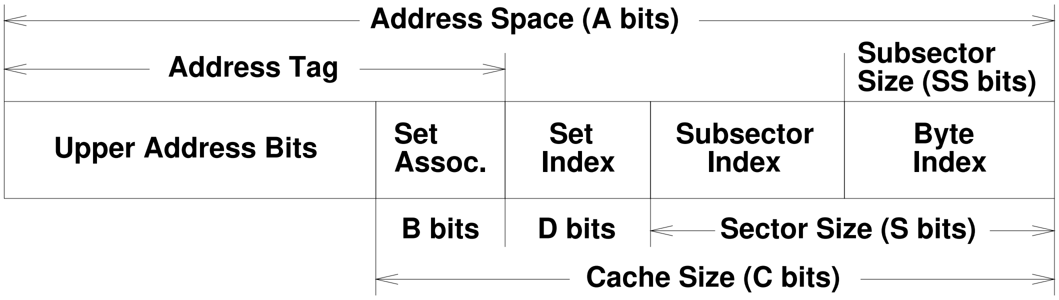


Figure 6: Breakdown of address bits into the address tag and indices relevant to cache information retrieval.

Figure 6 shows how the address is broken down into its component pieces. From the labeled quantities of address bits, we can calculate the number of total bits to implement a sector pool cache using the following formulas:

$$
\begin{aligned}
Bits_{Tag} &= A + B - C \\
Bits_{Ptr} &= \lceil log_2(Depth+1)\rceil \ (where\ 1 \leq Depth < 2^B) \\
&= 1 \ (where\ Depth = 2^B) \\
Bits_{Data} &= 8 * Depth * 2^{D+S} \\
Total\,Bits &= 2^D * Bits_{LRU} + 2^{C-S} * Bits_{Tag} + 2^{C-SS} * Bits_{Ptr} + Depth * 2^{D+S-SS} * Bits_{Dirty} + Bits_{Data}
\end{aligned}
$$

where $Bits_{LRU}$ is the number of bits to implement an LRU strategy (10 per set for 8–Way Set–Associativity), $Bits_{Dirty}$ is 0 for instruction or 1 for data and unified caches. In the case of a regular sector cache, $Bits_{Ptr}$ is equal to 1 and represents the valid bit.

3.4 Sector and Subsector Eviction

Each sector maintains pointers to its valid subsectors, so it is relatively easy to determine which subsectors in a set need to be evicted when a sector is replaced. The most similar previous work using a dynamic subsector mapping ([Sez94, Sez97]) maintained pointers from the subsectors to the corresponding sectors, which can make sector eviction somewhat difficult. On sector eviction it would be necessary for that scheme to search a significant fraction of the cache to determine which subsectors point to the sector being evicted. This search involves reading the status bits from all subsectors which could possibly belong to the sector being evicted, causing a significant number of cache access cycles to determine which subsectors should also be evicted. The sector pool design associates the placement information with the sector, allowing a much faster determination of the group of subsectors to be evicted when a sector is replaced.

4 Results

4.1 Miss Ratios

The miss ratios for a large variety of sector pool cache organizations were computed using our workload. Table 3 shows the miss ratios for a unified (combined instruction and data) cache, Tables 6 and 7 in Appendix A show the miss ratios for instruction and data caches. The parameters varied were nominal cache size (4K to 512K bytes), sector size (16 to 512 bytes), number of subsectors per sector (1 to 8), the set–associativity (2 to 8), and the number of possible depths (1 to 8). The buswidth was set to 8 bytes, which sets the lower bound on the minimum subsector size. We use the term *nominal cache size* to indicate that there are enough address tags to manage a cache of that size were it a traditional sector cache, but for sector pool caches with depth less than the set–associativity, the actual size will be less. For example, a cache having 64K nominal bytes with a depth of 5 (of 8 maximum) has really $65536 * \frac{5}{8} = 40960 = 40$K bytes of buffer space in which cached information can be stored.

Miss Ratios for 8–Way Set–Associative Unified Sector Pool Cache																		
Depth	Cache (Bytes)	Sector/Subsector Ratio=2						Sector/Subsector Ratio=4						Sector/Subsector Ratio=8				
		Sector Size (bytes)						Sector Size (bytes)						Sector Size (bytes)				
		16	32	64	128	256	512	16	32	64	128	256	512	32	64	128	256	512
1	4K	0.34039	0.24153	0.20320	0.20395	0.25089	0.34186	0.57034	0.34039	0.24159	0.20326	0.20409	0.25109	0.57041	0.34078	0.24206	0.20405	0.20543
	8K	0.26552	0.18636	0.15105	0.14300	0.16818	0.21924	0.44397	0.26553	0.18637	0.15111	0.14309	0.16835	0.44400	0.26562	0.18670	0.15156	0.14417
	16K	0.20590	0.14301	0.11324	0.10418	0.11401	0.14306	0.34488	0.20590	0.14301	0.11325	0.10423	0.11412	0.34490	0.20594	0.14309	0.11380	0.10473
	32K	0.15025	0.10396	0.08208	0.07379	0.07752	0.09490	0.25281	0.15025	0.10396	0.08208	0.07380	0.07759	0.25285	0.15027	0.10400	0.08214	0.07428
	64K	0.10824	0.07371	0.05716	0.05028	0.05225	0.06005	0.18376	0.10824	0.07372	0.05716	0.05028	0.05225	0.18378	0.10827	0.07374	0.05719	0.05032
	128K	0.07041	0.04715	0.03517	0.02936	0.02905	0.03177	0.12011	0.07041	0.04716	0.03517	0.02936	0.02905	0.12011	0.07042	0.04717	0.03519	0.02938
	256K	0.04898	0.03264	0.02411	0.01990	0.01918	0.01993	0.08441	0.04898	0.03264	0.02411	0.01990	0.01918	0.08441	0.04898	0.03264	0.02412	0.01991
	512K	0.03020	0.01956	0.01385	0.01078	0.00993	0.00985	0.05238	0.03020	0.01956	0.01385	0.01078	0.00993	0.05238	0.03020	0.01956	0.01385	0.01079
2	4K	0.24523	0.16540	0.12333	0.10373	0.10568	0.12271	0.42015	0.24532	0.16591	0.12386	0.10481	0.10691	0.42074	0.24764	0.16819	0.12764	0.10997
	8K	0.17382	0.11686	0.08694	0.07236	0.06985	0.07770	0.29750	0.17387	0.11698	0.08734	0.07276	0.07078	0.29773	0.17451	0.11835	0.08949	0.07607
	16K	0.12118	0.08036	0.05904	0.04883	0.04607	0.04919	0.20898	0.12121	0.08041	0.05912	0.04939	0.04648	0.20911	0.12144	0.08085	0.06079	0.05131
	32K	0.08235	0.05343	0.03860	0.03135	0.02980	0.03137	0.14439	0.08238	0.05345	0.03863	0.03140	0.03031	0.14456	0.08250	0.05363	0.03890	0.03278
	64K	0.05163	0.03272	0.02262	0.01738	0.01556	0.01574	0.09117	0.05165	0.03275	0.02265	0.01741	0.01560	0.09128	0.05180	0.03286	0.02277	0.01758
	128K	0.03302	0.02030	0.01351	0.00982	0.00800	0.00740	0.05861	0.03303	0.02031	0.01352	0.00984	0.00802	0.05865	0.03307	0.02035	0.01358	0.00992
	256K	0.02087	0.01261	0.00825	0.00585	0.00453	0.00389	0.03709	0.02087	0.01261	0.00826	0.00585	0.00453	0.03710	0.02088	0.01263	0.00827	0.00587
	512K	0.01476	0.00873	0.00552	0.00373	0.00271	0.00217	0.02640	0.01476	0.00874	0.00552	0.00373	0.00271	0.02640	0.01476	0.00874	0.00552	0.00374
3	4K	0.19389	0.13048	0.09636	0.07935	0.07640	0.08473	0.33176	0.19434	0.13199	0.09820	0.08232	0.08027	0.33432	0.19932	0.13777	0.10631	0.09306
	8K	0.13339	0.08899	0.06489	0.05405	0.04974	0.05278	0.22938	0.13359	0.08945	0.06663	0.05538	0.05222	0.23024	0.13552	0.09336	0.07149	0.06176
	16K	0.08815	0.05786	0.04212	0.03410	0.03304	0.03292	0.15424	0.08826	0.05806	0.04246	0.03573	0.03416	0.15473	0.08906	0.05931	0.04542	0.03930
	32K	0.05998	0.03777	0.02654	0.02106	0.01951	0.02045	0.10681	0.06009	0.03787	0.02668	0.02127	0.02076	0.10725	0.06047	0.03840	0.02742	0.02348
	64K	0.03646	0.02248	0.01478	0.01066	0.00884	0.00852	0.06506	0.03654	0.02258	0.01487	0.01075	0.00895	0.06537	0.03695	0.02289	0.01519	0.01117
	128K	0.02448	0.01459	0.00944	0.00658	0.00495	0.00416	0.04396	0.02450	0.01462	0.00947	0.00662	0.00501	0.04406	0.02463	0.01475	0.00962	0.00682
	256K	0.01566	0.00928	0.00589	0.00401	0.00292	0.00232	0.02803	0.01567	0.00929	0.00590	0.00402	0.00293	0.02807	0.01572	0.00933	0.00594	0.00407
	512K	0.01200	0.00697	0.00427	0.00275	0.00184	0.00133	0.02164	0.01200	0.00697	0.00428	0.00276	0.00185	0.02165	0.01202	0.00699	0.00429	0.00277
4	4K	0.16381	0.10975	0.08039	0.06578	0.06254	0.06654	0.27876	0.16586	0.11317	0.08460	0.07163	0.07060	0.28653	0.17536	0.12331	0.09724	0.08807
	8K	0.10769	0.07103	0.05138	0.04333	0.04047	0.04128	0.18552	0.10837	0.07237	0.05506	0.04727	0.04490	0.18799	0.11280	0.07972	0.06364	0.05631
	16K	0.07156	0.04557	0.03209	0.02517	0.02450	0.02594	0.12686	0.07193	0.04614	0.03294	0.02795	0.02778	0.12819	0.07377	0.04861	0.03767	0.03372
	32K	0.04761	0.02947	0.01987	0.01479	0.01300	0.01443	0.08552	0.04791	0.02975	0.02025	0.01536	0.01528	0.08644	0.04881	0.03091	0.02173	0.01872
	64K	0.02962	0.01794	0.01167	0.00816	0.00635	0.00567	0.05308	0.02985	0.01821	0.01188	0.00837	0.00664	0.05376	0.03066	0.01882	0.01247	0.00910
	128K	0.01943	0.01152	0.00740	0.00516	0.00383	0.00310	0.03493	0.01950	0.01161	0.00749	0.00526	0.00397	0.03518	0.01978	0.01185	0.00777	0.00562
	256K	0.01359	0.00796	0.00496	0.00328	0.00228	0.00174	0.02441	0.01362	0.00800	0.00499	0.00331	0.00231	0.02452	0.01375	0.00810	0.00508	0.00340
	512K	0.01072	0.00618	0.00373	0.00235	0.00151	0.00102	0.01943	0.01074	0.00619	0.00375	0.00236	0.00152	0.01948	0.01081	0.00626	0.00379	0.00240
5	4K	0.14192	0.09558	0.07084	0.05806	0.05478	0.05772	0.24501	0.14713	0.10223	0.07806	0.06697	0.06732	0.26002	0.16225	0.11756	0.09367	0.08672
	8K	0.09137	0.05892	0.04295	0.03646	0.03455	0.03531	0.16074	0.09315	0.06197	0.04874	0.04247	0.04158	0.16583	0.10089	0.07245	0.05946	0.05422
	16K	0.06227	0.03886	0.02645	0.02033	0.02031	0.02200	0.11180	0.06328	0.04015	0.02809	0.02447	0.02505	0.11459	0.06662	0.04398	0.03442	0.03169
	32K	0.03969	0.02452	0.01620	0.01161	0.00945	0.01108	0.07144	0.04037	0.02516	0.01700	0.01261	0.01272	0.07317	0.04199	0.02709	0.01917	0.01696
	64K	0.02565	0.01536	0.00996	0.00690	0.00523	0.00453	0.04628	0.02613	0.01590	0.01037	0.00731	0.00574	0.04746	0.02738	0.01685	0.01124	0.00830
	128K	0.01678	0.00990	0.00630	0.00434	0.00320	0.00257	0.03028	0.01695	0.01009	0.00647	0.00453	0.00345	0.03074	0.01742	0.01048	0.00691	0.00507
	256K	0.01244	0.00724	0.00446	0.00289	0.00194	0.00142	0.02245	0.01252	0.00733	0.00453	0.00295	0.00201	0.02271	0.01279	0.00753	0.00469	0.00311
	512K	0.00988	0.00566	0.00340	0.00212	0.00134	0.00087	0.01797	0.00993	0.00571	0.00344	0.00214	0.00136	0.01814	0.01009	0.00584	0.00353	0.00221
6	4K	0.12657	0.08505	0.06410	0.05322	0.05049	0.05422	0.22182	0.13727	0.09592	0.07486	0.06491	0.06638	0.24593	0.15696	0.11488	0.09239	0.08644
	8K	0.08100	0.05175	0.03685	0.03186	0.03110	0.03261	0.14542	0.08462	0.05688	0.04482	0.04022	0.04032	0.15366	0.09524	0.06957	0.05763	0.05359
	16K	0.05610	0.03461	0.02319	0.01741	0.01744	0.01950	0.10208	0.05818	0.03695	0.02591	0.02260	0.02341	0.10679	0.06308	0.04203	0.03313	0.03082
	32K	0.03527	0.02172	0.01426	0.01020	0.00817	0.00964	0.06407	0.03652	0.02284	0.01555	0.01171	0.01200	0.06682	0.03888	0.02538	0.01829	0.01650
	64K	0.02364	0.01408	0.00912	0.00631	0.00469	0.00394	0.04313	0.02446	0.01493	0.00979	0.00693	0.00541	0.04484	0.02611	0.01617	0.01086	0.00808
	128K	0.01518	0.00895	0.00565	0.00384	0.00279	0.00225	0.02752	0.01552	0.00928	0.00594	0.00415	0.00318	0.02827	0.01618	0.00981	0.00652	0.00483
	256K	0.01165	0.00677	0.00415	0.00265	0.00175	0.00124	0.02117	0.01185	0.00695	0.00428	0.00276	0.00186	0.02166	0.01227	0.00724	0.00451	0.00299
	512K	0.00926	0.00530	0.00316	0.00195	0.00122	0.00078	0.01694	0.00939	0.00542	0.00325	0.00201	0.00127	0.01729	0.00969	0.00562	0.00339	0.00212
7	4K	0.11660	0.07898	0.06018	0.05083	0.04857	0.05323	0.20843	0.13232	0.09294	0.07354	0.06425	0.06619	0.23976	0.15491	0.11409	0.09201	0.08637
	8K	0.07504	0.04755	0.03380	0.02974	0.02930	0.03139	0.13711	0.08080	0.05449	0.04335	0.03917	0.03983	0.14810	0.09319	0.06846	0.05697	0.05337
	16K	0.05214	0.03226	0.02160	0.01606	0.01600	0.01819	0.09622	0.05553	0.03562	0.02509	0.02186	0.02284	0.10279	0.06167	0.04137	0.03265	0.03051
	32K	0.03272	0.02013	0.01315	0.00947	0.00763	0.00914	0.06016	0.03466	0.02174	0.01490	0.01134	0.01181	0.06384	0.03756	0.02471	0.01790	0.01635
	64K	0.02206	0.01317	0.00859	0.00598	0.00445	0.00368	0.04067	0.02322	0.01431	0.00948	0.00676	0.00529	0.04280	0.02516	0.01575	0.01067	0.00798
	128K	0.01425	0.00843	0.00531	0.00356	0.00257	0.00208	0.02605	0.01479	0.00889	0.00571	0.00399	0.00308	0.02708	0.01562	0.00953	0.00637	0.00475
	256K	0.01112	0.00647	0.00396	0.00252	0.00164	0.00114	0.02041	0.01149	0.00677	0.00417	0.00268	0.00181	0.02116	0.01204	0.00713	0.00445	0.00295
	512K	0.00880	0.00504	0.00301	0.00185	0.00115	0.00074	0.01628	0.00907	0.00526	0.00315	0.00195	0.00123	0.01685	0.00949	0.00553	0.00334	0.00209
8	4K	0.11143	0.07704	0.05860	0.05006	0.04806	0.05305	0.20359	0.13115	0.09234	0.07320	0.06412	0.06616	0.23828	0.15445	0.11395	0.09193	0.08636
	8K	0.07253	0.04610	0.03294	0.02896	0.02847	0.03118	0.13398	0.07967	0.05391	0.04308	0.03891	0.03976	0.14653	0.09269	0.06829	0.05684	0.05332
	16K	0.05005	0.03134	0.02107	0.01574	0.01561	0.01797	0.09324	0.05470	0.03527	0.02486	0.02172	0.02277	0.10157	0.06133	0.04119	0.03251	0.03045
	32K	0.03102	0.01910	0.01259	0.00915	0.00742	0.00898	0.05740	0.03347	0.02103	0.01454	0.01117	0.01175	0.06168	0.03668	0.02413	0.01763	0.01630
	64K	0.02034	0.01231	0.00819	0.00578	0.00433	0.00360	0.03765	0.02179	0.01369	0.00924	0.00664	0.00524	0.04012	0.02401	0.01531	0.01050	0.00791
	128K	0.01380	0.00818	0.00517	0.00346	0.00250	0.00203	0.02539	0.01449	0.00875	0.00563	0.00395	0.00305	0.02661	0.01541	0.00943	0.00633	0.00472
	256K	0.01087	0.00634	0.00389	0.00246	0.00160	0.00111	0.02013	0.01137	0.00671	0.00414	0.00266	0.00179	0.02101	0.01198	0.00710	0.00443	0.00294
	512K	0.00856	0.00492	0.00294	0.00180	0.00112	0.00072	0.01601	0.00895	0.00520	0.00312	0.00193	0.00122	0.01671	0.00943	0.00550	0.00333	0.00208

Table 3: Unified miss ratios, real cache capacity is $\frac{Depth}{8} * Cache\ Size$.

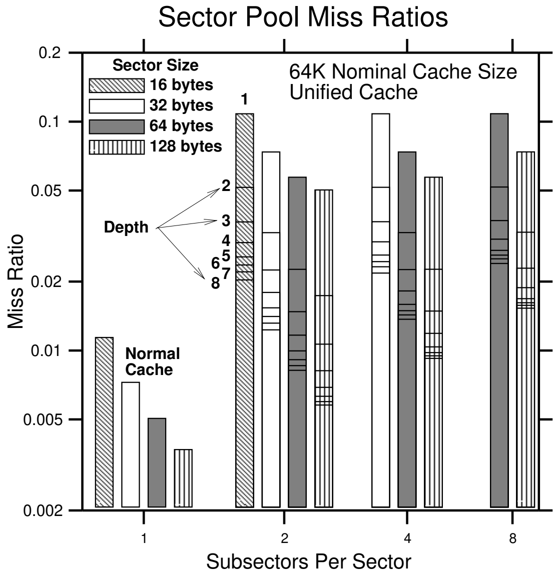


Figure 7: Miss ratios for 64K byte (nominal) size unified sector pool cache.

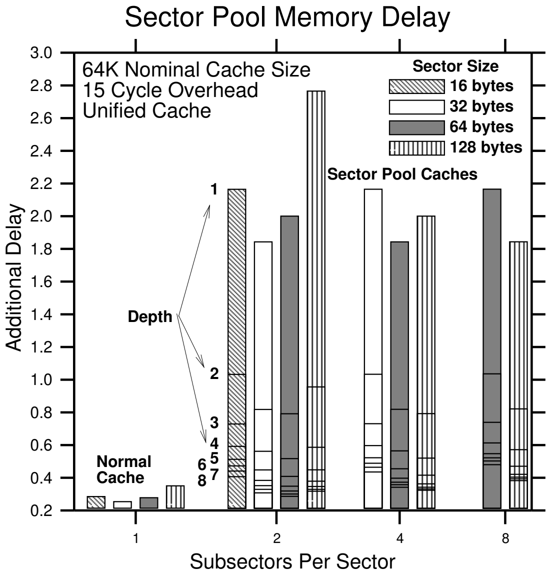


Figure 8: Sector pool cache additional delay for 64K byte nominal size unified cache, 15 cycle memory latency.

4.2 Single Level Caches

An initial comparison of the miss ratio and additional memory access delay looks promising compared to similar sector cache designs (sector pool cache with depth 8). For example, Figures 7 and 8 demonstrate that for caches with a sectoring degree of 4 or 8, decreasing the pool depth by up to 3 has little influence on performance (corresponding graphs for instruction and data caches can be found in Figures 12–16 in Appendix A). Particularly in Figure 8, the bars showing depth 5 through 8 are almost indistinguishable (e.g., 128 byte sectors with 32 byte subsectors). However, an important point to note is that the performance of normal caches (leftmost set of bars in Figures 7 and 8) contain at least one bar that outperforms all the sector caches. For any particular cache size and ignoring the resources (tag bits) required to implement it, a normal cache will outperform a sector cache due to the flexibility of mapping cache lines into the cache. A more proper comparison, however, would consider tag bits as well as data bits, and would consider memory access time, not just miss ratio.

The index of performance used in this paper to compare various cache organizations is *additional delay*, which does not take the intrinsic cache access delay of 1.0 into account. Additional delay (measured relative to processor cycle time) was determined for the single level cache using the formula:

$$additional\ delay = mr * (overhead + 5 * subsector/buswidth),$$

where overhead is either 5 or 15 cycles, buswidth is 8 bytes, "subsector" is the subsector size, and "mr" is the miss ratio. We choose 5 or 15 cycles by assuming a system using a 500 MHz CPU with a 100MHz 8 byte wide memory bus, using SDRAM with an X–1–1–1 response time, where X is 2 or 4 for single level caches and 5 for two–level caches. Note (see [Smi87]) that the minimum delay as we vary various parameters is dependent on the ratio of the overhead latency to the buswidth transfer rate, and not on either independently. For example, a 1000MHz CPU using a 100MHz motherboard would have double the observed miss induced memory delay (in processor cycles) of the delay values presented here, but the optimal configurations determined here would remain the same.

Figure 9 (Figures 17–19 in Appendix A for data and instruction caches) shows the minimal additional memory delay caused by cache misses vs. total bit resources for a first level unified cache. The miss penalty has startup overheads of 5 or 15 cycles, and a transfer time of 5 cycles for each 8 bytes. The best normal cache configurations are shown with the dashed lines with the block sizes shown. The sector pool configurations are displayed as dots and are too numerous to identify here, but can be found in Tables 9–11 in Appendix A. The solid lines connect the best sector pool configurations together to show a "best of all worlds" line, which also includes normal caches if they happen to be the minimal delay for the number of bits. For approximately one third of the best normal cache implementations (those with large block sizes), a sector pool implementation can provide similar if not better performance with many fewer bits. For example,

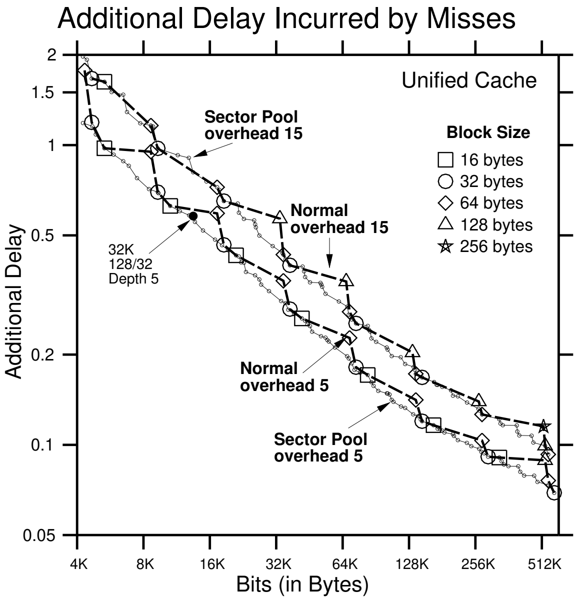


Figure 9: Minimal single level unified cache miss penalty designs (in processor cycles) for a given number of bits, 5 and 15 cycle memory transaction overheads.

a normal 32K byte cache with 128 byte blocks has an additional delay of 0.5685 and requires 33.23K bytes to implement. A sector pool cache with 32K nominal size with 128 byte sectors and 32 byte subsectors with depth 5 has an additional delay of 0.5670 and requires only 22.83K bytes to implement, which represents a saving of 31.3 percent in overall bits. Note, however, that a normal cache with a 64- or 32–byte block yields still better performance and require only a few more bits to implement.

Best Additional Memory Delay for Single Level On–chip Cache (15 Cycle Overhead)									
Bits (KBytes)	**Unified Cache**			**Instruction Cache**			**Data Cache**		
	Normal	Sector	Sector Pool	Normal	Sector	Sector Pool	Normal	Sector	Sector Pool
5.13	**1.667** 4K 32	**1.667** 4K 32/32	**1.667** D8 4K 32/32	**0.929** 4K 32	**0.929** 4K 32/32	**0.929** D8 4K 32/32	**1.573** 4K 32	**1.573** 4K 32/32	**1.573** D8 4K 32/32
6.46	**1.627** 4K 16	**1.627** 4K 16/16	**1.503** D5 8K 64/32	**0.929** 4K 32	**0.929** 4K 32/32	**0.720** D3 16K 256/32	**1.534** 4K 16	**1.534** 4K 16/16	**1.526** D5 8K 64/32
8.14	**1.627** 4K 16	**1.627** 4K 16/16	**1.183** D7 8K 64/32	**0.929** 4K 32	**0.929** 4K 32/32	**0.613** D7 8K 64/32	**1.534** 4K 16	**1.534** 4K 16/16	**1.284** D7 8K 64/32
10.25	**0.975** 8K 32	**0.975** 8K 32/32	**0.975** D8 8K 32/32	**0.528** 8K 32	**0.528** 8K 32/32	**0.528** D8 8K 32/32	**1.104** 8K 32	**1.104** 8K 32/32	**1.104** D8 8K 32/32
12.92	**0.975** 8K 32	**0.975** 8K 32/32	**0.907** D6 16K 128/32	**0.528** 8K 32	**0.528** 8K 32/32	**0.408** D6 16K 128/64	**1.104** 8K 32	**1.104** 8K 32/32	**1.104** D8 8K 32/32
16.28	**0.975** 8K 32	**0.975** 8K 32/32	**0.756** D7 16K 64/32	**0.528** 8K 32	**0.528** 8K 32/32	**0.369** D7 16K 64/32	**1.104** 8K 32	**1.104** 8K 32/32	**0.979** D7 16K 64/32
20.51	**0.651** 16K 32	**0.651** 16K 32/32	**0.651** D8 16K 32/32	**0.303** 16K 64	**0.303** 16K 64/64	**0.287** D5 32K 512/64	**0.895** 16K 32	**0.895** 16K 32/32	**0.895** D8 16K 32/32
25.84	**0.651** 16K 32	**0.651** 16K 32/32	**0.544** D6 32K 128/32	**0.303** 16K 64	**0.303** 16K 64/64	**0.219** D6 32K 128/64	**0.895** 16K 32	**0.895** 16K 32/32	**0.829** D5 32K 64/32
32.55	**0.651** 16K 32	**0.651** 16K 32/32	**0.460** D7 32K 64/32	**0.303** 16K 64	**0.261** 32K 512/64	**0.202** D7 32K 64/32	**0.895** 16K 32	**0.895** 16K 32/32	**0.756** D7 32K 64/32
41.02	**0.397** 32K 32	**0.397** 32K 32/32	**0.397** D8 32K 32/32	**0.175** 32K 32	**0.175** 32K 32/32	**0.175** D8 32K 32/32	**0.667** 32K 32	**0.667** 32K 32/32	**0.665** D4 64K 64/32
51.68	**0.397** 32K 32	**0.397** 32K 32/32	**0.343** D6 64K 128/32	**0.175** 32K 32	**0.175** 32K 32/32	**0.135** D3 128K 256/64	**0.667** 32K 32	**0.667** 32K 32/32	**0.577** D6 64K 128/32
65.11	**0.397** 32K 32	**0.397** 32K 32/32	**0.301** D7 64K 64/32	**0.175** 32K 32	**0.148** 64K 512/64	**0.114** D7 64K 128/64	**0.667** 32K 32	**0.667** 32K 32/32	**0.527** D7 64K 64/32
82.03	**0.254** 64K 32	**0.254** 64K 32/32	**0.254** D8 64K 32/32	**0.086** 64K 64	**0.086** 64K 64/64	**0.074** D5 128K 512/64	**0.468** 64K 32	**0.468** 64K 32/32	**0.468** D8 64K 32/32
103.35	**0.254** 64K 32	**0.254** 64K 32/32	**0.208** D6 128K 128/32	**0.086** 64K 64	**0.086** 64K 64/64	**0.056** D6 128K 128/64	**0.468** 64K 32	**0.468** 64K 32/32	**0.425** D3 256K 256/32
130.22	**0.254** 64K 32	**0.254** 64K 32/32	**0.186** D7 128K 64/32	**0.085** 128K 512	**0.062** 128K 512/64	**0.050** D7 128K 128/64	**0.468** 64K 32	**0.468** 64K 32/32	**0.402** D7 128K 64/32
164.06	**0.168** 128K 32	**0.168** 128K 32/32	**0.168** D8 128K 32/32	**0.043** 128K 64	**0.043** 128K 64/64	**0.041** D5 256K 512/64	**0.372** 128K 32	**0.372** 128K 32/32	**0.372** D8 128K 32/32
206.71	**0.168** 128K 32	**0.168** 128K 32/32	**0.146** D6 256K 128/64	**0.043** 128K 64	**0.043** 128K 64/64	**0.032** D6 256K 128/64	**0.372** 128K 32	**0.372** 128K 32/32	**0.334** D6 256K 128/64
260.43	**0.168** 128K 32	**0.162** 256K 512/64	**0.138** D7 256K 128/64	**0.036** 256K 256	**0.036** 256K 256/256	**0.029** D7 256K 64/32	**0.372** 128K 32	**0.370** 256K 512/64	**0.320** D7 256K 128/64
328.13	**0.126** 256K 64	**0.126** 256K 64/64	**0.126** D8 256K 64/64	**0.024** 256K 32	**0.024** 256K 32/32	**0.023** D5 512K 512/64	**0.294** 256K 64	**0.294** 256K 64/64	**0.294** D8 256K 64/64
413.41	**0.126** 256K 64	**0.126** 256K 64/64	**0.107** D6 512K 128/64	**0.024** 256K 32	**0.024** 256K 32/32	**0.017** D6 512K 128/64	**0.294** 256K 64	**0.294** 256K 64/64	**0.254** D6 512K 128/64
520.87	**0.115** 512K 256	**0.114** 512K 512/64	**0.102** D7 512K 128/64	**0.018** 512K 256	**0.017** 512K 256/128	**0.014** D7 512K 64/32	**0.268** 512K 256	**0.268** 512K 256/256	**0.241** D7 512K 128/64
656.25	**0.093** 512K 64	**0.093** 512K 64/64	**0.093** D8 512K 64/64	**0.012** 512K 32	**0.012** 512K 32/32	**0.012** D8 512K 32/32	**0.222** 512K 64	**0.222** 512K 64/64	**0.222** D8 512K 64/64

Table 4: Best on–chip single level cache implementation for the given gross cache size. Each entry consists of **additional memory delay**, nominal cache size (real cache size for normal and sector caches), sector (/subsector for sector and sector pool caches) and **D**, the pool depth (for sector pool caches).

Table 4 provides information for a sampling of configurations of performance vs. bit resources for first level caches (15 cycle overhead). The performance is measured by the additional memory delay caused by misses. Sector pool caches perform the best; both sector and normal caches are considered to be proper subsets of the sector pool design. However, it is important to note that each entry uses fewer or equal numbers of bits than the left–most column. Sector pool caches are not restricted to a power of two number of data bits, so intermediate implementations can be created that have no counterpart for normal or sector caches. The performance of the best sector pool caches follow the trend of the normal caches, deviating by only a few percent in the best cases. The chief advantage of single level sector pool caches is that the flexibility of the organization allows any arbitrary size of cache to be implemented and is less restricted in the size of the data space. As can be seen, regular sector caches are very rarely better than normal caches for single level caches, which was also noted in [RS99].

4.3 Two–Level Cache Organizations

Two–level caches have become increasingly common, because they often yield higher performance. A small cache is often significantly faster than a larger one, and thus the cost of more misses from the smaller first level cache is more than compensated for by the shorter access time, which itself often translates directly to a higher clock frequency [JW94]. A desirable configuration for a two–level cache is for the first level to hold both the first level cache itself and the tags for the second level cache. In this case, a second level sector cache has the advantage that the limited number of bits available on the first level can control a very large set of data arrays on the second level. We consider that design in this section.

To evaluate two–level cache organizations, we assume that the processor die contains the entire first level cache and the tags and other control circuitry for the second level cache. The base delay for a hit in the second level cache is two cycles plus one cycle for each eight bytes transferred from the L2 cache to the L1 cache. The formula for measuring the delay is:

$$additional\ memory\ delay = mr_1 * (2 + subsector_1/8) + mr_2 * (20 + 5 * subsector_2/8)$$

For the two–level experiments presented here, we examined first level caches from 4K to 256K bytes. The second level cache varied from 8K to 512K bytes. The sector sizes were varied from 16 to 512 bytes and the subsector sizes between 8 and 512 bytes. The bus width between the CPU, the caches, and main memory was set to 8 bytes. Both L1 and L2 caches were eight–way set–associative and all possible combinations of normal, sector, and sector pool organizations for first and second level caches were examined to find the minimum miss induced memory delay for a given upper limit of on–chip cache bits.

Due to the computationally intensive nature of multilevel cache simulations, the miss ratios of the single level simulations were used to generate the results in this section. This may violate the inclusion principle of multilevel memory hierarchies when the ratio of the sector sizes from L1 and L2 caches is large [BW88]. 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.

Sector pool caches are never an optimal choice for an L2 cache, because they require more tag/control bits than normal sector caches due to the necessary pointer bits associated with each sector. Using a sector pool cache as the first level cache and a normal sector cache as the second level cache in a two–level cache organization generally shows the best performance. When the second level tag and control bits are on–chip, a level two sector cache allows the possibility of controlling the most cache with the smallest number of bits. The combination of an L1 sector pool cache with an L2 sector cache with on–chip tags provides significantly better performance (in terms of memory access delay) than a normal cache, when the gross on–chip cache size is limited.

Table 5 shows the best configurations for a given gross on–chip cache size, which includes the whole level one cache and the tags for an off–chip level two cache. For each type of address stream (unified, instruction, data), three different organizations (normal, sector, and sector pool) are presented. The results show that sector pool organizations for a first level cache outperform normal caches, with a large difference in memory delay for small cache sizes. However, once there are enough on–chip bits to control the largest off–chip cache size simulated, performance for the various organizations begins to converge. For the largest on–chip bit budget allowed (328KB), the best organizations use only normal caches. In some cases (unified and data caches), the extra delay of an L2 cache only twice the size of the L1 cache results in certain configurations that are better off without an L2 cache. We believe that it is quite likely that the advantages of sector pool caches for the first level would continue for the larger first level caches, were we to consider still larger second level caches and large commercial (data base or transaction processing) workloads. Unfortunately, we don't have such workload traces available.

An example of the trade–offs between sector, normal and sector pool caches can be seen in part of one row in Table 5, which shows the three organizational choices for a data cache with a maximum allowed 6.46KB gross on–chip cache size. The sector and sector pool cache both use 4K level one caches with 32 byte sectors and 16 byte subsectors. However, the sector pool cache instead utilizes 25 percent of its potential data space as tag bits to enable management of twice as much off–chip L2 cache (256KB) as the sector design (128KB) and eight times as much as the normal cache design (32KB). This trade–off decreases delay induced by misses by about 13.3 percent over the sector cache organization and 42.8 percent over the normal cache organization.

An interesting feature of Table 5 is the pattern by which level two cache is added as the on–chip resources increase. Given a choice between increasing the L2 cache size while keeping large (and inefficient) subsector sizes, or decreasing the subsector size with constant L2 cache size, increasing the cache size is the preferred option. Both options require roughly the same number of tag bits as both require the same number of tags. By using a sector cache organization, more efficient transfer (subsector) sizes can be utilized while keeping the total number of tags constant. Sector caches are thus a powerful method of adding second level cache with reasonable L2 fetch sizes when the number of on–chip bits is constrained.

4.4 Summary of Results

From the results presented here, it is possible to see that the sector pool cache provides small, but in some cases significant improvements in performance for single level caches. The best use for sector pool caches, however, is for use as the first level in two–level designs.

Sector pool caches perform best when used as a first level cache in a two–level cache organization with second level cache tags on–chip. Particularly when the number of on–chip bits is small, sector pool caches allow more flexible organizations that can trade L1 cache data bits for L2 cache tag/control bits to more precisely manage larger off–chip caches. Sector pool caches are not a good choice for the off–chip cache; regular sector caches perform the best in that role over most of the range we examined.

We note that to a large extent, it is possible to deduce our simulation results (of course, had we simply deduced them, readers would rightly have insisted on simulations to confirm our insights). Sector caches trade mapping flexibility for a savings in tag bits. This trade–off is worthwhile when tag bits are a significant component of the storage bits. Tag bits are significant when the line sizes are small. Small line sizes are best when transmission time for a miss is much higher than latency and when cache sizes are small [Smi87]. So sector caches are most useful when caches are small and miss latencies are small. Sector pool caches are most useful, as explained above, when the savings in data array area can be put to good use; that occurs most conspicuously when a first level sector pool cache shares space with tags for a second level sector cache.

5 Conclusions

We have presented and evaluated a new cache design called the **sector pool cache**. Regular sector caches associate several transfer units (subsectors) with each address tag (sector frame) in the cache. Studies have shown that many of the subsectors are not accessed before the corresponding sector is evicted from the cache. Sector pool caches attempt to utilize this potentially wasted space by sharing a pool of subsectors among all the sectors in a set. Each set maintains a list of subsectors for each possible subsector position in a sector frame. Associated with the sector frames are pointers into these lists. When the number of subsectors in each list is smaller than the set–associativity of the sets, less data space is used than in a normal cache, but with only a small impact on the miss ratio and particularly on the observed memory delay.

The increase in memory delay and miss ratio is small compared to the performance of regular sector caches, but it is still significantly higher than that of normal (unsectored) caches using the same number of blocks as the sector cache has sectors. To evaluate the best design, we determined which organization has the best performance for a given number of bits with which to implement the cache. Sector pool caches generally have the best performance for almost any specified number of bits; however, since there many more configuration of sector pool caches this should not be a great surprise. The best performing sector pool caches follow the performance trend of normal caches for single level on–chip caches.

Sector pool caches are most useful when used as the first level cache for two–level cache architectures. In a system that contains the first level cache and the tags for the second level cache on the same die, the versatile combination of a sector pool design as the level one cache and a regular sector cache design for level two outperforms normal or plain sector cache configurations. This combination is particularly powerful for small on–chip cache budgets, as it allows control of a large second level cache by trading–off some of the data space of the first level cache.

Our test methodology used a trace–driven simulated multiprogrammed workload to strenuously evaluate the variety of cache designs. Previous research [GHPS93] has shown that many of the workloads traditionally used as benchmarks to evaluate caches tend have low and unrepresentative miss ratios that do not put much stress on the memory subsystem. Our simulated workload accesses many more unique memory locations than the bytes available in the caches under test, which results in a fairer and more realistic evaluation of the various configurations.

References

[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.

[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.

[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 Trans. on Computers*, 38(12):1612–1630, December 1989.

[JW94] Norman P. Jouppi and Steven J. E. Wilton. Tradeoffs in Two–Level On–Chip Caching. *Proc. 21st Annual International Symposium on Computer Architecture*, pp. 34–45, Chicago, IL April 18–31 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.

[PHS98] Jih-Kwon Peir, Windsor Hsu, and Alan Jay Smith. Implementation Issues in Modern Cache Memories. Technical Report UCB/CSD–98–1023, Computer Science Division, U.C. Berkeley, Berkeley, CA, November 1998. To appear in IEEE Trans. on Computers.

[Prz90] 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.

[RS99] Jeffrey B. Rothman and Alan Jay Smith. Sector Cache Design and Performance. Technical Report UCB/CSD–99–1034, Computer Science Division, U.C. Berkeley, Berkeley, CA, January 1999.

[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.

[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.

[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.

[WSY95] Hong Wang, Tong Sun, and Qing Yang. CAT – Caching Address Tags: A Technique for Reducing Area Cost of On–chip Caches. *Proc. 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 22–24 1995.

[WSY97] Hong Wang, Tong Sun, and Qing Yang. Minimizing Area Cost of On–Chip Cache Memories by Caching Address Tags. *IEEE Trans. on Computers*, 46(11):1187–1201, November 1997.

Best Additional Memory Delay for Given Gross Cache Size									
Bits (KBytes)	Unified Cache			Instruction Cache			Data Cache		
	Normal	Sector	Sector Pool	Normal	Sector	Sector Pool	Normal	Sector	Sector Pool
5.13	**1.581** c1 4K 64 c2 64K 512	**1.078** c1 4K 64/16 c2 64K 512/128	**0.815** c1 4K 32/16 D6 c2 128K 512/128	**0.560** c1 4K 64 c2 64K 512	**0.484** c1 4K 64/32 c2 64K 512/64	**0.400** c1 4K 64/32 D7 c2 128K 512/128	**1.797** c1 4K 32 no c2	**1.411** c1 4K 128/16 c2 64K 512/64	**0.980** c1 4K 32/8 D3 c2 256K 512/128
6.46	**0.947** c1 4K 32 c2 128K 512	**0.746** c1 4K 32/16 c2 128K 512/64	**0.690** c1 4K 32/16 D6 c2 256K 512/128	**0.330** c1 4K 16 c2 128K 512	**0.317** c1 4K 16/16 c2 128K 512/256	**0.317** c1 4K 16/16 D8 c2 128K 512/256	**1.462** c1 4K 32 c2 32K 128	**0.982** c1 4K 32/16 c2 128K 512/64	**0.851** c1 4K 32/16 D6 c2 256K 512/128
8.14	**0.630** c1 4K 16 c2 256K 512	**0.570** c1 4K 16/16 c2 256K 512/128	**0.570** c1 4K 16/16 D8 c2 256K 512/128	**0.292** c1 4K 16 c2 256K 512	**0.282** c1 4K 16/16 c2 256K 512/128	**0.256** c1 8K 64/16 D5 c2 256K 512/256	**0.941** c1 4K 16 c2 256K 512	**0.784** c1 4K 16/16 c2 256K 512/128	**0.765** c1 4K 32/8 D5 c2 512K 512/256
10.25	**0.537** c1 4K 16 c2 512K 512	**0.519** c1 4K 16/16 c2 512K 512/256	**0.490** c1 8K 32/16 D6 c2 256K 512/128	**0.264** c1 8K 64 c2 128K 512	**0.237** c1 8K 64/32 c2 128K 512/128	**0.215** c1 8K 64/32 D7 c2 256K 512/128	**0.710** c1 4K 16 c2 512K 512	**0.669** c1 4K 16/16 c2 512K 512/256	**0.669** c1 4K 16/16 D8 c2 512K 512/256
12.92	**0.490** c1 8K 16 c2 256K 512	**0.453** c1 8K 32/16 c2 256K 512/64	**0.432** c1 8K 32/16 D7 c2 512K 512/512	**0.199** c1 8K 16 c2 256K 512	**0.190** c1 8K 16/16 c2 256K 512/128	**0.188** c1 16K 256/32 D5 c2 256K 512/128	**0.710** c1 4K 16 c2 512K 512	**0.652** c1 4K 16/16 c2 512K 512/128	**0.600** c1 8K 32/16 D6 c2 512K 512/128
16.28	**0.397** c1 8K 16 c2 512K 512	**0.372** c1 8K 16/16 c2 512K 512/128	**0.372** c1 8K 16/16 D8 c2 512K 512/128	**0.176** c1 8K 16 c2 512K 512	**0.172** c1 8K 16/16 c2 512K 512/128	**0.153** c1 16K 64/32 D5 c2 512K 512/128	**0.618** c1 8K 16 c2 512K 512	**0.559** c1 8K 16/16 c2 512K 512/128	**0.559** c1 8K 16/16 D8 c2 512K 512/128
20.51	**0.369** c1 8K 16 c2 512K 256	**0.363** c1 8K 16/16 c2 512K 256/128	**0.329** c1 16K 32/16 D6 c2 512K 512/128	**0.148** c1 16K 64 c2 256K 512	**0.136** c1 16K 64/32 c2 256K 512/128	**0.127** c1 16K 64/32 D7 c2 512K 512/128	**0.551** c1 8K 16 c2 512K 256	**0.537** c1 8K 16/16 c2 512K 256/128	**0.537** c1 8K 16/16 D8 c2 512K 256/128
25.84	**0.318** c1 16K 16 c2 512K 512	**0.301** c1 16K 16/16 c2 512K 512/256	**0.300** c1 32K 64/16 D4 c2 512K 512/128	**0.113** c1 16K 32 c2 512K 512	**0.108** c1 16K 32/32 c2 512K 512/128	**0.099** c1 32K 256/32 D5 c2 512K 512/128	**0.551** c1 8K 16 c2 512K 256	**0.533** c1 16K 32/16 c2 512K 512/128	**0.518** c1 16K 32/16 D6 c2 512K 256/128
32.55	**0.290** c1 16K 16 c2 512K 256	**0.284** c1 16K 16/16 c2 512K 256/128	**0.259** c1 32K 64/16 D6 c2 512K 512/128	**0.106** c1 16K 32 c2 512K 256	**0.105** c1 16K 32/32 c2 512K 256/128	**0.084** c1 32K 64/32 D6 c2 512K 512/128	**0.515** c1 16K 16 c2 512K 256	**0.501** c1 16K 16/16 c2 512K 256/128	**0.495** c1 32K 32/16 D4 c2 512K 256/128
41.02	**0.276** c1 16K 16 c2 512K 128	**0.247** c1 32K 64/32 c2 512K 512/128	**0.240** c1 64K 128/32 D4 c2 512K 512/128	**0.083** c1 32K 64 c2 512K 512	**0.077** c1 32K 64/32 c2 512K 512/128	**0.074** c1 64K 128/32 D4 c2 512K 512/128	**0.484** c1 16K 16 c2 512K 128	**0.479** c1 32K 64/16 c2 512K 512/128	**0.466** c1 32K 64/16 D5 c2 512K 128/128
51.68	**0.224** c1 32K 16 c2 512K 256	**0.218** c1 32K 16/16 c2 512K 256/128	**0.218** c1 32K 16/16 D8 c2 512K 256/128	**0.069** c1 32K 32 c2 512K 256	**0.068** c1 32K 32/32 c2 512K 256/128	**0.065** c1 64K 128/32 D5 c2 512K 256/256	**0.446** c1 32K 16 c2 512K 256	**0.432** c1 32K 16/16 c2 512K 256/128	**0.432** c1 32K 16/16 D8 c2 512K 256/128
65.11	**0.210** c1 32K 16 c2 512K 128	**0.210** c1 32K 16/16 c2 512K 128/128	**0.200** c1 64K 64/16 D5 c2 512K 128/128	**0.065** c1 32K 32 c2 512K 128	**0.065** c1 32K 32/32 c2 512K 128/128	**0.057** c1 128K 128/32 D3 c2 512K 256/128	**0.414** c1 32K 16 c2 512K 128	**0.414** c1 32K 16/16 c2 512K 128/128	**0.397** c1 64K 64/16 D5 c2 512K 128/128
82.03	**0.207** c1 32K 16 c2 512K 64	**0.194** c1 64K 64/32 c2 512K 256/128	**0.183** c1 128K 128/16 D4 c2 512K 256/128	**0.044** c1 64K 32 c2 512K 256	**0.044** c1 64K 32/32 c2 512K 256/256	**0.044** c1 128K 128/32 D4 c2 512K 256/128	**0.413** c1 32K 16 c2 512K 64	**0.403** c1 64K 64/16 c2 512K 256/128	**0.381** c1 64K 32/16 D6 c2 512K 128/128
103.35	**0.173** c1 64K 16 c2 512K 128	**0.173** c1 64K 16/16 c2 512K 128/128	**0.173** c1 128K 64/16 D5 c2 512K 256/128	**0.040** c1 64K 32 c2 512K 128	**0.040** c1 64K 32/32 c2 512K 128/128	**0.035** c1 128K 128/32 D5 c2 512K 128/128	**0.367** c1 64K 16 c2 512K 128	**0.367** c1 64K 16/16 c2 512K 128/128	**0.367** c1 64K 16/16 D8 c2 512K 128/128
130.22	**0.170** c1 64K 16 c2 512K 64	**0.170** c1 64K 16/16 c2 512K 64/64	**0.160** c1 128K 64/16 D6 c2 512K 128/128	**0.038** c1 64K 32 c2 512K 64	**0.038** c1 64K 32/32 c2 512K 64/64	**0.031** c1 128K 64/32 D6 c2 512K 128/128	**0.366** c1 64K 16 c2 512K 64	**0.366** c1 64K 16/16 c2 512K 64/64	**0.358** c1 256K 128/16 D3 c2 512K 128/128
164.06	**0.161** c1 128K 64 c2 512K 128	**0.156** c1 128K 64/32 c2 512K 128/128	**0.153** c1 256K 128/16 D4 c2 512K 128/128	**0.027** c1 128K 32 c2 512K 128	**0.027** c1 128K 32/32 c2 512K 128/128	**0.027** c1 128K 32/32 D8 c2 512K 128/128	**0.366** c1 64K 16 c2 512K 64	**0.357** c1 128K 64/16 c2 512K 128/128	**0.350** c1 256K 128/16 D4 c2 512K 128/128
206.71	**0.148** c1 128K 16 c2 512K 64	**0.148** c1 128K 16/16 c2 512K 64/64	**0.148** c1 128K 16/16 D8 c2 512K 64/64	**0.026** c1 128K 32 c2 512K 64	**0.026** c1 128K 32/32 c2 512K 64/64	**0.024** c1 256K 128/32 D5 c2 512K 64/64	**0.345** c1 128K 16 c2 512K 64	**0.345** c1 128K 16/16 c2 512K 64/64	**0.343** c1 256K 64/16 D5 c2 512K 128/128
260.43	**0.148** c1 128K 16 c2 512K 64	**0.148** c1 128K 16/16 c2 512K 64/64	**0.143** c1 256K 64/32 D6 c2 512K 64/64	**0.026** c1 128K 32 c2 512K 64	**0.026** c1 128K 32/32 c2 512K 64/64	**0.022** c1 256K 64/32 D6 c2 512K 64/64	**0.345** c1 128K 16 c2 512K 64	**0.345** c1 128K 16/16 c2 512K 64/64	**0.337** c1 256K 32/16 D6 c2 512K 128/128
328.13	**0.138** c1 256K 64 no c2	**0.138** c1 256K 64/64 no c2	**0.138** c1 256K 64/64 D8 no c2	**0.020** c1 256K 32 c2 512K 64	**0.020** c1 256K 32/32 c2 512K 64/64	**0.020** c1 256K 32/32 D8 c2 512K 64/64	**0.321** c1 256K 64 no c2	**0.321** c1 256K 64/64 no c2	**0.321** c1 256K 64/64 D8 no c2

Table 5: Best memory delay for given on–chip resources. Each entry contains the average memory delay, the level one cache size (**c1**), which consists of the nominal cache size, sector/subsector (or just block) size in bytes, and the pool depth (**D**) for sector pool organizations. The second level cache (**c2**) shows the same type of information as the c1 cache. When not specified, the pool depth is the same as the set–associativity of the cache (8).

A Supplemental Tables and Figures

This appendix contains figures and tables of an archival nature, not necessary for understanding the research presented here, but useful to the computer designer and implementor planning to apply our results.

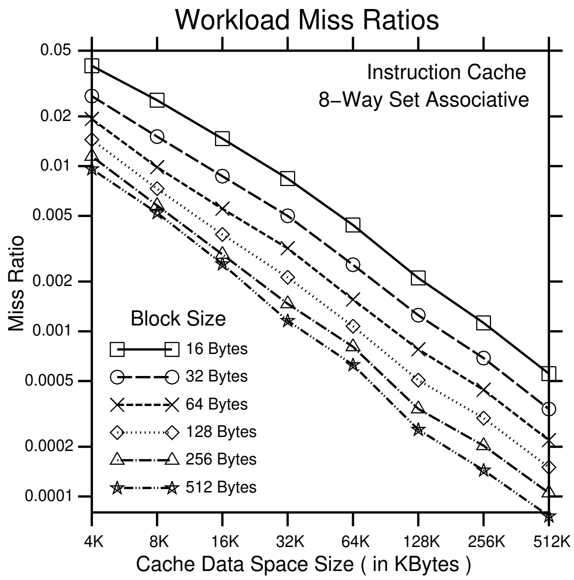


Figure 10: Miss ratios of the simulated multiprogrammed workload for a normal instruction cache.

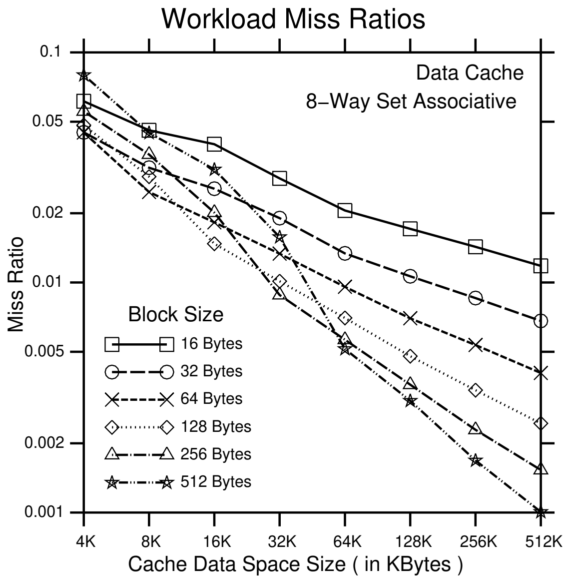


Figure 11: Miss ratios of the simulated multiprogrammed workload for a normal data cache.

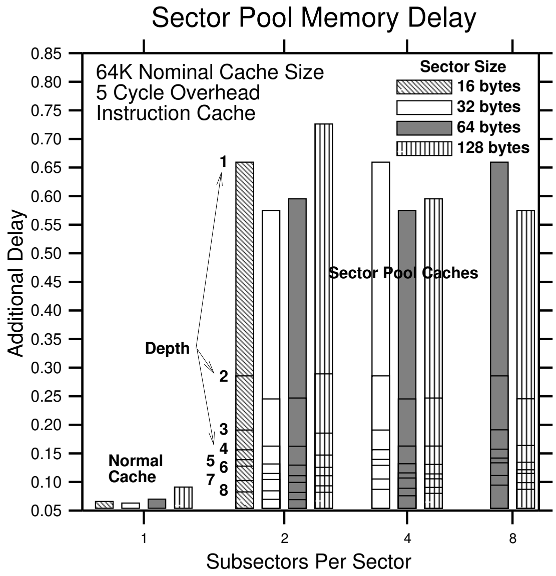


Figure 12: Sector pool cache additional delay for 64K byte nominal size instruction cache, 5 cycle memory transaction overhead.

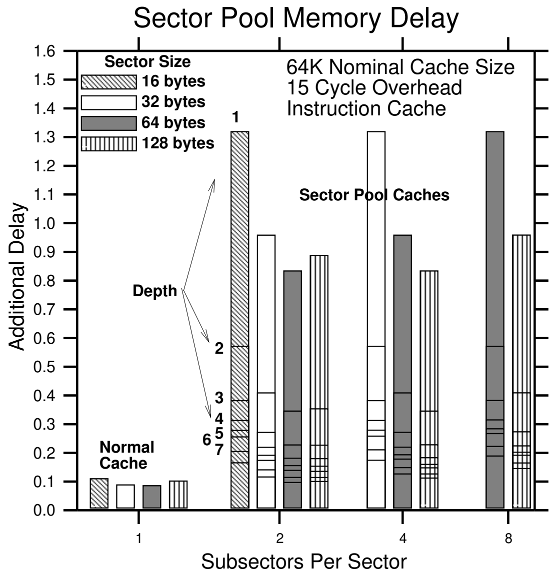


Figure 13: Sector pool cache additional delay for 64K byte nominal size instruction cache, 15 cycle memory transaction overhead.

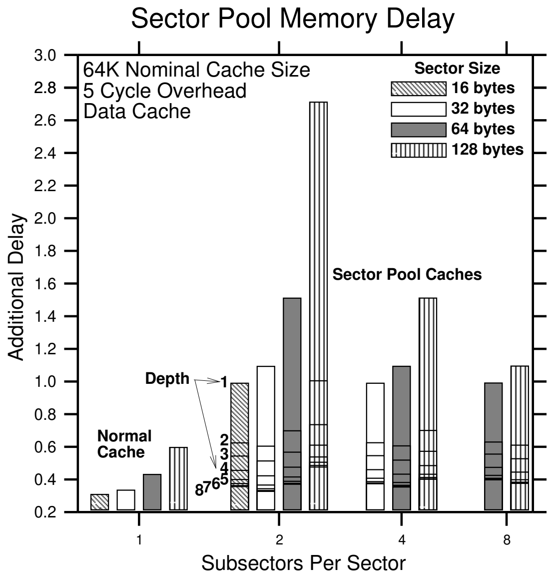


Figure 14: Sector pool cache additional delay for 64K byte nominal size data cache, 5 cycle memory transaction overhead.

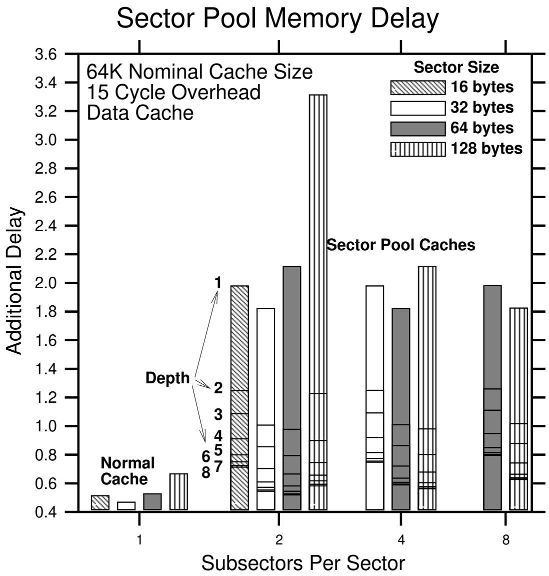
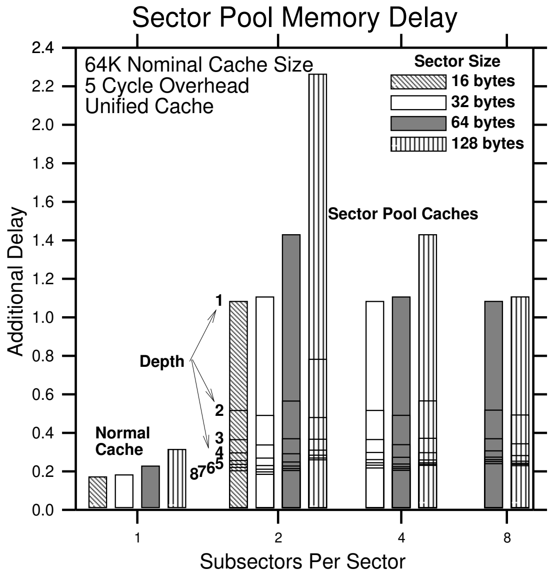


Figure 15: Sector pool cache additional delay for 64K byte nominal size data cache, 15 cycle memory transaction overhead.



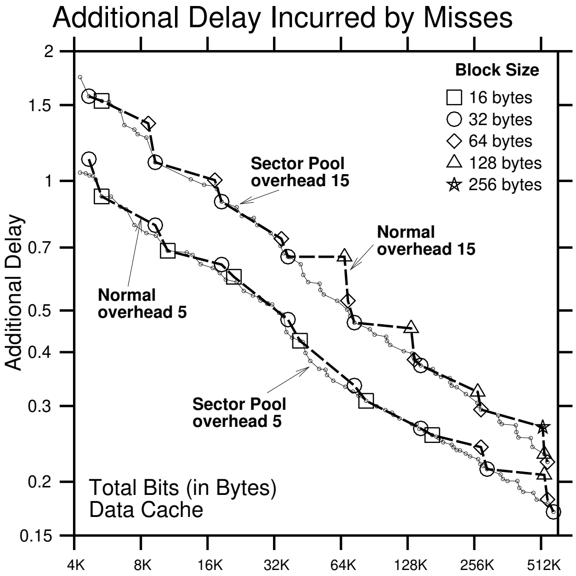
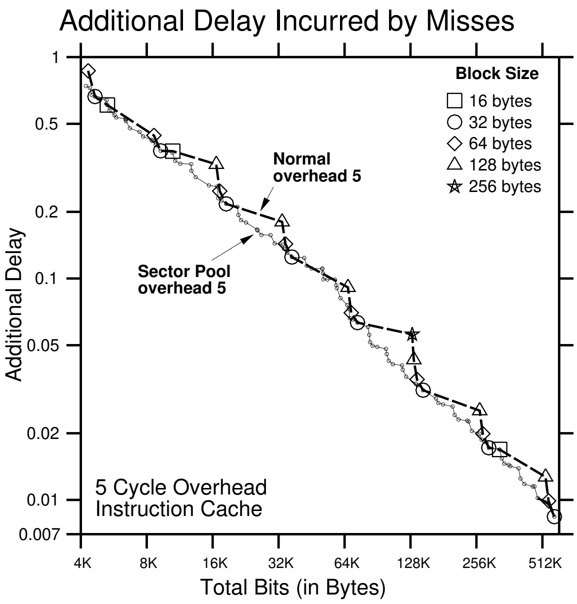


Figure 16: Sector pool cache additional delay for 64K byte nominal size unified cache, 5 cycle memory transaction overhead.

Figure 17: Minimal single level data cache miss penalty designs for a given number of bits, 5 and 15 cycle memory transaction overheads. The organization information for the sector pool caches can be found in Table 11.



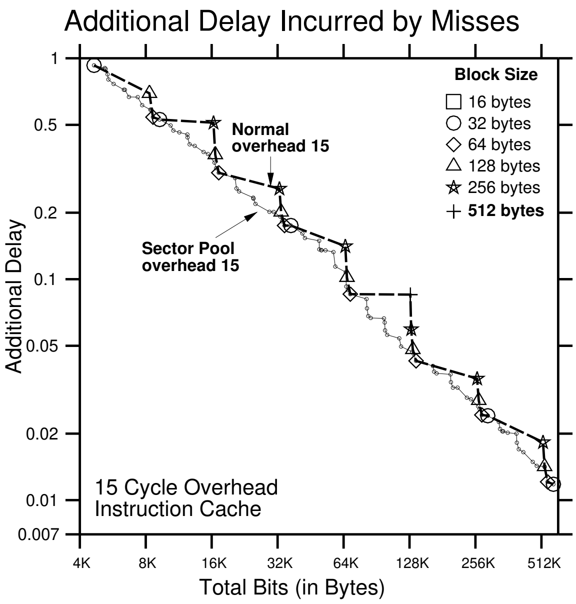


Figure 18: Minimal single level instruction cache miss penalty designs for a given number of bits, 5 cycle memory transaction overhead. The organization information for the sector pool caches can be found in Table 10.

Figure 19: Minimal single level instruction cache miss penalty designs for a given number of bits, 15 cycle memory transaction overhead. The organization information for the sector pool caches can be found in Table 10.

Miss Ratios for 8–Way Set–Associative Instruction Sector Pool Cache																		
Depth	Cache (Bytes)	Sector/Subsector Ratio=2						Sector/Subsector Ratio=4						Sector/Subsector Ratio 8				
		Sector Size (bytes)						Sector Size (bytes)						Sector Size (bytes)				
		16	32	64	128	256	512	16	32	64	128	256	512	32	64	128	256	512
1	4K	0.27940	0.16357	0.10242	0.06761	0.05111	0.04096	0.50931	0.27940	0.16357	0.10243	0.06762	0.05111	0.50932	0.27941	0.16359	0.10246	0.06764
	8K	0.20237	0.11914	0.07480	0.05009	0.03736	0.02929	0.36633	0.20237	0.11914	0.07480	0.05010	0.03736	0.36634	0.20238	0.11915	0.07484	0.05014
	16K	0.15033	0.08923	0.05638	0.03761	0.02708	0.02091	0.27118	0.15033	0.08923	0.05638	0.03761	0.02708	0.27118	0.15034	0.08924	0.05640	0.03763
	32K	0.09878	0.05779	0.03634	0.02453	0.01808	0.01437	0.17983	0.09878	0.05779	0.03634	0.02453	0.01808	0.17983	0.09878	0.05779	0.03634	0.02454
	64K	0.06595	0.03834	0.02381	0.01614	0.01194	0.00914	0.12073	0.06595	0.03834	0.02381	0.01614	0.01194	0.12073	0.06595	0.03834	0.02382	0.01614
	128K	0.03887	0.02266	0.01409	0.00937	0.00711	0.00553	0.07067	0.03887	0.02266	0.01409	0.00937	0.00711	0.07067	0.03887	0.02266	0.01409	0.00937
	256K	0.02481	0.01441	0.00881	0.00585	0.00442	0.00345	0.04543	0.02481	0.01441	0.00881	0.00585	0.00442	0.04543	0.02481	0.01441	0.00881	0.00585
	512K	0.01189	0.00682	0.00412	0.00266	0.00187	0.00135	0.02167	0.01189	0.00682	0.00412	0.00266	0.00187	0.02167	0.01189	0.00682	0.00412	0.00266
2	4K	0.19488	0.11514	0.07165	0.04744	0.03434	0.02737	0.35516	0.19488	0.11516	0.07167	0.04746	0.03436	0.35519	0.19501	0.11536	0.07197	0.04767
	8K	0.13203	0.07795	0.04934	0.03334	0.02426	0.01847	0.23855	0.13204	0.07796	0.04935	0.03338	0.02430	0.23857	0.13211	0.07810	0.04958	0.03357
	16K	0.08273	0.04852	0.03065	0.02116	0.01567	0.01200	0.14999	0.08274	0.04853	0.03066	0.02117	0.01569	0.14999	0.08276	0.04857	0.03074	0.02133
	32K	0.04933	0.02821	0.01713	0.01138	0.00847	0.00670	0.09081	0.04934	0.02821	0.01713	0.01138	0.00847	0.09082	0.04935	0.02823	0.01715	0.01142
	64K	0.02857	0.01635	0.00988	0.00643	0.00478	0.00376	0.05254	0.02857	0.01635	0.00988	0.00643	0.00479	0.05254	0.02857	0.01637	0.00990	0.00645
	128K	0.01746	0.00982	0.00580	0.00368	0.00256	0.00192	0.03244	0.01746	0.00982	0.00580	0.00368	0.00256	0.03244	0.01746	0.00983	0.00581	0.00369
	256K	0.00878	0.00502	0.00301	0.00192	0.00138	0.00103	0.01616	0.00878	0.00502	0.00301	0.00192	0.00138	0.01616	0.00878	0.00503	0.00301	0.00192
	512K	0.00448	0.00259	0.00156	0.00099	0.00069	0.00049	0.00812	0.00448	0.00259	0.00156	0.00099	0.00069	0.00812	0.00448	0.00259	0.00156	0.00099
3	4K	0.15322	0.09070	0.05662	0.03811	0.02711	0.02084	0.27490	0.15329	0.09083	0.05676	0.03828	0.02731	0.27534	0.15385	0.09172	0.05781	0.03922
	8K	0.09776	0.05863	0.03710	0.02557	0.01874	0.01438	0.17580	0.09778	0.05870	0.03719	0.02573	0.01886	0.17592	0.09809	0.05931	0.03792	0.02641
	16K	0.05676	0.03270	0.02027	0.01349	0.01008	0.00815	0.10390	0.05677	0.03272	0.02030	0.01355	0.01017	0.10395	0.05691	0.03294	0.02057	0.01399
	32K	0.03623	0.02041	0.01221	0.00780	0.00551	0.00423	0.06719	0.03624	0.02042	0.01223	0.00782	0.00554	0.06723	0.03631	0.02052	0.01232	0.00797
	64K	0.01910	0.01086	0.00650	0.00412	0.00285	0.00215	0.03535	0.01910	0.01086	0.00652	0.00414	0.00286	0.03536	0.01913	0.01093	0.00660	0.00421
	128K	0.01234	0.00684	0.00396	0.00246	0.00167	0.00119	0.02320	0.01234	0.00684	0.00396	0.00246	0.00167	0.02320	0.01234	0.00685	0.00397	0.00248
	256K	0.00514	0.00294	0.00176	0.00112	0.00078	0.00058	0.00940	0.00514	0.00294	0.00176	0.00112	0.00078	0.00940	0.00514	0.00294	0.00177	0.00113
	512K	0.00280	0.00163	0.00098	0.00063	0.00042	0.00029	0.00505	0.00280	0.00163	0.00099	0.00063	0.00042	0.00505	0.00280	0.00163	0.00099	0.00063
4	4K	0.12866	0.07804	0.04982	0.03331	0.02388	0.01778	0.22892	0.12906	0.07854	0.05036	0.03393	0.02437	0.23110	0.13171	0.08090	0.05275	0.03563
	8K	0.07337	0.04322	0.02866	0.01987	0.01503	0.01179	0.13341	0.07349	0.04347	0.02905	0.02032	0.01545	0.13399	0.07448	0.04499	0.03057	0.02176
	16K	0.04582	0.02600	0.01587	0.01046	0.00764	0.00610	0.08469	0.04588	0.02611	0.01603	0.01065	0.00791	0.08494	0.04633	0.02674	0.01673	0.01159
	32K	0.02631	0.01495	0.00893	0.00573	0.00402	0.00303	0.04869	0.02636	0.01501	0.00900	0.00578	0.00411	0.04889	0.02663	0.01528	0.00922	0.00613
	64K	0.01563	0.00877	0.00518	0.00327	0.00221	0.00158	0.02916	0.01564	0.00879	0.00523	0.00333	0.00225	0.02921	0.01574	0.00897	0.00544	0.00349
	128K	0.00801	0.00446	0.00261	0.00162	0.00112	0.00082	0.01496	0.00801	0.00447	0.00261	0.00162	0.00113	0.01498	0.00804	0.00450	0.00265	0.00169
	256K	0.00381	0.00219	0.00132	0.00083	0.00055	0.00039	0.00692	0.00381	0.00219	0.00132	0.00083	0.00056	0.00693	0.00383	0.00222	0.00134	0.00085
	512K	0.00206	0.00120	0.00073	0.00047	0.00031	0.00021	0.00368	0.00206	0.00120	0.00073	0.00047	0.00032	0.00369	0.00207	0.00122	0.00075	0.00048
5	4K	0.10836	0.06665	0.04342	0.02974	0.02140	0.01604	0.19332	0.10995	0.06807	0.04497	0.03111	0.02243	0.19885	0.11551	0.07263	0.04976	0.03412
	8K	0.06043	0.03487	0.02187	0.01538	0.01190	0.00965	0.11091	0.06095	0.03562	0.02291	0.01643	0.01301	0.11262	0.06325	0.03852	0.02557	0.01909
	16K	0.03885	0.02193	0.01320	0.00857	0.00624	0.00493	0.07217	0.03909	0.02226	0.01361	0.00903	0.00684	0.07296	0.04010	0.02345	0.01482	0.01042
	32K	0.02105	0.01199	0.00716	0.00454	0.00315	0.00238	0.03893	0.02123	0.01220	0.00734	0.00468	0.00336	0.03956	0.02193	0.01273	0.00774	0.00522
	64K	0.01390	0.00766	0.00444	0.00279	0.00188	0.00132	0.02627	0.01395	0.00773	0.00457	0.00293	0.00198	0.02643	0.01419	0.00809	0.00495	0.00322
	128K	0.00600	0.00336	0.00197	0.00121	0.00082	0.00061	0.01113	0.00602	0.00338	0.00199	0.00124	0.00086	0.01121	0.00610	0.00346	0.00206	0.00134
	256K	0.00311	0.00180	0.00108	0.00068	0.00045	0.00031	0.00562	0.00312	0.00181	0.00110	0.00070	0.00046	0.00567	0.00318	0.00187	0.00115	0.00074
	512K	0.00160	0.00094	0.00058	0.00037	0.00025	0.00017	0.00285	0.00160	0.00095	0.00059	0.00038	0.00026	0.00287	0.00163	0.00098	0.00062	0.00041
6	4K	0.09148	0.05635	0.03704	0.02553	0.01935	0.01483	0.16541	0.09654	0.06054	0.04024	0.02875	0.02155	0.17764	0.10723	0.06793	0.04629	0.03350
	8K	0.05321	0.03063	0.01909	0.01310	0.00986	0.00832	0.09883	0.05461	0.03222	0.02106	0.01483	0.01179	0.10244	0.05839	0.03641	0.02444	0.01822
	16K	0.03400	0.01916	0.01147	0.00741	0.00534	0.00425	0.06368	0.03464	0.01987	0.01227	0.00821	0.00629	0.06537	0.03634	0.02166	0.01385	0.00994
	32K	0.01833	0.01047	0.00630	0.00398	0.00269	0.00205	0.03423	0.01884	0.01097	0.00666	0.00425	0.00305	0.03555	0.02005	0.01179	0.00723	0.00493
	64K	0.01278	0.00696	0.00397	0.00247	0.00168	0.00118	0.02440	0.01292	0.00713	0.00422	0.00272	0.00186	0.02476	0.01336	0.00767	0.00474	0.00311
	128K	0.00494	0.00279	0.00164	0.00102	0.00067	0.00049	0.00914	0.00500	0.00285	0.00170	0.00107	0.00074	0.00933	0.00518	0.00301	0.00182	0.00121
	256K	0.00261	0.00152	0.00092	0.00059	0.00039	0.00026	0.00472	0.00265	0.00157	0.00097	0.00062	0.00042	0.00486	0.00279	0.00169	0.00106	0.00069
	512K	0.00130	0.00076	0.00047	0.00031	0.00021	0.00015	0.00233	0.00132	0.00079	0.00050	0.00033	0.00023	0.00240	0.00139	0.00086	0.00055	0.00038
7	4K	0.08036	0.04938	0.03323	0.02387	0.01821	0.01413	0.14903	0.08896	0.05695	0.03860	0.02782	0.02120	0.16621	0.10295	0.06645	0.04570	0.03320
	8K	0.04875	0.02817	0.01752	0.01213	0.00918	0.00763	0.09187	0.05144	0.03066	0.02032	0.01434	0.01157	0.09742	0.05626	0.03569	0.02404	0.01809
	16K	0.03130	0.01761	0.01054	0.00685	0.00489	0.00391	0.05950	0.03254	0.01881	0.01172	0.00787	0.00608	0.06212	0.03486	0.02098	0.01349	0.00974
	32K	0.01652	0.00949	0.00576	0.00367	0.00246	0.00184	0.03132	0.01747	0.01030	0.00631	0.00405	0.00291	0.03331	0.01905	0.01130	0.00697	0.00478
	64K	0.01025	0.00563	0.00327	0.00207	0.00146	0.00106	0.01960	0.01053	0.00594	0.00362	0.00241	0.00171	0.02020	0.01113	0.00659	0.00422	0.00285
	128K	0.00422	0.00242	0.00144	0.00090	0.00059	0.00043	0.00786	0.00439	0.00255	0.00155	0.00098	0.00069	0.00824	0.00468	0.00277	0.00171	0.00115
	256K	0.00225	0.00133	0.00082	0.00053	0.00035	0.00024	0.00414	0.00238	0.00143	0.00090	0.00059	0.00040	0.00443	0.00261	0.00161	0.00102	0.00067
	512K	0.00111	0.00066	0.00041	0.00027	0.00019	0.00013	0.00203	0.00117	0.00071	0.00046	0.00031	0.00022	0.00217	0.00129	0.00081	0.00053	0.00036
8	4K	0.07448	0.04660	0.03230	0.02291	0.01753	0.01396	0.14122	0.08517	0.05633	0.03830	0.02757	0.02118	0.16081	0.10220	0.06630	0.04557	0.03319
	8K	0.04662	0.02712	0.01694	0.01181	0.00896	0.00752	0.08902	0.05024	0.03021	0.02013	0.01420	0.01156	0.09563	0.05562	0.03546	0.02393	0.01808
	16K	0.02739	0.01576	0.00964	0.00643	0.00467	0.00383	0.05244	0.02932	0.01737	0.01112	0.00766	0.00604	0.05599	0.03219	0.01990	0.01313	0.00973
	32K	0.01582	0.00914	0.00559	0.00357	0.00239	0.00180	0.03040	0.01709	0.01013	0.00623	0.00400	0.00288	0.03273	0.01882	0.01120	0.00691	0.00475
	64K	0.00827	0.00465	0.00277	0.00182	0.00132	0.00099	0.01587	0.00872	0.00506	0.00321	0.00221	0.00162	0.01670	0.00946	0.00581	0.00384	0.00268
	128K	0.00388	0.00225	0.00136	0.00086	0.00057	0.00041	0.00733	0.00416	0.00245	0.00150	0.00096	0.00068	0.00785	0.00451	0.00270	0.00167	0.00113
	256K	0.00207	0.00124	0.00077	0.00051	0.00034	0.00023	0.00390	0.00228	0.00139	0.00089	0.00058	0.00039	0.00430	0.00256	0.00160	0.00102	0.00067
	512K	0.00102	0.00061	0.00038	0.00026	0.00018	0.00013	0.00192	0.00112	0.00069	0.00045	0.00030	0.00021	0.00210	0.00127	0.00081	0.00053	0.00036

Table 6: Instruction miss ratios, real cache capacity is $\frac{Depth}{8} * Cache\ Size$.

Miss Ratios for 8–Way Set–Associative Data Sector Pool Cache																		
Depth	Cache (Bytes)	Sector/Subsector Ratio=2						Sector/Subsector Ratio=4						Sector/Subsector Ratio=8				
		Sector Size (bytes)						Sector Size (bytes)						Sector Size (bytes)				
		16	32	64	128	256	512	16	32	64	128	256	512	32	64	128	256	512
1	4K	0.25510	0.22250	0.22115	0.25050	0.30202	0.37590	0.38934	0.25515	0.22292	0.22151	0.25120	0.30306	0.38963	0.25702	0.22469	0.22401	0.25588
	8K	0.20734	0.17510	0.16820	0.17937	0.21170	0.26430	0.32716	0.20736	0.17514	0.16828	0.17952	0.21212	0.32731	0.20762	0.17569	0.16909	0.18116
	16K	0.16606	0.13404	0.12375	0.13167	0.14802	0.18360	0.26943	0.16609	0.13406	0.12379	0.13177	0.14820	0.26951	0.16621	0.13425	0.12541	0.13261
	32K	0.13105	0.10270	0.09249	0.09653	0.10677	0.12888	0.21685	0.13108	0.10272	0.09252	0.09655	0.10701	0.21709	0.13115	0.10282	0.09265	0.09812
	64K	0.09893	0.07286	0.06042	0.06024	0.06575	0.07784	0.16820	0.09895	0.07288	0.06045	0.06026	0.06578	0.16826	0.09907	0.07297	0.06054	0.06038
	128K	0.07254	0.05070	0.03839	0.03611	0.03773	0.04397	0.12414	0.07254	0.05070	0.03840	0.03611	0.03774	0.12416	0.07256	0.05074	0.03844	0.03616
	256K	0.05817	0.03936	0.02877	0.02605	0.02555	0.02779	0.10138	0.05817	0.03936	0.02877	0.02605	0.02555	0.10138	0.05817	0.03937	0.02878	0.02606
	512K	0.04194	0.02768	0.01990	0.01611	0.01467	0.01465	0.07332	0.04194	0.02768	0.01990	0.01611	0.01467	0.07332	0.04194	0.02768	0.01990	0.01611
2	4K	0.17267	0.13808	0.12539	0.12990	0.15802	0.20674	0.28199	0.17304	0.13927	0.12645	0.13237	0.16233	0.28340	0.17697	0.14356	0.13311	0.14625
	8K	0.13553	0.10436	0.09045	0.09068	0.10172	0.12891	0.22730	0.13570	0.10461	0.09167	0.09148	0.10365	0.22797	0.13683	0.10709	0.09567	0.09680
	16K	0.10815	0.07926	0.06455	0.06283	0.07059	0.08265	0.18615	0.10823	0.07940	0.06474	0.06449	0.07157	0.18644	0.10877	0.08020	0.06884	0.06979
	32K	0.08413	0.05836	0.04642	0.04276	0.04692	0.05500	0.14758	0.08430	0.05843	0.04654	0.04290	0.04862	0.14833	0.08458	0.05881	0.04710	0.04690
	64K	0.06243	0.04027	0.02792	0.02232	0.02036	0.02485	0.11038	0.06250	0.04039	0.02800	0.02241	0.02049	0.11064	0.06296	0.04068	0.02830	0.02282
	128K	0.04654	0.02886	0.01964	0.01460	0.01184	0.01156	0.08205	0.04655	0.02889	0.01967	0.01464	0.01189	0.08215	0.04667	0.02902	0.01981	0.01481
	256K	0.03631	0.02183	0.01437	0.01032	0.00794	0.00684	0.06432	0.03631	0.02183	0.01438	0.01032	0.00795	0.06434	0.03636	0.02188	0.01442	0.01037
	512K	0.03051	0.01788	0.01121	0.00754	0.00538	0.00427	0.05463	0.03051	0.01788	0.01121	0.00754	0.00538	0.05464	0.03052	0.01790	0.01122	0.00755
3	4K	0.13987	0.10933	0.09714	0.09454	0.10845	0.14078	0.23275	0.14106	0.11222	0.09982	0.09989	0.11929	0.23637	0.14831	0.12084	0.11082	0.12264
	8K	0.10948	0.08100	0.06618	0.06753	0.07124	0.08475	0.18661	0.11005	0.08183	0.07116	0.06970	0.07532	0.18829	0.11262	0.08838	0.07918	0.07827
	16K	0.08620	0.06271	0.04976	0.04430	0.05114	0.05696	0.15074	0.08643	0.06316	0.05036	0.04861	0.05356	0.15138	0.08768	0.06495	0.05588	0.05621
	32K	0.07003	0.04581	0.03473	0.03052	0.03045	0.03836	0.12490	0.07053	0.04603	0.03504	0.03094	0.03434	0.12625	0.07115	0.04683	0.03622	0.03618
	64K	0.05437	0.03424	0.02270	0.01635	0.01312	0.01306	0.09732	0.05458	0.03457	0.02290	0.01659	0.01344	0.09793	0.05553	0.03514	0.02343	0.01735
	128K	0.03841	0.02323	0.01556	0.01130	0.00858	0.00723	0.06784	0.03848	0.02332	0.01565	0.01140	0.00871	0.06810	0.03876	0.02360	0.01594	0.01174
	256K	0.03202	0.01890	0.01199	0.00819	0.00592	0.00468	0.05712	0.03205	0.01893	0.01202	0.00822	0.00595	0.05723	0.03219	0.01906	0.01213	0.00835
	512K	0.02725	0.01571	0.00957	0.00614	0.00408	0.00295	0.04928	0.02725	0.01572	0.00958	0.00615	0.00409	0.04931	0.02730	0.01577	0.00962	0.00619
4	4K	0.12171	0.09171	0.07943	0.07820	0.08717	0.10819	0.20485	0.12445	0.09717	0.08762	0.08688	0.10515	0.21166	0.13546	0.11007	0.10225	0.11613
	8K	0.09557	0.06655	0.05235	0.05212	0.05867	0.06707	0.16547	0.09688	0.06840	0.05977	0.06105	0.06547	0.16854	0.10111	0.07762	0.07239	0.07222
	16K	0.07541	0.05095	0.03899	0.03373	0.03766	0.04730	0.13380	0.07593	0.05189	0.04030	0.04005	0.04720	0.13503	0.07801	0.05470	0.04811	0.05203
	32K	0.06090	0.03883	0.02664	0.02090	0.01835	0.02533	0.10907	0.06193	0.03932	0.02733	0.02180	0.02468	0.11110	0.06299	0.04071	0.02926	0.02912
	64K	0.04557	0.02815	0.01901	0.01356	0.01054	0.00951	0.08115	0.04603	0.02882	0.01940	0.01398	0.01111	0.08221	0.04743	0.02969	0.02017	0.01504
	128K	0.03495	0.02090	0.01367	0.00971	0.00723	0.00590	0.06206	0.03514	0.02112	0.01387	0.00992	0.00750	0.06259	0.03567	0.02159	0.01435	0.01050
	256K	0.02967	0.01735	0.01080	0.00715	0.00497	0.00378	0.05319	0.02974	0.01745	0.01088	0.00722	0.00506	0.05346	0.03005	0.01769	0.01110	0.00746
	512K	0.02532	0.01446	0.00867	0.00543	0.00348	0.00237	0.04609	0.02534	0.01450	0.00870	0.00545	0.00350	0.04619	0.02548	0.01462	0.00880	0.00555
5	4K	0.11087	0.07859	0.06887	0.06917	0.07582	0.09532	0.18947	0.11601	0.08962	0.08042	0.08118	0.10047	0.20003	0.12999	0.10509	0.09776	0.11429
	8K	0.08635	0.05807	0.04361	0.04506	0.05175	0.05840	0.15098	0.08871	0.06129	0.05340	0.05642	0.06126	0.15580	0.09473	0.07233	0.06763	0.06958
	16K	0.07052	0.04545	0.03235	0.02748	0.03227	0.04228	0.12657	0.07150	0.04701	0.03447	0.03595	0.04401	0.12854	0.07431	0.05058	0.04382	0.04963
	32K	0.05701	0.03611	0.02367	0.01722	0.01408	0.02073	0.10278	0.05858	0.03700	0.02482	0.01868	0.02251	0.10544	0.06012	0.03897	0.02727	0.02774
	64K	0.03996	0.02439	0.01662	0.01195	0.00917	0.00809	0.07092	0.04076	0.02544	0.01728	0.01257	0.01002	0.07248	0.04250	0.02655	0.01829	0.01386
	128K	0.03294	0.01956	0.01256	0.00872	0.00637	0.00516	0.05882	0.03332	0.01996	0.01292	0.00908	0.00684	0.05972	0.03411	0.02062	0.01359	0.00989
	256K	0.02799	0.01625	0.01001	0.00650	0.00437	0.00322	0.05056	0.02819	0.01646	0.01018	0.00665	0.00455	0.05110	0.02871	0.01686	0.01052	0.00702
	512K	0.02392	0.01357	0.00804	0.00495	0.00310	0.00203	0.04378	0.02400	0.01367	0.00812	0.00501	0.00316	0.04406	0.02431	0.01391	0.00831	0.00518
6	4K	0.10409	0.07258	0.06331	0.06292	0.06966	0.08968	0.18100	0.11165	0.08606	0.07669	0.07795	0.09926	0.19445	0.12733	0.10275	0.09574	0.11388
	8K	0.07947	0.05263	0.03851	0.03925	0.04593	0.05378	0.14035	0.08311	0.05727	0.04966	0.05205	0.05802	0.14702	0.09075	0.06937	0.06470	0.06727
	16K	0.06792	0.04270	0.02882	0.02243	0.02666	0.03677	0.12281	0.06943	0.04485	0.03161	0.03167	0.03983	0.12548	0.07277	0.04886	0.04140	0.04650
	32K	0.05450	0.03459	0.02255	0.01615	0.01296	0.01880	0.09874	0.05651	0.03576	0.02401	0.01806	0.02192	0.10191	0.05839	0.03798	0.02676	0.02750
	64K	0.03753	0.02289	0.01556	0.01123	0.00850	0.00732	0.06696	0.03870	0.02427	0.01649	0.01205	0.00958	0.06899	0.04072	0.02556	0.01769	0.01350
	128K	0.03154	0.01870	0.01188	0.00809	0.00583	0.00469	0.05672	0.03219	0.01930	0.01241	0.00864	0.00650	0.05803	0.03323	0.02011	0.01325	0.00962
	256K	0.02683	0.01551	0.00948	0.00608	0.00402	0.00288	0.04885	0.02724	0.01589	0.00977	0.00633	0.00430	0.04976	0.02800	0.01642	0.01024	0.00682
	512K	0.02283	0.01290	0.00757	0.00461	0.00285	0.00183	0.04207	0.02306	0.01311	0.00774	0.00474	0.00296	0.04264	0.02356	0.01347	0.00803	0.00499
7	4K	0.10061	0.06968	0.06056	0.05953	0.06478	0.08813	0.17731	0.10990	0.08442	0.07541	0.07629	0.09905	0.19243	0.12636	0.10206	0.09507	0.11373
	8K	0.07637	0.05043	0.03667	0.03716	0.04258	0.05017	0.13669	0.08121	0.05612	0.04868	0.05068	0.05669	0.14470	0.08979	0.06876	0.06405	0.06672
	16K	0.06672	0.04184	0.02798	0.02129	0.02483	0.03470	0.12129	0.06872	0.04440	0.03116	0.03094	0.03908	0.12456	0.07238	0.04860	0.04111	0.04619
	32K	0.05197	0.03291	0.02160	0.01550	0.01243	0.01836	0.09436	0.05438	0.03441	0.02338	0.01768	0.02178	0.09807	0.05667	0.03703	0.02628	0.02744
	64K	0.03626	0.02215	0.01505	0.01080	0.00815	0.00695	0.06519	0.03777	0.02378	0.01617	0.01182	0.00941	0.06763	0.04003	0.02521	0.01750	0.01337
	128K	0.03060	0.01815	0.01149	0.00773	0.00553	0.00443	0.05552	0.03156	0.01893	0.01217	0.00844	0.00636	0.05718	0.03277	0.01984	0.01310	0.00952
	256K	0.02604	0.01505	0.00916	0.00581	0.00380	0.00269	0.04781	0.02672	0.01561	0.00958	0.00618	0.00419	0.04908	0.02767	0.01623	0.01012	0.00674
	512K	0.02205	0.01243	0.00727	0.00438	0.00269	0.00172	0.04100	0.02249	0.01280	0.00754	0.00459	0.00287	0.04189	0.02317	0.01325	0.00790	0.00491
8	4K	0.09921	0.06865	0.05963	0.05873	0.06414	0.08764	0.17575	0.10938	0.08395	0.07511	0.07611	0.09898	0.19172	0.12609	0.10190	0.09496	0.11370
	8K	0.07525	0.04973	0.03615	0.03661	0.04170	0.04970	0.13558	0.08075	0.05586	0.04849	0.05033	0.05658	0.14422	0.08963	0.06865	0.06395	0.06668
	16K	0.06620	0.04153	0.02776	0.02099	0.02434	0.03427	0.12067	0.06849	0.04428	0.03107	0.03077	0.03899	0.12427	0.07228	0.04854	0.04106	0.04617
	32K	0.04825	0.03060	0.02048	0.01487	0.01205	0.01821	0.08735	0.05136	0.03258	0.02244	0.01721	0.02171	0.09222	0.05439	0.03532	0.02549	0.02734
	64K	0.03566	0.02180	0.01484	0.01059	0.00800	0.00682	0.06445	0.03738	0.02358	0.01606	0.01173	0.00934	0.06712	0.03978	0.02508	0.01744	0.01333
	128K	0.03017	0.01788	0.01133	0.00760	0.00542	0.00434	0.05503	0.03130	0.01878	0.01208	0.00839	0.00631	0.05687	0.03260	0.01974	0.01306	0.00950
	256K	0.02566	0.01484	0.00903	0.00570	0.00372	0.00262	0.04742	0.02653	0.01552	0.00951	0.00613	0.00416	0.04886	0.02757	0.01617	0.01009	0.00672
	512K	0.02163	0.01219	0.00712	0.00427	0.00262	0.00167	0.04055	0.02225	0.01268	0.00746	0.00454	0.00284	0.04162	0.02305	0.01319	0.00787	0.00488

Table 7: Data miss ratios, real cache capacity is $\frac{Depth}{8} * Cache\ Size$.

Miss Ratios of Normal 8–Way Set–Associative Caches																		
Cache Size	Sector Size (bytes)																	
	Unified Cache						Instruction Cache						Data Cache					
	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	0.0651	0.0476	0.0395	0.0367	0.0378	0.0451	0.0405	0.0265	0.0193	0.0145	0.0115	0.0096	0.0613	0.0449	0.0448	0.0483	0.0555	0.0798
8K	0.0418	0.0279	0.0211	0.0207	0.0219	0.0258	0.0251	0.0151	0.0098	0.0073	0.0058	0.0052	0.0459	0.0316	0.0247	0.0288	0.0360	0.0447
16K	0.0286	0.0186	0.0132	0.0105	0.0118	0.0149	0.0147	0.0087	0.0055	0.0039	0.0029	0.0026	0.0399	0.0256	0.0183	0.0147	0.0201	0.0310
32K	0.0176	0.0113	0.0078	0.0060	0.0051	0.0072	0.0084	0.0050	0.0032	0.0021	0.0015	0.0012	0.0283	0.0190	0.0133	0.0101	0.0088	0.0158
64K	0.0114	0.0073	0.0051	0.0037	0.0029	0.0026	0.0044	0.0025	0.0016	0.0011	0.0008	0.0006	0.0206	0.0134	0.0096	0.0070	0.0056	0.0051
128K	0.0078	0.0048	0.0031	0.0021	0.0016	0.0014	0.0021	0.0013	0.0008	0.0005	0.0003	0.0003	0.0171	0.0106	0.0070	0.0048	0.0036	0.0031
256K	0.0060	0.0037	0.0023	0.0015	0.0010	0.0007	0.0011	0.0007	0.0004	0.0003	0.0002	0.0001	0.0143	0.0086	0.0053	0.0034	0.0023	0.0017
512K	0.0047	0.0028	0.0017	0.0010	0.0007	0.0004	0.0006	0.0003	0.0002	0.0001	0.0001	0.0001	0.0118	0.0068	0.0040	0.0024	0.0015	0.0010

Miss Ratios of Normal 4–Way Set–Associative Caches																		
Cache Size	Sector Size (bytes)																	
	Unified Cache						Instruction Cache						Data Cache					
	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	0.0710	0.0512	0.0428	0.0401	0.0405	0.0506	0.0432	0.0286	0.0198	0.0150	0.0118	0.0097	0.0664	0.0521	0.0506	0.0579	0.0658	0.0864
8K	0.0455	0.0320	0.0251	0.0237	0.0255	0.0293	0.0260	0.0159	0.0104	0.0076	0.0061	0.0054	0.0509	0.0388	0.0335	0.0352	0.0458	0.0537
16K	0.0294	0.0198	0.0147	0.0129	0.0137	0.0171	0.0149	0.0089	0.0057	0.0040	0.0030	0.0026	0.0387	0.0266	0.0208	0.0182	0.0228	0.0336
32K	0.0179	0.0116	0.0081	0.0063	0.0056	0.0074	0.0088	0.0052	0.0033	0.0022	0.0016	0.0013	0.0281	0.0189	0.0135	0.0104	0.0094	0.0149
64K	0.0115	0.0074	0.0051	0.0038	0.0031	0.0029	0.0045	0.0026	0.0016	0.0011	0.0008	0.0006	0.0209	0.0136	0.0097	0.0072	0.0059	0.0055
128K	0.0080	0.0050	0.0033	0.0023	0.0017	0.0015	0.0022	0.0013	0.0008	0.0006	0.0004	0.0003	0.0173	0.0108	0.0071	0.0050	0.0038	0.0032
256K	0.0062	0.0037	0.0024	0.0015	0.0010	0.0008	0.0012	0.0007	0.0005	0.0003	0.0002	0.0001	0.0145	0.0087	0.0054	0.0035	0.0024	0.0018
512K	0.0048	0.0028	0.0017	0.0011	0.0007	0.0005	0.0007	0.0004	0.0003	0.0002	0.0001	0.0001	0.0119	0.0069	0.0041	0.0025	0.0016	0.0010

Miss Ratios of Normal 2–Way Set–Associative Caches																		
Cache Size	Sector Size (bytes)																	
	Unified Cache						Instruction Cache						Data Cache					
	16	32	64	128	256	512	16	32	64	128	256	512	16	32	64	128	256	512
4K	0.0803	0.0590	0.0488	0.0460	0.0491	0.0624	0.0485	0.0306	0.0211	0.1109	0.0120	0.0102	0.0793	0.0646	0.0628	0.1595	0.0826	0.1118
8K	0.0533	0.0386	0.0313	0.0298	0.0313	0.0382	0.0281	0.0171	0.0113	0.0084	0.0067	0.0061	0.0584	0.0464	0.0428	0.0469	0.0550	0.0688
16K	0.0325	0.0225	0.0173	0.0155	0.0157	0.0210	0.0160	0.0097	0.0063	0.0047	0.0037	0.0033	0.0404	0.0280	0.0223	0.0204	0.0249	0.0395
32K	0.0201	0.0134	0.0098	0.0080	0.0074	0.0095	0.0096	0.0057	0.0036	0.0025	0.0019	0.0016	0.0291	0.0198	0.0147	0.0119	0.0116	0.0176
64K	0.0122	0.0081	0.0058	0.0045	0.0039	0.0040	0.0044	0.0027	0.0018	0.0013	0.0010	0.0008	0.0221	0.0145	0.0104	0.0080	0.0069	0.0070
128K	0.0086	0.0055	0.0037	0.0027	0.0022	0.0020	0.0023	0.0014	0.0009	0.0007	0.0005	0.0004	0.0182	0.0114	0.0076	0.0054	0.0043	0.0039
256K	0.0067	0.0041	0.0026	0.0017	0.0012	0.0010	0.0014	0.0009	0.0006	0.0004	0.0003	0.0002	0.0150	0.0090	0.0057	0.0037	0.0027	0.0021
512K	0.0052	0.0031	0.0019	0.0012	0.0008	0.0005	0.0009	0.0005	0.0003	0.0002	0.0002	0.0001	0.0126	0.0073	0.0043	0.0026	0.0017	0.0012

Table 8: Miss ratios for normal set–associative caches.

Best Sector Pool Unified Cache Configurations			
5 Cycle Overhead		15 Cycle Overhead	
4.25KB **1.1847** 4K 32/16 D7	140.56KB **0.1247** 256K 128/32 D4	4.25KB **1.9745** 4K 32/16 D7	172.56KB **0.1585** 256K 128/32 D5
4.69KB **1.1556** 4K 32/16 D8	144.56KB **0.1216** 256K 128/16 D4	4.43KB **1.9271** 8K 128/32 D4	181.12KB **0.1561** 256K 64/32 D5
4.86KB **1.0856** 8K 64/16 D4	146.62KB **0.1199** 128K 32/32 D8	4.66KB **1.6672** 4K 32/32 D8	198.28KB **0.1520** 256K 256/64 D6
5.32KB **0.9759** 4K 16/16 D8	165.25KB **0.1163** 128K 16/16 D8	5.32KB **1.6266** 4K 16/16 D8	202.56KB **0.1459** 256K 128/64 D6
5.86KB **0.9295** 8K 64/16 D5	172.56KB **0.1132** 256K 128/32 D5	5.74KB **1.5034** 8K 64/32 D5	213.12KB **0.1452** 256K 64/32 D6
6.48KB **0.8838** 8K 32/16 D5	176.56KB **0.1129** 256K 128/16 D5	6.48KB **1.4731** 8K 32/16 D5	234.56KB **0.1384** 256K 128/64 D7
6.86KB **0.8532** 8K 64/16 D6	181.12KB **0.1115** 256K 64/32 D5	6.74KB **1.2896** 8K 64/32 D6	264.16KB **0.1319** 512K 512/64 D4
7.48KB **0.7762** 8K 32/16 D6	185.12KB **0.1100** 256K 64/16 D5	7.74KB **1.1829** 8K 64/32 D7	268.31KB **0.1299** 512K 256/64 D4
8.48KB **0.7132** 8K 32/16 D7	202.25KB **0.1086** 256K 32/16 D5	8.64KB **1.1621** 8K 64/64 D8	274.12KB **0.1263** 256K 64/64 D8
9.29KB **0.6966** 8K 32/32 D8	204.56KB **0.1071** 256K 128/32 D6	8.68KB **1.1530** 8K 64/32 D8	328.16KB **0.1217** 512K 512/64 D5
9.35KB **0.6915** 8K 32/16 D8	213.12KB **0.1037** 256K 64/32 D6	8.85KB **1.1528** 16K 128/32 D4	332.31KB **0.1178** 512K 256/64 D5
10.58KB **0.6267** 8K 16/16 D8	234.25KB **0.1015** 256K 32/16 D6	9.29KB **0.9752** 8K 32/32 D8	340.62KB **0.1163** 512K 128/64 D5
11.70KB **0.6022** 16K 64/16 D5	245.12KB **0.0990** 256K 64/32 D7	11.45KB **0.9257** 16K 64/32 D5	394.31KB **0.1161** 512K 256/128 D6
12.89KB **0.5829** 16K 32/16 D5	266.25KB **0.0970** 256K 32/16 D7	12.85KB **0.9068** 16K 128/32 D6	396.31KB **0.1108** 512K 256/64 D6
13.45KB **0.5798** 16K 64/32 D6	272.31KB **0.0948** 512K 256/32 D4	13.45KB **0.8117** 16K 64/32 D6	404.62KB **0.1075** 512K 128/64 D6
13.70KB **0.5542** 16K 64/16 D6	280.62KB **0.0937** 512K 128/32 D4	15.45KB **0.7561** 16K 64/32 D7	460.31KB **0.1073** 512K 256/64 D7
14.89KB **0.5191** 16K 32/16 D6	292.25KB **0.0913** 256K 32/32 D8	17.26KB **0.7236** 16K 64/64 D8	468.62KB **0.1018** 512K 128/64 D7
16.89KB **0.4838** 16K 32/16 D7	328.50KB **0.0907** 256K 16/16 D8	17.66KB **0.7087** 32K 128/32 D4	529.62KB **0.0993** 512K 128/128 D8
18.16KB **0.4636** 32K 128/16 D4	336.31KB **0.0882** 512K 256/32 D5	18.52KB **0.6505** 16K 32/32 D8	547.25KB **0.0929** 512K 64/64 D8
19.33KB **0.4463** 32K 64/16 D4	344.62KB **0.0860** 512K 128/32 D5	21.41KB **0.6384** 32K 128/64 D5	
21.03KB **0.4288** 16K 16/16 D8	361.25KB **0.0849** 512K 64/32 D5	21.66KB **0.5950** 32K 128/32 D5	
21.66KB **0.4250** 32K 128/32 D5	400.31KB **0.0848** 512K 256/32 D6	22.83KB **0.5670** 32K 64/32 D5	
22.16KB **0.4063** 32K 128/16 D5	408.62KB **0.0813** 512K 128/32 D6	25.41KB **0.5611** 32K 128/64 D6	
22.83KB **0.4050** 32K 64/32 D5	425.25KB **0.0790** 512K 64/32 D6	25.66KB **0.5442** 32K 128/32 D6	
23.33KB **0.3774** 32K 64/16 D5	472.62KB **0.0789** 512K 128/32 D7	25.88KB **0.5316** 64K 256/32 D3	
25.66KB **0.3678** 32K 32/16 D5	489.25KB **0.0752** 512K 64/32 D7	26.83KB **0.4990** 32K 64/32 D6	
26.83KB **0.3564** 32K 64/32 D6	549.25KB **0.0735** 512K 64/32 D8	30.83KB **0.4603** 32K 64/32 D7	
27.33KB **0.3426** 32K 64/16 D6	582.50KB **0.0692** 512K 32/32 D8	34.13KB **0.4365** 64K 256/32 D4	
29.66KB **0.3258** 32K 32/16 D6		34.45KB **0.4311** 32K 64/64 D8	
33.66KB **0.3019** 32K 32/16 D7		35.27KB **0.4158** 64K 128/32 D4	
35.27KB **0.2970** 64K 128/32 D4		36.91KB **0.3972** 32K 32/32 D8	
36.27KB **0.2824** 64K 128/16 D4		42.13KB **0.3933** 64K 256/32 D5	
38.53KB **0.2731** 64K 64/16 D4		42.77KB **0.3793** 64K 128/64 D5	
41.81KB **0.2641** 32K 16/16 D8		43.27KB **0.3631** 64K 128/32 D5	
43.27KB **0.2594** 64K 128/32 D5		45.53KB **0.3485** 64K 64/32 D5	
44.27KB **0.2528** 64K 128/16 D5		50.77KB **0.3470** 64K 128/64 D6	
45.53KB **0.2490** 64K 64/32 D5		51.27KB **0.3425** 64K 128/32 D6	
46.53KB **0.2385** 64K 64/16 D5		51.70KB **0.3367** 128K 256/32 D3	
51.06KB **0.2304** 64K 32/16 D5		53.53KB **0.3190** 64K 64/32 D6	
53.53KB **0.2279** 64K 64/32 D6		61.53KB **0.3008** 64K 64/32 D7	
54.53KB **0.2239** 64K 64/16 D6		67.20KB **0.2891** 128K 256/64 D4	
55.41KB **0.2212** 128K 128/16 D3		68.20KB **0.2718** 128K 256/32 D4	
59.06KB **0.2112** 64K 32/16 D6		70.41KB **0.2620** 128K 128/32 D4	
67.06KB **0.1975** 64K 32/16 D7		73.56KB **0.2542** 64K 32/32 D8	
68.20KB **0.1942** 128K 256/32 D4		83.20KB **0.2493** 128K 256/64 D5	
70.41KB **0.1871** 128K 128/32 D4		84.20KB **0.2419** 128K 256/32 D5	
72.41KB **0.1778** 128K 128/16 D4		85.41KB **0.2389** 128K 128/64 D5	
76.81KB **0.1741** 128K 64/16 D4		86.41KB **0.2265** 128K 128/32 D5	
83.13KB **0.1710** 64K 16/16 D8		90.81KB **0.2206** 128K 64/32 D5	
86.41KB **0.1618** 128K 128/32 D5		101.41KB **0.2112** 128K 128/64 D6	
88.41KB **0.1573** 128K 128/16 D5		102.41KB **0.2078** 128K 128/32 D6	
92.81KB **0.1513** 128K 64/16 D5		106.81KB **0.1978** 128K 64/32 D6	
101.63KB **0.1485** 128K 32/16 D5		117.41KB **0.1960** 128K 128/64 D7	
102.41KB **0.1484** 128K 128/32 D6		122.81KB **0.1860** 128K 64/32 D7	
104.41KB **0.1472** 128K 128/16 D6		134.28KB **0.1819** 256K 256/64 D4	
106.81KB **0.1413** 128K 64/32 D6		136.28KB **0.1777** 256K 256/32 D4	
108.81KB **0.1392** 128K 64/16 D6		137.31KB **0.1725** 128K 64/64 D8	
117.62KB **0.1343** 128K 32/16 D6		146.62KB **0.1679** 128K 32/32 D8	
122.81KB **0.1328** 128K 64/32 D7		166.28KB **0.1620** 256K 256/64 D5	
133.62KB **0.1264** 128K 32/16 D7		170.56KB **0.1588** 256K 128/64 D5	

Table 9: The best sector pool unified cache configurations for a given bit budget, 5 and 15 cycle overhead. Each entry contains the total bits to implement the cache (in Kbytes), the additional delay (in bold), the nominal cache size, the sector/subsector size, and the depth (1-8).

Best Sector Pool Instruction Cache Configurations			
5 Cycle Overhead		15 Cycle Overhead	
4.22KB **0.7407** 4K 32/16 D7	73.31KB **0.0632** 64K 32/32 D8	4.20KB **1.1175** 8K 256/64 D4	97.85KB **0.0665** 128K 512/64 D6
4.40KB **0.7263** 8K 128/32 D4	81.85KB **0.0604** 128K 512/64 D5	4.25KB **1.0701** 8K 256/32 D4	98.58KB **0.0637** 128K 256/128 D6
4.49KB **0.6749** 8K 128/16 D4	82.95KB **0.0557** 128K 256/64 D5	4.32KB **1.0603** 4K 64/64 D8	98.95KB **0.0587** 128K 256/64 D6
4.64KB **0.6636** 4K 32/32 D8	83.70KB **0.0516** 128K 256/32 D5	4.40KB **1.0168** 8K 128/32 D4	101.16KB **0.0560** 128K 128/64 D6
4.80KB **0.6520** 8K 64/16 D4	85.91KB **0.0497** 128K 128/32 D5	4.64KB **0.9291** 4K 32/32 D8	114.95KB **0.0540** 128K 256/64 D7
5.25KB **0.6393** 8K 256/32 D5	90.31KB **0.0491** 128K 64/32 D5	5.20KB **0.9039** 8K 256/64 D5	117.16KB **0.0496** 128K 128/64 D7
5.29KB **0.6077** 4K 16/16 D8	98.95KB **0.0481** 128K 256/64 D6	5.25KB **0.8950** 8K 256/32 D5	131.64KB **0.0469** 256K 512/64 D4
5.40KB **0.5729** 8K 128/32 D5	99.70KB **0.0456** 128K 256/32 D6	5.35KB **0.8461** 8K 128/64 D5	133.78KB **0.0459** 256K 256/64 D4
5.71KB **0.5468** 8K 64/32 D5	101.91KB **0.0425** 128K 128/32 D6	5.40KB **0.8020** 8K 128/32 D5	137.06KB **0.0427** 128K 64/64 D8
5.80KB **0.5343** 8K 64/16 D5	106.31KB **0.0410** 128K 64/32 D6	5.71KB **0.7655** 8K 64/32 D5	163.64KB **0.0405** 256K 512/64 D5
6.40KB **0.5265** 8K 128/32 D6	117.16KB **0.0405** 128K 128/64 D7	6.35KB **0.7206** 8K 128/64 D6	165.78KB **0.0382** 256K 256/64 D5
6.41KB **0.5231** 8K 32/16 D5	117.91KB **0.0387** 128K 128/32 D7	6.42KB **0.7201** 16K 256/32 D3	170.06KB **0.0375** 256K 128/64 D5
6.42KB **0.5143** 16K 256/32 D3	122.31KB **0.0360** 128K 64/32 D7	6.71KB **0.6683** 8K 64/32 D6	197.03KB **0.0371** 256K 256/128 D6
6.71KB **0.4773** 8K 64/32 D6	135.28KB **0.0335** 256K 256/32 D4	7.35KB **0.6672** 8K 128/64 D7	197.78KB **0.0342** 256K 256/64 D6
7.41KB **0.4594** 8K 32/16 D6	139.56KB **0.0330** 256K 128/32 D4	7.71KB **0.6133** 8K 64/32 D7	202.06KB **0.0323** 256K 128/64 D6
7.71KB **0.4381** 8K 64/32 D7	146.12KB **0.0313** 128K 32/32 D8	8.39KB **0.5856** 16K 256/64 D4	212.12KB **0.0323** 256K 64/32 D6
8.41KB **0.4225** 8K 32/16 D7	167.28KB **0.0287** 256K 256/32 D5	8.49KB **0.5855** 16K 256/32 D4	234.06KB **0.0291** 256K 128/64 D7
8.49KB **0.4182** 16K 256/32 D4	171.56KB **0.0274** 256K 128/32 D5	8.63KB **0.5414** 8K 64/64 D8	244.12KB **0.0286** 256K 64/32 D7
8.79KB **0.4008** 16K 128/32 D4	180.12KB **0.0270** 256K 64/32 D5	9.26KB **0.5282** 8K 32/32 D8	263.16KB **0.0266** 512K 512/64 D4
9.26KB **0.3773** 8K 32/32 D8	199.28KB **0.0264** 256K 256/32 D6	10.39KB **0.4967** 16K 256/64 D5	267.31KB **0.0259** 512K 256/64 D4
10.49KB **0.3706** 16K 256/32 D5	203.56KB **0.0242** 256K 128/32 D6	10.69KB **0.4712** 16K 128/64 D5	273.62KB **0.0243** 256K 64/64 D8
10.79KB **0.3402** 16K 128/32 D5	212.12KB **0.0231** 256K 64/32 D6	11.38KB **0.4621** 16K 64/32 D5	291.25KB **0.0241** 256K 32/32 D8
11.38KB **0.3301** 16K 64/32 D5	232.25KB **0.0228** 256K 32/16 D6	12.39KB **0.4518** 16K 256/64 D6	327.16KB **0.0226** 512K 512/64 D5
12.77KB **0.3289** 16K 32/16 D5	235.56KB **0.0226** 256K 128/32 D7	12.42KB **0.4386** 32K 512/64 D3	331.31KB **0.0210** 512K 256/64 D5
12.79KB **0.3068** 16K 128/32 D6	244.12KB **0.0205** 256K 64/32 D7	12.69KB **0.4078** 16K 128/64 D6	339.62KB **0.0205** 512K 128/64 D5
13.38KB **0.2868** 16K 64/32 D6	264.25KB **0.0199** 256K 32/16 D7	13.38KB **0.4015** 16K 64/32 D6	342.62KB **0.0205** 512K 128/32 D5
15.38KB **0.2635** 16K 64/32 D7	270.31KB **0.0187** 512K 256/32 D4	14.69KB **0.3767** 16K 128/64 D7	359.25KB **0.0202** 512K 64/32 D5
16.77KB **0.2602** 32K 256/64 D4	278.62KB **0.0184** 512K 128/32 D4	15.38KB **0.3689** 16K 64/32 D7	393.81KB **0.0200** 512K 256/128 D6
16.96KB **0.2306** 32K 256/32 D4	284.62KB **0.0182** 512K 128/16 D4	16.48KB **0.3371** 32K 512/64 D4	395.31KB **0.0182** 512K 256/64 D6
17.54KB **0.2251** 32K 128/32 D4	291.25KB **0.0172** 256K 32/32 D8	16.77KB **0.3181** 32K 256/64 D4	403.62KB **0.0170** 512K 128/64 D6
18.45KB **0.2173** 16K 32/32 D8	326.50KB **0.0169** 256K 16/16 D8	17.23KB **0.3034** 16K 64/64 D8	423.25KB **0.0165** 512K 64/32 D6
20.77KB **0.2108** 32K 256/64 D5	334.31KB **0.0154** 512K 256/32 D5	20.48KB **0.2869** 32K 512/64 D5	467.62KB **0.0149** 512K 128/64 D7
20.96KB **0.1936** 32K 256/32 D5	342.62KB **0.0146** 512K 128/32 D5	20.77KB **0.2576** 32K 256/64 D5	487.25KB **0.0143** 512K 64/32 D7
21.54KB **0.1835** 32K 128/32 D5	359.25KB **0.0144** 512K 64/32 D5	21.35KB **0.2497** 32K 128/64 D5	529.12KB **0.0142** 512K 128/128 D8
22.70KB **0.1791** 32K 64/32 D5	365.25KB **0.0142** 512K 64/16 D5	24.77KB **0.2340** 32K 256/64 D6	529.62KB **0.0141** 512K 128/64 D8
25.54KB **0.1665** 32K 128/32 D6	398.31KB **0.0139** 512K 256/32 D6	24.82KB **0.2314** 64K 512/64 D3	546.25KB **0.0121** 512K 64/64 D8
25.63KB **0.1650** 64K 256/32 D3	406.62KB **0.0125** 512K 128/32 D6	25.35KB **0.2192** 32K 128/64 D6	580.50KB **0.0118** 512K 32/32 D8
26.70KB **0.1575** 32K 64/32 D6	423.25KB **0.0118** 512K 64/32 D6	29.35KB **0.2020** 32K 128/64 D7	
29.41KB **0.1571** 32K 32/16 D6	462.50KB **0.0115** 512K 32/16 D6	30.70KB **0.2016** 32K 64/32 D7	
30.70KB **0.1440** 32K 64/32 D7	470.62KB **0.0115** 512K 128/32 D7	32.94KB **0.1920** 64K 512/64 D4	
33.41KB **0.1423** 32K 32/16 D7	487.25KB **0.0102** 512K 64/32 D7	33.51KB **0.1829** 64K 256/64 D4	
33.88KB **0.1361** 64K 256/32 D4	526.50KB **0.0098** 512K 32/16 D7	34.39KB **0.1750** 32K 64/64 D8	
35.02KB **0.1308** 64K 128/32 D4	547.25KB **0.0096** 512K 64/32 D8	36.78KB **0.1750** 32K 32/32 D8	
36.78KB **0.1250** 32K 32/32 D8	580.50KB **0.0084** 512K 32/32 D8	41.51KB **0.1612** 64K 256/64 D5	
41.88KB **0.1239** 64K 256/32 D5	649.00KB **0.0083** 512K 16/16 D8	42.64KB **0.1536** 64K 128/64 D5	
43.02KB **0.1142** 64K 128/32 D5		49.51KB **0.1493** 64K 256/64 D6	
45.28KB **0.1111** 64K 64/32 D5		49.60KB **0.1365** 128K 512/64 D3	
50.64KB **0.1109** 64K 128/64 D6		50.64KB **0.1356** 64K 128/64 D6	
50.70KB **0.1106** 128K 256/64 D3		50.70KB **0.1352** 128K 256/64 D3	
51.02KB **0.1056** 64K 128/32 D6		52.91KB **0.1351** 128K 128/64 D3	
51.20KB **0.0992** 128K 256/32 D3		57.51KB **0.1328** 64K 256/64 D7	
53.41KB **0.0990** 128K 128/32 D3		58.64KB **0.1141** 64K 128/64 D7	
57.81KB **0.0989** 128K 64/32 D3		65.48KB **0.1077** 128K 512/128 D4	
58.64KB **0.0934** 64K 128/64 D7		65.85KB **0.0927** 128K 512/64 D4	
59.02KB **0.0905** 64K 128/32 D7		66.95KB **0.0893** 128K 256/64 D4	
61.28KB **0.0816** 64K 64/32 D7		68.66KB **0.0856** 64K 64/64 D8	
65.85KB **0.0759** 128K 512/64 D4		81.48KB **0.0814** 128K 512/128 D5	
66.95KB **0.0730** 128K 256/64 D4		81.85KB **0.0738** 128K 512/64 D5	
67.70KB **0.0662** 128K 256/32 D4		82.95KB **0.0681** 128K 256/64 D5	
69.91KB **0.0653** 128K 128/32 D4		85.16KB **0.0668** 128K 128/64 D5	

Table 10: The best sector pool instruction cache configurations for a given bit budget, 5 and 15 cycle overhead. Each entry contains the total bits to implement the cache (in Kbytes), the additional delay (in bold), the nominal cache size, the sector/subsector size, and the depth (1-8).

Best Sector Pool Data Cache Configurations			
5 Cycle Overhead		15 Cycle Overhead	
4.25KB **1.0451** 4K 32/16 D7	213.12KB **0.2369** 256K 64/32 D6	4.25KB **1.7419** 4K 32/16 D7	394.31KB **0.2705** 512K 256/128 D6
4.51KB **1.0409** 4K 16/8 D6	220.62KB **0.2365** 512K 128/16 D3	4.66KB **1.5725** 4K 32/32 D8	396.31KB **0.2608** 512K 256/64 D6
4.69KB **1.0298** 4K 32/16 D8	234.25KB **0.2326** 256K 32/16 D6	5.32KB **1.5336** 4K 16/16 D8	404.62KB **0.2535** 512K 128/64 D6
4.86KB **1.0261** 8K 64/16 D4	245.12KB **0.2290** 256K 64/32 D7	5.74KB **1.5264** 8K 64/32 D5	460.31KB **0.2527** 512K 256/64 D7
5.01KB **1.0061** 4K 16/8 D7	266.25KB **0.2257** 256K 32/16 D7	6.48KB **1.4519** 8K 32/16 D5	468.62KB **0.2409** 512K 128/64 D7
5.32KB **0.9201** 4K 16/16 D8	272.31KB **0.2201** 512K 256/32 D4	6.74KB **1.3477** 8K 64/32 D6	529.62KB **0.2319** 512K 128/128 D8
5.86KB **0.9194** 8K 64/16 D5	280.62KB **0.2175** 512K 128/32 D4	7.48KB **1.3157** 8K 32/16 D6	547.25KB **0.2223** 512K 64/64 D8
6.48KB **0.8711** 8K 32/16 D5	292.25KB **0.2140** 256K 32/32 D8	7.74KB **1.2835** 8K 64/32 D7	
6.86KB **0.8590** 8K 64/16 D6	336.31KB **0.2079** 512K 256/32 D5	8.48KB **1.2606** 8K 32/16 D7	
7.48KB **0.7894** 8K 32/16 D6	344.62KB **0.2030** 512K 128/32 D5	9.29KB **1.1043** 8K 32/32 D8	
8.48KB **0.7564** 8K 32/16 D7	361.25KB **0.2009** 512K 64/32 D5	13.45KB **1.0086** 16K 64/32 D6	
9.35KB **0.7460** 8K 32/16 D8	400.31KB **0.2008** 512K 256/32 D6	15.45KB **0.9792** 16K 64/32 D7	
10.58KB **0.6881** 8K 16/16 D8	408.62KB **0.1936** 512K 128/32 D6	17.32KB **0.9714** 16K 64/32 D8	
12.89KB **0.6818** 16K 32/16 D5	425.25KB **0.1894** 512K 64/32 D6	17.66KB **0.9564** 32K 128/32 D4	
13.70KB **0.6727** 16K 64/16 D6	472.62KB **0.1884** 512K 128/32 D7	18.52KB **0.8947** 16K 32/32 D8	
14.89KB **0.6404** 16K 32/16 D6	489.25KB **0.1817** 512K 64/32 D7	21.66KB **0.8686** 32K 128/32 D5	
16.89KB **0.6276** 16K 32/16 D7	549.25KB **0.1781** 512K 64/32 D8	22.83KB **0.8285** 32K 64/32 D5	
18.16KB **0.6107** 32K 128/16 D4	582.50KB **0.1702** 512K 32/32 D8	25.88KB **0.8201** 64K 256/32 D3	
19.33KB **0.5898** 32K 64/16 D4		26.83KB **0.7891** 32K 64/32 D6	
21.66KB **0.5824** 32K 32/16 D4		30.83KB **0.7561** 32K 64/32 D7	
23.33KB **0.5549** 32K 64/16 D5		34.13KB **0.7061** 64K 256/32 D4	
25.66KB **0.5417** 32K 32/16 D5		35.27KB **0.6790** 64K 128/32 D4	
27.33KB **0.5364** 32K 64/16 D6		36.91KB **0.6665** 32K 32/32 D8	
27.77KB **0.5272** 64K 128/16 D3		37.53KB **0.6653** 64K 64/32 D4	
29.66KB **0.5189** 32K 32/16 D6		42.13KB **0.6402** 64K 256/32 D5	
30.03KB **0.5186** 64K 64/16 D3		43.27KB **0.6050** 64K 128/32 D5	
31.33KB **0.5161** 32K 64/16 D7		45.53KB **0.5818** 64K 64/32 D5	
33.66KB **0.4936** 32K 32/16 D7		51.27KB **0.5771** 64K 128/32 D6	
34.83KB **0.4887** 32K 64/16 D8		51.70KB **0.5579** 128K 256/32 D3	
35.27KB **0.4850** 64K 128/32 D4		53.53KB **0.5446** 64K 64/32 D6	
36.27KB **0.4453** 64K 128/16 D4		61.53KB **0.5266** 64K 64/32 D7	
38.53KB **0.4324** 64K 64/16 D4		68.20KB **0.5024** 128K 256/32 D4	
41.81KB **0.4252** 32K 16/16 D8		70.41KB **0.4856** 128K 128/32 D4	
43.06KB **0.4222** 64K 32/16 D4		73.56KB **0.4684** 64K 32/32 D8	
44.27KB **0.3982** 64K 128/16 D5		86.41KB **0.4521** 128K 128/32 D5	
46.53KB **0.3817** 64K 64/16 D5		90.81KB **0.4394** 128K 64/32 D5	
51.06KB **0.3658** 64K 32/16 D5		102.41KB **0.4345** 128K 128/32 D6	
54.53KB **0.3641** 64K 64/16 D6		103.28KB **0.4246** 256K 256/32 D3	
55.41KB **0.3540** 128K 128/16 D3		106.81KB **0.4157** 128K 64/32 D6	
59.06KB **0.3434** 64K 32/16 D6		122.81KB **0.4021** 128K 64/32 D7	
67.06KB **0.3323** 64K 32/16 D7		134.28KB **0.3972** 256K 256/64 D4	
72.41KB **0.3239** 128K 128/16 D4		136.28KB **0.3887** 256K 256/32 D4	
76.81KB **0.3168** 128K 64/16 D4		137.31KB **0.3843** 128K 64/64 D8	
83.13KB **0.3083** 64K 16/16 D8		140.56KB **0.3810** 256K 128/32 D4	
92.81KB **0.2994** 128K 64/16 D5		146.62KB **0.3725** 128K 32/32 D8	
101.63KB **0.2935** 128K 32/16 D5		166.28KB **0.3656** 256K 256/64 D5	
108.81KB **0.2895** 128K 64/16 D6		170.56KB **0.3573** 256K 128/64 D5	
110.56KB **0.2858** 256K 128/16 D3		172.56KB **0.3563** 256K 128/32 D5	
117.62KB **0.2805** 128K 32/16 D6		181.12KB **0.3504** 256K 64/32 D5	
133.62KB **0.2722** 128K 32/16 D7		198.28KB **0.3483** 256K 256/64 D6	
140.56KB **0.2721** 256K 128/32 D4		199.16KB **0.3403** 512K 512/64 D3	
144.56KB **0.2654** 256K 128/16 D4		202.56KB **0.3341** 256K 128/64 D6	
153.12KB **0.2617** 256K 64/16 D4		213.12KB **0.3317** 256K 64/32 D6	
165.25KB **0.2567** 128K 16/16 D8		234.56KB **0.3197** 256K 128/64 D7	
172.56KB **0.2545** 256K 128/32 D5		264.16KB **0.3051** 512K 512/64 D4	
176.56KB **0.2528** 256K 128/16 D5		268.31KB **0.2999** 512K 256/64 D4	
181.12KB **0.2503** 256K 64/32 D5		274.12KB **0.2942** 256K 64/64 D8	
185.12KB **0.2470** 256K 64/16 D5		328.16KB **0.2851** 512K 512/64 D5	
202.25KB **0.2438** 256K 32/16 D5		332.31KB **0.2758** 512K 256/64 D5	
206.31KB **0.2406** 512K 256/32 D3		340.62KB **0.2722** 512K 128/64 D5	

Table 11: The best sector pool data cache configurations for a given bit budget, 5 and 15 cycle overhead. Each entry contains the total bits to implement the cache (in Kbytes), the additional delay (in bold), the nominal cache size, the sector/subsector size, and the depth (1-8).