Performance Implications of Tolerating Cache Faults†

Andreas Farid Pour
Mark D. Hill

Computer Sciences Department
University of Wisconsin — Madison
1210 West Dayton Street
Madison, Wisconsin 53706

ABSTRACT

Microprocessors are increasingly incorporating one or more on-chip caches. These caches are occupying a greater share of chip area, and thus may be the locus of manufacturing defects. Some of these defects will cause faults in cache tag or data memory. These faults can be tolerated by disabling the cache blocks that contain them. This approach lets chips with defects be used without requiring on-chip caches to have redundant row or columns or to use error correcting codes. Disabling blocks, however, typically increases a cache's miss ratio.

This paper investigates how much cache miss ratios increase when blocks are disabled. It shows how the mean miss ratio increase can be characterized as a function of the miss ratios of related caches, develops an efficient approach for calculating the exact distribution of miss ratio increases from all fault patterns, and applies this approach to the ATUM traces [1]. Results reveal that the mean relative miss ratio increase from a few faults decreases with increasing cache size, and is negligible (< 2% per defect) unless a set is completely disabled by faults. The maximum relative increase is also acceptable (< 5% per fault) if no set is entirely disabled.

Index Terms: computer architecture, cache performance, trace-driven simulation, fault tolerance, on-chip caches, microprocessors.

---

† The preliminary version of this work appeared as University of Wisconsin Computer Sciences Technical Report #991, January 1991. This work is supported in part by the National Science Foundation (MIPS-8957278 and CCR-8902536), A.T.& T. Bell Laboratories, Cray Research Foundation and Digital Equipment Corporation.

1. Introduction

Commercial and academic microprocessor architectures are increasingly incorporating caches on the processor chip itself to avoid off-chip latencies [3, 6, 8, 12]. These *on-chip caches* are currently small, but the trend is toward larger sizes to hide relatively slower off-chip memory speeds; thus, these chips devote an increasing portion of their area to the memory (tags and blocks) of the cache. As cache chip area becomes large, so will the fraction of manufacturing defects that land in the cache.

A manufacturing defect causes a *fault* in a cache if it impairs the correct operation of the cache. We will study those faults that make a bit in the cache unable to retain the value written to it, but that do not otherwise perturb the operation of the cache (e.g., do not cause an electrical short circuit). A fault causes an *error* if it causes the system to enter a logical state other than the one intended. We can prevent faults in an on-chip cache from causing errors by (1) discarding chips with such faults, (2) using redundant memory, or (3) disabling cache blocks that contain faults. The advantage of discarding chips, method (1), is that it works for any defect. Its disadvantage, however, is that by reducing yield it increases chip cost.

Redundant memory, method (2), can be employed to tolerate faults in at least three ways: (a) add extra memory words (rows) that are selected instead of faulty ones, (b) add extra bits per word (columns) that are selected instead of faulty ones, or (c) add extra bits per word to store an error correcting code. The advantage of these approaches is that they work for any memory. Two disadvantages, however, are (a) the cost or opportunity cost of the extra memory, and (b) a possible memory access time increase caused by implementing them.

Disabling cache blocks that contain faults, method (3), is applicable only to caches. Since a cache merely keeps a copy of data from main memory, all memory in a cache is redundant. Thus, instead of building redundancy on top of the redundant memory in a cache, this method just causes the cache to avoid using the memory that contains faulty bits.

Caches are buffers used to hold data from recently-used parts of main memory [14]. Data is usually transferred from main memory in aligned *blocks* (also called *lines*). The number of bytes in a block is the *block size*. A block is stored in a cache with memory that holds its contents, its tag (part of its main memory address) and some state bits, including a *valid bit* that indicates whether a block is present. The blocks in a cache are usually divided into *sets*. Every block maps to one set, so that only blocks in that set must be searched on a reference. A cache with *associativity n* has n blocks in each set. A cache is *direct-mapped* if $n = 1$, *fully-associative* if n equals the number of blocks in the cache, or *n-way set-associative* otherwise. *Cache size* is the number of blocks in a cache multiplied by the block size. Finally, the *number of sets* is the cache size divided by the product of the block size and associativity. Caches are usually characterized by their block size, associativity and cache size.

One approach to implement the disabling of cache blocks, proposed by Patterson, et al. [11], is to use a second valid bit. Each cache block normally contains a valid bit that is set when a block is brought in and reset when the block is invalidated. A cache hit occurs when the address tag matches the address referenced by the processor and the valid bit is set. We can implement method (3) by adding a second valid bit to each block which is set or reset with a memory test mechanism,[1] but left unchanged during normal cache operation. With this enhancement, a cache hit now requires a tag match and both valid bits to be set. On a

[1] The memory test mechanism can be invoked when a chip is tested, or it can be built in and run whenever the system is reset.

cache miss, the incoming data should be loaded into a block whose second valid bit is set. If all blocks in a set are faulty, then one can either (a) discard the chip, (b) bypass the cache and send the requested data directly to its destination (CPU or memory), or (c) save the data in a special buffer, such as a victim cache [7]. Clearly, applying option (a) to a direct-mapped cache is equivalent discarding all faulty chips.

Disabling cache blocks offers two advantages over using redundant memory, but also suffers two disadvantages. The first advantage is that, unlike with redundant memory, implementing a second valid bit does not increase cache access time on a hit. The second advantage is that the method allows all non-faulty blocks to be used, whereas redundant memory is used only when some memory is faulty.

The disadvantages of this method are that it can increase both the mean and variability of the cache miss ratio, whereas using redundant memory leaves the miss ratio unchanged. Consider disabling one block in set i of a four-way set-associative, 64K-byte cache with 32-byte blocks using the LRU (least-recently-used) replacement algorithm. References to set i will see a set with associativity three, while references to the other 511 sets will behave normally. References that would have hit in the fourth-recently-used block of set i in an intact cache will now miss; all other references perform the same.

Nevertheless, disabling cache blocks can lead to a better average access time than using redundant memory. The average access time with redundant memory, method (2), is:

$$t_{method(2)} = (t_{cache} + t_{\Delta}) + m \times t_{memory},$$

where t_{cache} is the access time to a cache without redundant memory, t_{Δ} is the additional time needed to implement redundant memory, m is the miss ratio of the cache, and t_{memory} is the access time for main memory. For disabling cache blocks, method (3), the average access time is:

$$t_{method(3)} = t_{cache} + m \times (1 + \delta) \times t_{memory},$$

where t_{cache}, m, and t_{memory} are as above and δ is the relative increase in the cache miss ratio due to faults. Rearranging the terms, we find that disabling blocks is inferior to redundant memory only when:

$$t_{\Delta} < \delta\, m\, t_{memory}.$$

Since our results show that values for δ (from one or two faults) are often two to four percent (see Section 4) and $t_{cache} > m\, t_{memory}$ for most caches, t_{Δ} must be less than two to four percent of t_{cache} for method (2) to exhibit better performance than method (3). Furthermore, method (3) is clearly faster for chips with no faults (where $\delta = 0$), and it is likely that its performance can be improved significantly with victim caches [7].

To the best of our knowledge, the only paper to do a detailed investigation of the effect on miss ratios of disabling cache blocks is by Sohi [15]. Sohi investigated the degradation in cache performance by randomly injecting faults into the cache and then running a trace-driven simulation. For each cache, Sohi reports the average miss ratio of several simulations with different fault patterns. He presents results for the number of faulty blocks ranging from 0% to 50% of the blocks in caches of three sizes (256, 1K, and 8K bytes), three associativities (direct-mapped, two-way set-associative, and fully associative) and three block sizes (8, 16, and 32 bytes).

This paper extends Sohi's work in two key ways. First, we show that the distribution of miss ratio increases can be calculated from LRU distance probabilities for each set, while Sohi's paper did not consider this issue. One implication of our equations is to confirm the intuition that the mean of the miss ratio for a cache with s sets and a single fault is equivalent to $s - 1$ sets seeing an unperturbed cache and a single set seeing an associativity decreased by one.[2] For example, let m_0 be the miss ratio for a 64K-byte cache with associativity four, a block size of 32 bytes and no faults and m_1 be the miss ratio for a 48K-byte cache with associativity three, a block size of 32 bytes and no faults. We show the mean miss ratio for a 64K-byte cache with associativity four, a block size of 32 bytes and one fault is $(511/512) \times m_0 + (1/512) \times m_1$.

Second, we show how *all-associativity simulation* [5] can be extended to collect information for finding the effect of all possible patterns of faults on caches with many associativities and sizes (but one block size) *in one pass through an address trace*. Sohi, on the other hand, performed a simulation for each fault pattern in each cache. For this reason, Sohi estimates the mean miss ratio increases from a small fraction of the possible fault patterns[3] and limits the variety of caches examined. Our approach, on the other hand, allows us to calculate the exact mean, maximum and standard deviation of miss ratio increases for several faults and a wider range of caches. We concentrate on caches with few faults, because we believe that chips with many faults in the cache will usually have faults in other critical resources, and thus will be discarded anyway.

Results with the ATUM traces [1] suggest that the *mean* relative miss ratio increase from a few faults decreases with increasing cache size, and is usually small (< 5% per fault). Furthermore, if no set is completely disabled, mean degradation for large caches is negligible. Consequently, it is likely that the effective access time of a cache with some blocks disabled will be less than that of a cache using redundant memory.

The *maximum* relative miss ratio increase for a single cache fault − or for two cache faults in distinct sets − is acceptable if the associativity of the cache is two or greater and the block size is 8 or 16 bytes. Larger block sizes suffer slightly larger miss ratio degradations when blocks are disabled. With a direct-mapped cache, however, there is a probability (albeit small with a large number of sets) that the executing program heavily references the faulty block(s), thereby severely degrading the cache's performance. Nevertheless, we expect that the overall impact of this worst-case behavior will not be important on machines used to run many different programs.

The rest of the paper proceeds as follows. Section 2 develops equations for the impact of cache faults on the miss ratios. Section 3 discusses how data for many fault patterns in many caches can be gathered with a single pass through an address trace. Section 4 presents the results of the investigation, and Section 5 concludes our discussion.

2. The Impact of Faulty Blocks

We now turn to the impact of disabling faulty cache blocks on the cache miss ratio. We show how the impact can be expressed from per-set simulation data for any pattern of faults and then derive equations for some simple cases. These equations show what data must be gathered in trace-driven simulations so that miss ratios for any fault pattern can be calculated.

Note that the following derivation makes no assumptions about the distribution of the reference stream. Assume a cache has s sets labeled 0 through $s - 1$, and is referenced by a dynamic reference stream of R references. Assuming all blocks within a set are ordered according to some *stack replacement algorithm* [10] (such as LRU), define $D_i(j)$ to be the number of references to the j-th block in the i-th set

---

[2] Assume that the miss ratio of a cache with associativity zero is one.

[3] For $m \ll s$, the number of ways to place m faults in s sets is $O(s^m)$

and $D(j)$ to be the number of references to the j-th block in any set. Then:

$$D(j) = \sum_{i=0}^{s-1} D_i(j), \quad j > 0.$$

Let $M(n)$ be the number of misses in an n-way set-associative cache with s sets accessed by R references. Since a miss occurs when a reference is not to one of the first n blocks of a set:

$$M(n) = R - \sum_{j=1}^{n} D(j), \quad n > 0 \quad \text{and}$$

$$M(0) = R.$$

Further, define $\mathbf{F}$ to be an s-element fault vector $(f_0, f_1, \cdots, f_{s-1})$, where f_i is the number of faulty blocks in set i. The additional misses these faults cause in an n-way set-associative cache are:

$$\Delta(n, \mathbf{F}) = \sum_{i=0}^{s-1} \sum_{j=0}^{f_i - 1} D_i(n-j), \quad n > 0.$$

The number of misses for the faulty cache, $M(n, \mathbf{F})$, its miss ratio, $m(n, \mathbf{F})$, and the relative miss ratio increase with respect to a similar fault-free cache, $\delta(n, \mathbf{F})$, are:

$$M(n, \mathbf{F}) = M(n) + \Delta(n, \mathbf{F}), \quad n > 0,$$

$$m(n, \mathbf{F}) = \frac{M(n, \mathbf{F})}{R}, \quad n > 0, \quad \text{and}$$

$$\delta(n, \mathbf{F}) = \frac{m(n, \mathbf{F}) - m(n)}{m(n)} = \frac{\Delta(n, \mathbf{F})}{M(n)}, \quad n > 0.$$

Thus, the performance of a cache with s sets and associativity n with *any* pattern of faults can be determined from the values of $D_i(j)$ for $i = 0$ to $s-1$ and $j = 1$ to n. Section 3 will show how to perform trace-driven simulation to gather $D_i(j)$'s prior to selecting a fault vector. This allows us to apply many fault vectors to the same simulation results.

Next we apply the above equations to the important special cases of one and two faults. Table 1 repeats frequently-used notation.

2.1. Single Faults

Single fault vectors $\mathbf{F_1}$ are a special case of a fault vector $\mathbf{F}$ where set i has one fault and all other sets have no faults:

$$f_i = 1, \quad 0 \le i \le s-1, \quad \text{and}$$

$$f_k = 0, \quad k \ne i.$$

The effect of a fault in set i of an n-way set-associative cache is to cause a cache miss on references to block n in set i, which would not have missed without the fault. The other $n-1$ blocks in the set with the fault are unaffected, as are the remaining $s-1$ sets in the cache. This implies that the additional misses induced by the fault vector $\mathbf{F_1}$ are:

$$\Delta(n, \mathbf{F_1}) = D_i(n).$$

Term	Definition
s	the number of sets; implicitly-used in definitions below.
n	the associativity.
R	the number of references (trace length).
$D_i(j)$	the number of references to the j-th block in the i-th set.
$D(j)$	the number of references to the j-th block in any set.
$M(n)$	the number of misses in an n-way set-associative cache.
$m(n)$	the miss ratio of an n-way set-associative cache.
$\mathbf{F}$	an s-element fault vector $(f_0, f_1, \cdots, f_{s-1})$, where f_i is the number of faulty blocks in set i.
$\Delta(n, \mathbf{F})$	the number of additional misses in an n-way set-associative cache with faults according to fault vector $\mathbf{F}$ over a similar fault-free cache.
$m(n, \mathbf{F})$	the miss ratio of an n-way set-associative cache with faults according to fault vector $\mathbf{F}$.
$\delta(n, \mathbf{F})$	the relative miss ratio increase of an n-way set-associative cache with faults according to fault vector $\mathbf{F}$ over a similar fault-free cache; for brevity, referred to as the *relative increase*.

Table 1 Frequently-used notation.

Since the number of sets in a cache, s, can be large, it is worthwhile to distill the distribution of $\Delta(n, \mathbf{F_1})$ across all sets i. Assume that the fault is equally likely to be present in any set i. Then the mean (E), maximum (MAX), and standard deviation (STD) of $\Delta(n, \mathbf{F_1})$ are:

$$E\left[\Delta(n,\ \mathbf{F_1})\right] = \frac{1}{s}\sum_{i=0}^{s-1} D_i(n) = \frac{D(n)}{s},$$

$$MAX\left[\Delta(n,\ \mathbf{F_1})\right] = \underset{i}{MAX}\left[D_i(n)\right], \quad \text{and}$$

$$STD\left[\Delta(n,\ \mathbf{F_1})\right] = \left[\sum_{i=0}^{s-1}\left(\frac{D_i(n)}{s}\right)^2 - \left(\sum_{i=0}^{s-1}\frac{D_i(n)}{s}\right)^2\right]^{\frac{1}{2}}. \tag{1}$$

Several substitutions may be made to aid in understanding the mean additional misses, mean miss ratio, and mean relative miss ratio increase:

$$E\left[\Delta(n,\ \mathbf{F_1})\right] = \frac{D(n)}{s} = \frac{M(n-1) - M(n)}{s},$$

$$E\left[m(n,\ \mathbf{F_1})\right] = m(n) + \frac{m(n-1) - m(n)}{s} = \frac{s-1}{s}\, m(n) + \frac{1}{s}\, m(n-1), \quad \text{and} \tag{2}$$

$$E\left[\delta(n,\ \mathbf{F_1})\right] = \frac{1}{s}\,\frac{D(n)}{M(n)} = \frac{1}{s}\,\frac{m(n-1) - m(n)}{m(n)} = \frac{1}{s}\left(\frac{m(n-1)}{m(n)} - 1\right). \tag{3}$$

Equation (2) confirms the intuition that the mean of the miss ratio with a single fault is equivalent to $s-1$ sets seeing an unperturbed cache and a single set seeing an associativity decreased by one. Equation (3) suggests that the mean relative miss ratio increase will be small when:

i) s is large, or

ii) $[m(n-1) - m(n)]$ is small.

2.2. Double Faults

The case of two faults may be subdivided into three cases, where the two faults occur (a) in the different sets (with fault vector denoted by $\mathbf{F_{2,diff}}$), (b) in the same set ($\mathbf{F_{2,same}}$), or (c) anywhere ($\mathbf{F_2}$).

2.2.1. Double Faults in Different Sets

Fault vector $\mathbf{F_{2,diff}}$ denotes the case of two faults in different sets i and j:

$$f_i = f_j = 1, \quad 0 \le i,\ j \le s-1, \quad i \ne j, \quad \text{and}$$

$$f_k = 0, \quad k \ne i,\ k \ne j.$$

Like the single-fault case, the number of misses is an n-way set-associative cache increases with each reference to the n-th block in sets i and j:

$$\Delta(n,\ \mathbf{F_{2,diff}}) = D_i(n) + D_j(n).$$

Assume that the first fault is equally likely to land in any set i and the second fault is equally likely to land in any other set. Then,[4]

$$E\left[\Delta(n,\ \mathbf{F_{2,diff}})\right] = \frac{1}{s(s-1)}\sum_{i=0}^{s-1}\ \sum_{j=0,\, j\ne i}^{s-1}\left(D_i(n) + D_j(n)\right)$$

---

[4] Define $\sum_{j=0,\, j\ne i}^{n} x_j$ as $\left(\sum_{j=0}^{n} x_j\right) - x_i$.

$$= \frac{D(n)}{s} + \frac{1}{s(s-1)} \sum_{j=0}^{s-1} (s-1)\, D_j(n)$$

$$= \frac{2\,D(n)}{s} = \frac{2\left(M(n-1) - M(n)\right)}{s}$$

$$= 2\,E\left[\Delta(n,\ \mathbf{F_1})\right].$$

Similarly,

$$E\left[m(n,\ \mathbf{F_{2,diff}})\right] = \frac{s-2}{s}\, m(n) + \frac{2}{s}\, m(n-1), \quad \text{and}$$

$$E\left[\delta(n,\ \mathbf{F_{2,diff}})\right] = 2\,E\left[\delta(n,\ \mathbf{F_1})\right]. \tag{4}$$

In general, the mean miss ratio and mean relative miss ratio increase for a fault vector with g faults, no two of which map to the same set ($0 \le g \le s$), are:

$$E\left[m(n,\ \mathbf{F_{g,diff}})\right] = m(n) + \frac{\left(m(n-1) - m(n)\right) g}{s} = \frac{s-g}{s}\, m(n) + \frac{g}{s}\, m(n-1) \ \text{ and} \tag{5}$$

$$E\left[\delta(n,\ \mathbf{F_{g,diff}})\right] = g\,E\left[\delta(n,\ \mathbf{F_1})\right].$$

2.2.2. Double Faults in the Same Set

Fault vector $\mathbf{F_{2,same}}$ denotes the case of two faults in the same set i:

$$f_i = 2, \quad 0 \le i \le s-1, \ \text{ and}$$

$$f_k = 0, \quad k \ne i.$$

The additional misses induced by the fault vector $\mathbf{F_{2,same}}$ in an n-way set-associative cache is equal to the number of references to the n-th and $(n-1)$-th blocks in set i:

$$\Delta(n,\ \mathbf{F_{2,same}}) = D_i(n) + D_i(n-1), \quad n \ge 2.$$

If each set i is equally likely to have the faults, then:

$$E\left[\Delta(n,\ \mathbf{F_{2,same}})\right] = \frac{1}{s} \sum_{i=0}^{s-1} \left(D_i(n) + D_i(n-1)\right)$$

$$= \frac{D(n) + D(n-1)}{s} = \frac{M(n-2) - M(n)}{s}.$$

$$E\left[m(n,\ \mathbf{F_{2,same}})\right] = m(n) + \frac{m(n-2) - m(n)}{s} = \frac{s-1}{s}\, m(n) + \frac{1}{s}\, m(n-2), \quad \text{and}$$

$$E\left[\delta(n,\ \mathbf{F_{2,same}})\right] = \frac{1}{s}\, \frac{m(n-2) - m(n)}{m(n)} = \frac{1}{s} \left(\frac{m(n-2)}{m(n)} - 1 \right).$$

2.2.3. Double Faults Anywhere

Fault vector $\mathbf{F_2}$ denotes the case where two faults may or may not be in the same set:

$$f_i + f_j = 2, \quad 0 \le i,\ j \le s-1, \quad i \ne j, \quad \text{and}$$

$$f_k = 0, \quad k \ne i, \ k \ne j.$$

Assuming $n \ge 2$, the number of additional misses induced by the fault vector $\mathbf{F_2}$ is:

$$\Delta(n,\ \mathbf{F_2}) = \begin{cases} D_i(n) + D_j(n), & i \ne j \\ D_i(n) + D_i(n-1), & i = j \end{cases}.$$

If the first fault is equally likely to land in any set and the second fault is independent of the first and equally likely to land in any set, then the probability that both faults land in the same set is $1/s$. Therefore,

$$E\left[\Delta(n,\ \mathbf{F_2})\right] = \frac{s-1}{s} E\left[\Delta(n,\ \mathbf{F_{2,diff}})\right] + \frac{1}{s} E\left[\delta(n,\ \mathbf{F_{2,same}})\right]$$

$$= E\left[\Delta(n,\ \mathbf{F_{2,diff}})\right] + \frac{1}{s}\left(E\left[\Delta(n,\ \mathbf{F_{2,same}})\right] - E\left[\Delta(n,\ \mathbf{F_{2,diff}})\right]\right). \tag{6}$$

Similar equations can be derived for m and δ. For large s, the mean behavior with two faults anywhere (fault vector $\mathbf{F_2}$) will be indistinguishable from the mean with two faults in different sets ($\mathbf{F_{2,diff}}$), because the final term has $\frac{1}{s}$ as a factor.

2.2.4. Multiple Faults Anywhere

Multiple fault vectors are the general case of a fault vector $\mathbf{F}$ where the number of faults, m, are in the range $2 \le m \le sn$; thus, all the blocks in the set may be faulty. These cases do not allow for concise descriptions and require a probabilistic approach, since exhaustive simulation has time complexity of $O(s^m)$.[5]

To do these calculations, it is necessary to determine the probabilities of a certain number of faults occurring in a particular set given the total number of faults. Then, the expected miss ratio can be described by:

$$E\left[m(n,\ \mathbf{F})\right] = \frac{P[\,Y=0\,]\,m(n) + P[\,Y=1\,]\,m(n-1) + P[\,Y=2\,]\,m(n-2) + \cdots + P[\,Y=g\,]\,m(n-g)}{P[\,Y=0\,] + P[\,Y=1\,] + P[\,Y=2\,] + \cdots + P[\,Y=g\,]}, \tag{7}$$

where $P[\,Y=i\,]$ is the probability that a particular set has i faults when all m faults are uniformly randomly distributed among s sets with associativity n and at most g faults are allowed in a particular set ($0 \le g \le n$). To calculate the probabilities, we first iteratively determined all possible distributions of f faults over s sets (a partitioning problem), and then used multinomials to calculate how times ways a particular distribution occurs.

[5] This approximation holds only for the range $0 < m < \frac{sn}{2}$.

3. Methods

In the last section, we showed how the $D_i(j)$'s, the number of references to the j-th block in the i-th of s sets, can be used to calculate the miss ratio for a single cache with any fault pattern. In this section, we show how to find the $D_i(j)$'s for many caches with a single pass through an address trace. We also describe the address traces that we use.

All-associativity simulation [5] is an algorithm that calculates the miss ratios for caches of many sizes and associativities with a single pass through an address trace, provided that all caches have the same block size, use the least-recently-used replacement algorithm and do no prefetching. In an efficient manner, the algorithm examines a trace reference to determine that the reference is to the j-th most-recently-used block of the i-th of s sets for caches with many values of s. This information is sufficient to calculate the $D_i(j)$'s. All-associativity simulation, however, collapses the information and only calculates the $D(j)$'s, since it does not need to retain set numbers to determine the miss ratios of fault-free caches.

We extended all-associativity simulation to record $D_i(j)$'s, instead of $D(j)$'s, simply by expanding its counters by a factor of s to record the set referenced. Our extension has a negligible effect on simulation run-time. It causes only a modest increase in simulation storage, because the storage needed for the expanded counters is still smaller than the storage needed for the address tags of the caches being simulated.

Once we know each $D_i(j)$, we can calculate miss ratios of caches with faults without additional trace-driven simulation. The faults are distributed in every possible way and the relevant statistics extracted by the equations presented in Section 2. Since a direct-mapped 32K-byte cache with a block size of 8 bytes contains 2^{12} sets, exhaustively calculating all miss ratios for three faults in the cache involves $(2^{12})^3 = 2^{36}$ calculations. Therefore, we use a probabilistic model for more than two faults. Nevertheless, we base our results on many more cases than Sohi, since we can calculate a new miss ratio by summing appropriate $D_i(j)$'s rather than by performing a complete trace-driven simulation. We validated our results by comparing them with results of the exhaustive simulations.

We use the ATUM traces, because they were the only available traces that included operating systems references and multiprogramming effects [1]. Table 2 shows the number of instruction fetches, data reads, and data writes for each of the traces used, as well as a brief description of their origins. Due to the large number of traces, we give results only for a combined trace, denoted by *all*. We constructed the combined trace by alternating the individual traces and cache flushes. For results from two representative individual traces, see Pour and Hill [13]. We only simulate caches smaller than 64K bytes, because the individual traces are not long enough (typically, 400,000 references) to properly exercise larger caches.

4. Results

This section presents simulation results for one, two and many faults. Most of the analysis uses the relative miss ratio increase caused by introducing faults ($\delta(n, \mathbf{F})$). For brevity, we will usually refer to this metric as the *relative increase*. Unless otherwise indicated, the block size will be 16 bytes.

4.1. Single Faults

Figure 1 shows the mean and maximum miss ratios of the ''all'' simulation with one fault ($\mathbf{F_1}$) for various associativities (number of blocks per set) and cache sizes. The miss ratios ostensibly follow the behavior of fault-free cache miss ratios [2, 4, 5]: for all associativities, the miss ratios decrease with

Name	Data Reads	Data Writes	Instruction Fetches	Description
dec0.000	106459	72500	183023	DECSIM behavioral simulation of some cache hardware
dec0.003	103906	73001	176533	Same as previous
fora.000	108979	79156	199799	FORTRAN compiler compiling airco.for
forf.000	108048	85845	207284	Two FORTRAN compilations: 4x1x5.for and linpack.for
forf.001	110027	73093	203595	Same as previous
forf.002	105131	91233	217509	Same as previous
forf.003	107969	69328	190915	Same as previous
fsxzz.000	78265	37840	123229	ULTRIX file system exerciser, 20 tasks
ivex.000	97335	41123	203510	Interconnect verify
macr.000	96904	57222	188702	Macro assembler assembling linpack2.mar
memxx.000	126660	99139	219050	ULTRIX memory exerciser, 10 tasks
mul2.001	112102	71845	201866	Multiprocessing 2 jobs: ALLC (Microcode address allocator, bit string inner loop) and SPIC (Spice simulating output buffer)
mul2.002	106329	74357	201941	Same as previous
mul2.003	96662	64884	205295	Same as previous
mul8.001	126139	74749	207538	Multiprocessing 8 jobs: Unknown jobs
mul8.002	105813	74881	208900	Multiprocessing 8 jobs: Unknown jobs
mul8.003	141126	88851	199455	Multiprocessing 8 jobs: Unknown jobs
savec.000	130288	85373	215867	ULTRIX C compiler
ue02.000	98385	59452	199973	UETP (User Environment Test Program, a VMS diagnostic), 2 tasks
ue10.000	98494	61476	212150	UETP, 10 tasks
ue20.000	100670	62188	201582	UETP, 20 tasks

Table 2 Traces used in the simulations.

increasing cache size and for a given cache size, the miss ratios decrease with increasing associativity. This comes as no surprise, since Equation (2) predicts the mean miss ratio to be equivalent to a cache with $s - 1$ unperturbed sets and one set with an effective associativity of $n - 1$. Thus in the worst case ($n = 1$) the mean miss ratio will degrade by $\frac{1}{s}$.

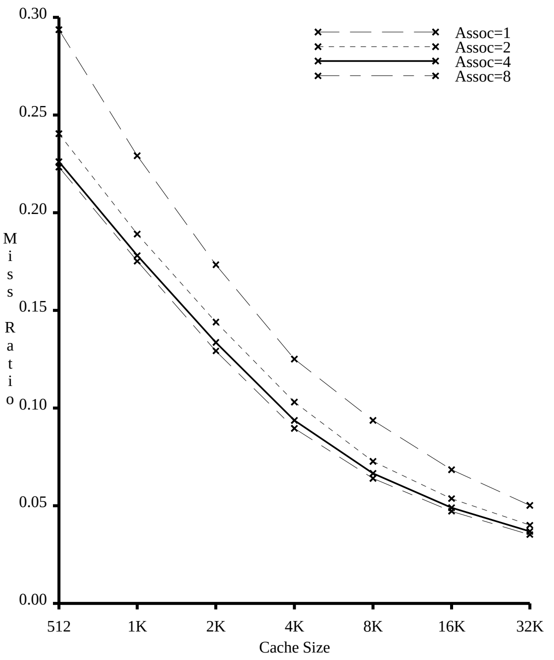

Figure 1a Mean miss ratio of the ''all'' simulation with one fault for various associativities

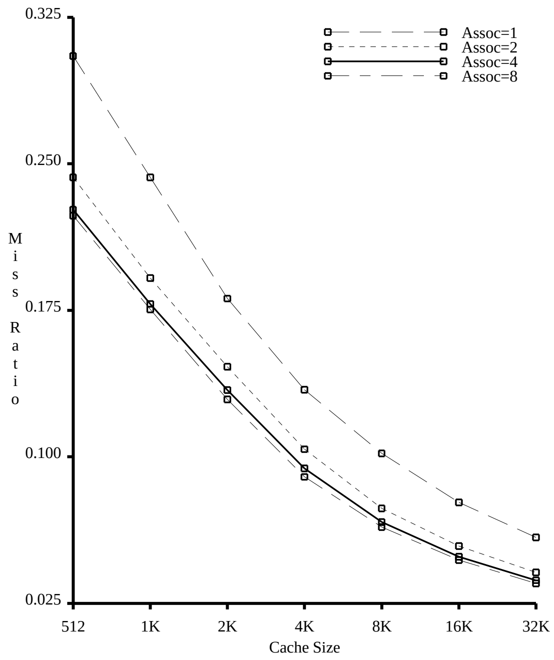

Figure 1b Maximum miss ratio of the ''all'' simulation with one fault for various associativities

More rapid insight into the effect of introducing a fault can be gained by studying the relative (miss ratio) increase. Figures 2a and 2b show the mean and maximum relative increase for the ''all'' simulation. Figure 2a reveals that the mean relative increase gets smaller as the associativity or number of sets gets larger. For direct-mapped caches (associativity one), one fault disables a whole set; thus, on average, $\frac{D(1)}{s}$ additional references will miss. As the associativity increases, the effect of the fault is to reduce the associativity of a particular set by one. Locality of references reduces the impact of this with larger associativities. At the extreme of a fully-associative cache, the cache size is merely reduced by one block, resulting in a negligible impact on the miss ratio.

As the cache size increases with a given associativity, the mean relative increase gets smaller for all associativities simulated. It does so, because increasing the cache size (while holding the block size and associativity constant) increases the number of sets in the cache. From Equation (3) we can see that the mean relative increase is proportional to $\frac{1}{s}$. Furthermore, the fraction $\frac{m(n-1)}{m(n)}$ also tends to decrease with increasing cache size [5].

Figure 2b illustrates that maximum relative increase for one fault:

$$MAX\left[\delta(n,\ \mathbf{F_1})\right] = \frac{1}{M(n)}\underset{i}{MAX}\left[D_i(n)\right].$$

For associativities of two and larger, as cache size gets larger, the maximum relative increase generally gets smaller because $\underset{i}{MAX}\left[D_i(n)\right]$ decreases faster than the total number of misses ($M(n)$) declines. Each

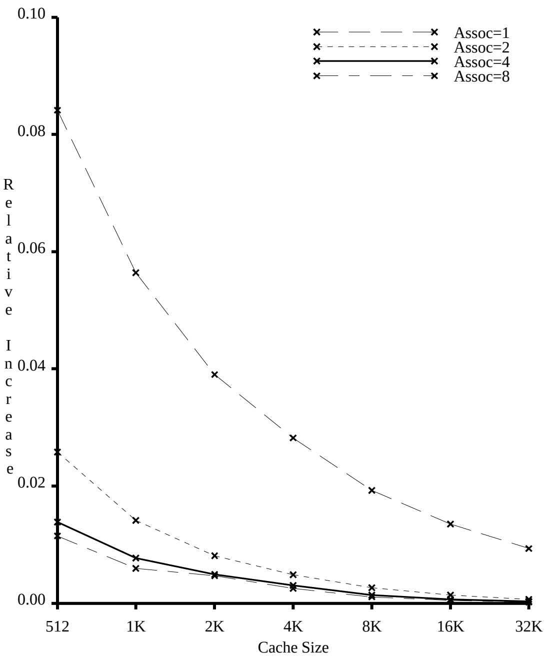

Figure 2a Mean relative miss ratio increase of the ''all'' simulation with one fault for various associativities

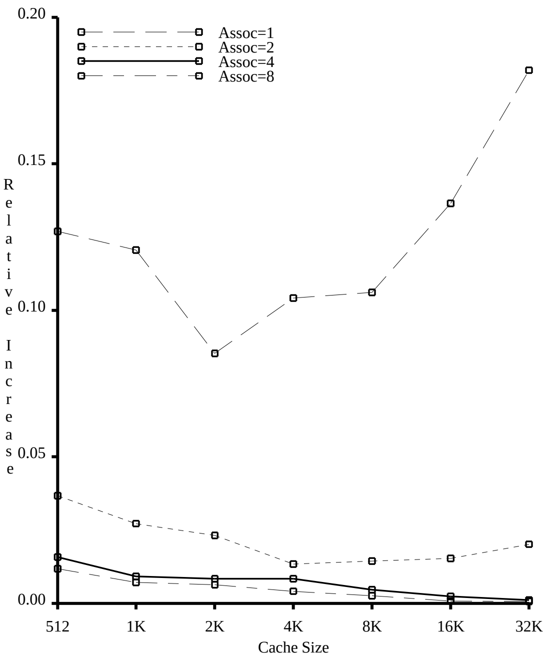

Figure 2b Maximum relative miss ratio increase of the ''all'' simulation with one fault for various associativities

doubling of cache size doubles the number of sets, thereby typically reducing the number of cases where *n* blocks map the same set. Eventually, $\underset{i}{MAX}\left[D_i(n)\right]$ reaches zero, when the cache size is large enough that no set sees *n* blocks.

With direct-mapped caches, however, maximum relative increase gets larger for larger caches. This is because, as cache size increases, $\underset{i}{MAX}\left[D_i(1)\right]$ approaches the number references in the address trace to the most-recently referenced block (instead of approaching zero as it did for set-associative caches), while the total number of misses still declines. Even though the maximum relative increase gets larger with cache size, the maximum miss ratios still decrease (see Figure 1b).

For brevity, we will not present numerical results for standard deviations. However, for caches with associativities greater than one the general trend for the standard deviation of the relative increase is a steady, small decrease as caches get larger. With direct-mapped caches, the trend is toward slight increase as caches get larger. The standard deviation in relative increase also gets smaller with larger associativity.

Figure 3 depicts the effect of changing the block size on the relative increase. All the graphs in Figure 3 include block sizes of 8 bytes, 16 bytes, and 32 bytes. Figure 3a displays the mean relative increase for associativities of one and two, while Figure 3b does so for associativities of four and eight. Qualitatively, the results do not digress from the observations made above; the lines for a block size of 16 are as in Figure 2. The most notable aspect of these graphs come as no surprise: the larger the block size, the greater the relative increase. Disabling a 32-byte block is for all practical purposes equivalent to disabling two adjacent 16-byte blocks. Thus on the average it seems reasonable to expect an approximate doubling in the additional misses caused by disabling a B-byte block with respect to a $B/2$-byte block; some discrepancy is caused by the fact that the miss ratios for equal sized B-byte block and $B/2$-byte block caches will differ.

Figures 3c and 3d show maximum relative increases. Note that the scales of the two figures differ by a factor of ten. Again, the lines for block size 16 are as in Figure 2. No qualitative differences are found here, either. As with the mean, the maximum relative increase is approximately proportional to the block size.

4.2. Double Faults

Figure 4 shows relative increases for caches with two faults. Figures 4a and 4b show the means, while Figures 4c and 4d show the maxima. Figures 4a and 4c require that the two faults be placed in different sets (fault vectors $\mathbf{F}_{\mathbf{2,diff}}$), whereas Figures 4b and 4d put them in the same set ($\mathbf{F}_{\mathbf{2,same}}$), and thus omit direct-mapped caches which have only one block per set.

If the two faults are distributed independently, then Figures 4a and 4c are most relevant, since almost all pairs of independently distributed faults land in different sets (see Section 2.2.3). For this reason, data for two faults placed anywhere ($\mathbf{F}_{\mathbf{2}}$) are indistinguishable from the $\mathbf{F}_{\mathbf{2,diff}}$ case, and we do not display it seperately. The mean relative increases follow the same qualitative trend as for the single fault case. As predicted by Equation (5), the values tend to be twice those of the single fault case. The maximum relative increases with two faults in different sets are smaller than corresponding single fault numbers, since the sum of the two largest $D_i(n)$'s is almost always less than twice the largest $D_i(n)$.

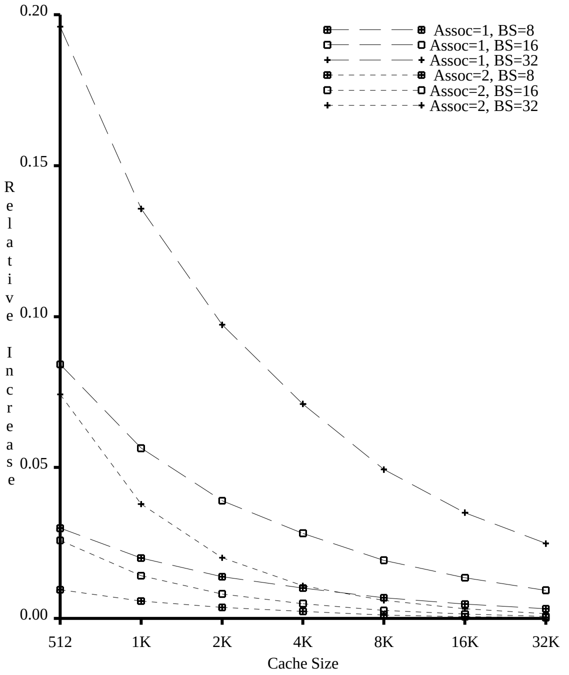

Figure 3a Mean relative miss ratio increase with one fault for associativities of 1 and 2 and various block sizes

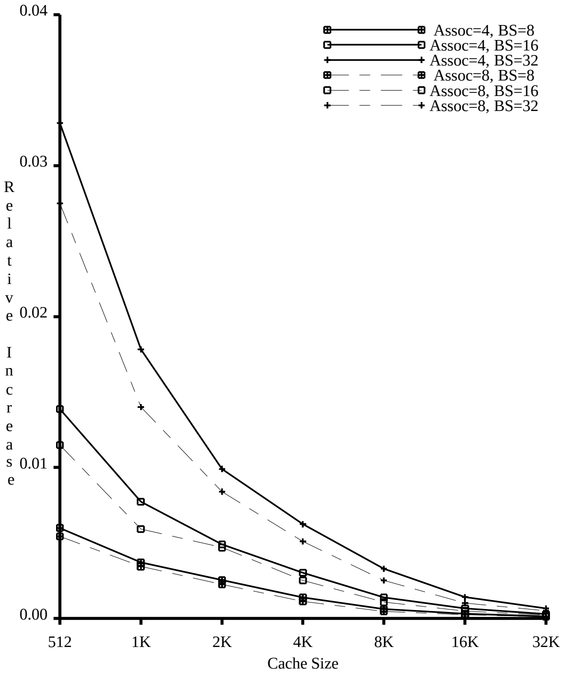

Figure 3b Mean relative miss ratio increase with one fault for associativities of 4 and 8 and various block sizes

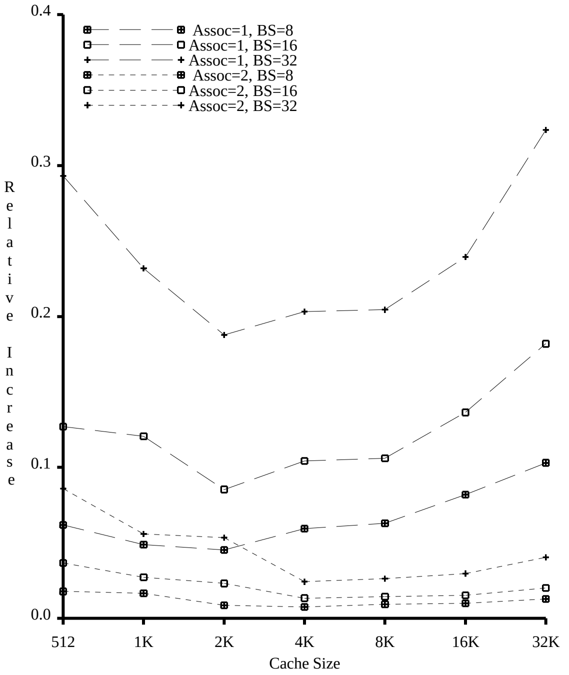

Figure 3c Maximum relative miss ratio increase with one fault for associativities of 1 and 2 and various block sizes

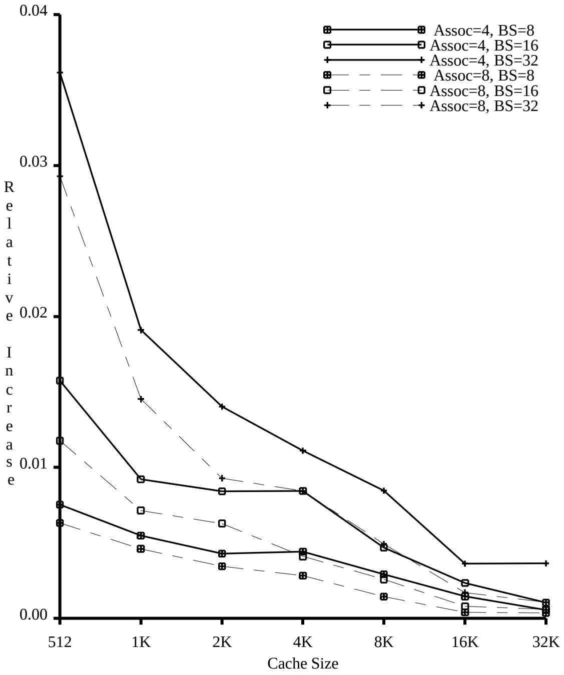

Figure 3d Maximum relative miss ratio increase with one fault for associativities of 4 and 8 and various block sizes

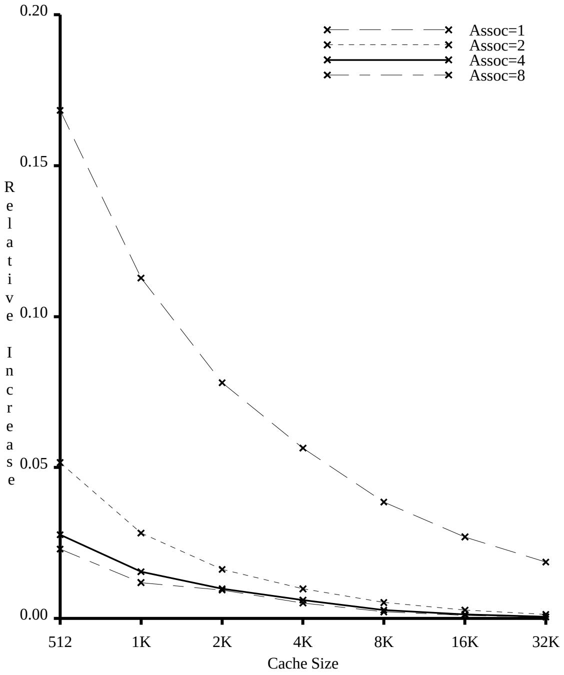

Figure 4a Mean relative miss ratio increase for two faults in different sets

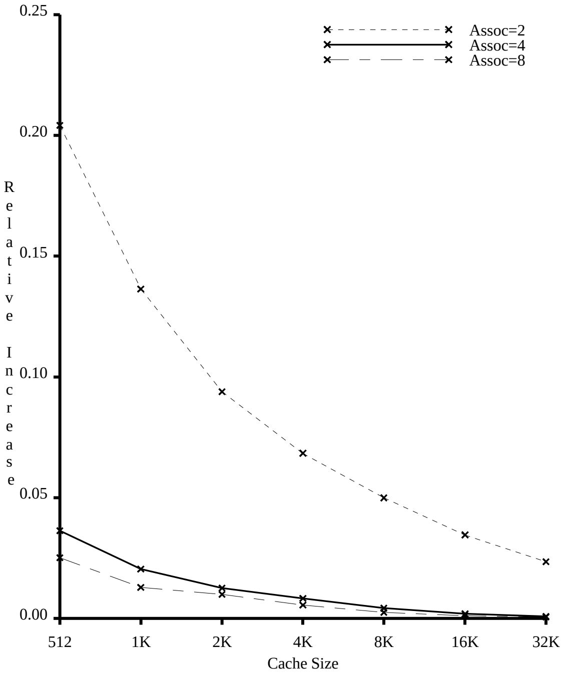

Figure 4b Mean relative miss ratio increase for two faults in the same set

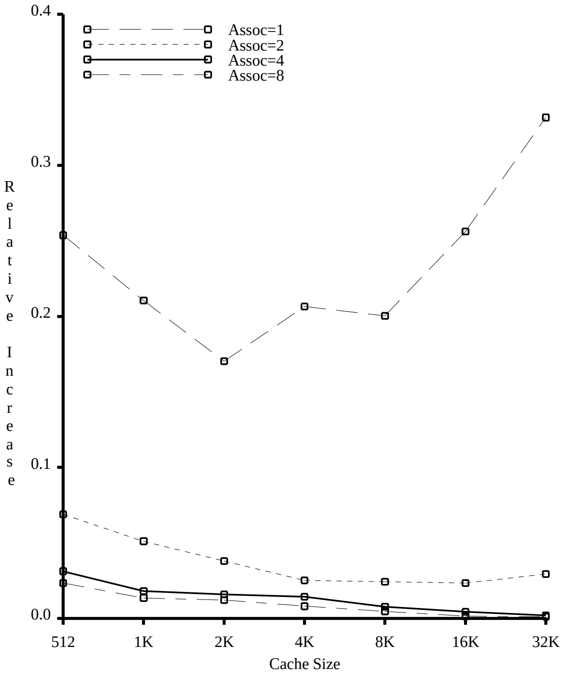

Figure 4c Maximum relative miss ratio increase for two faults in different sets

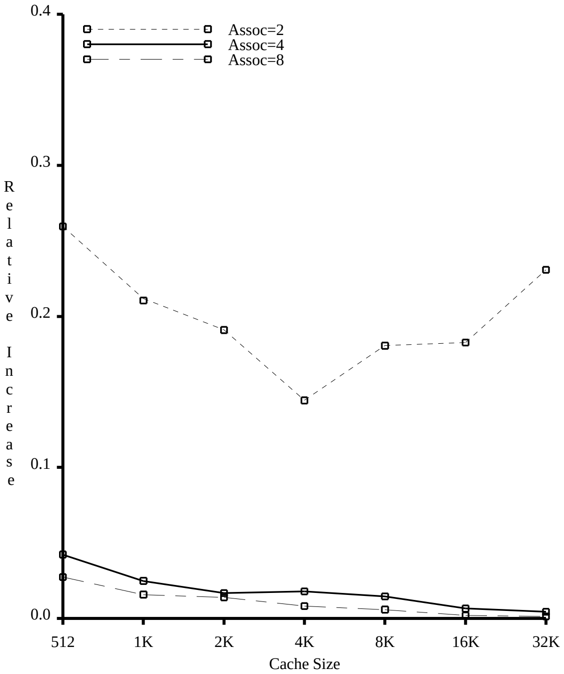

Figure 4d Maximum relative miss ratio increase for two faults in the same set

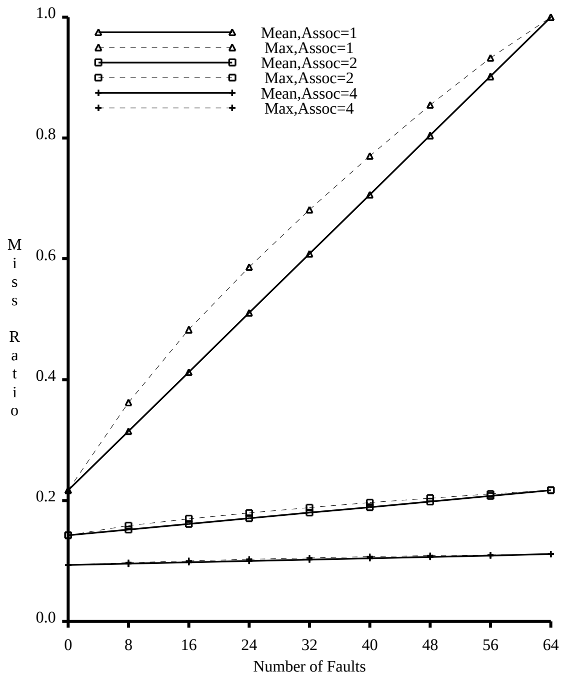

Figure 5a Mean and maximum miss ratios with numerous faults, but with at most one fault per set

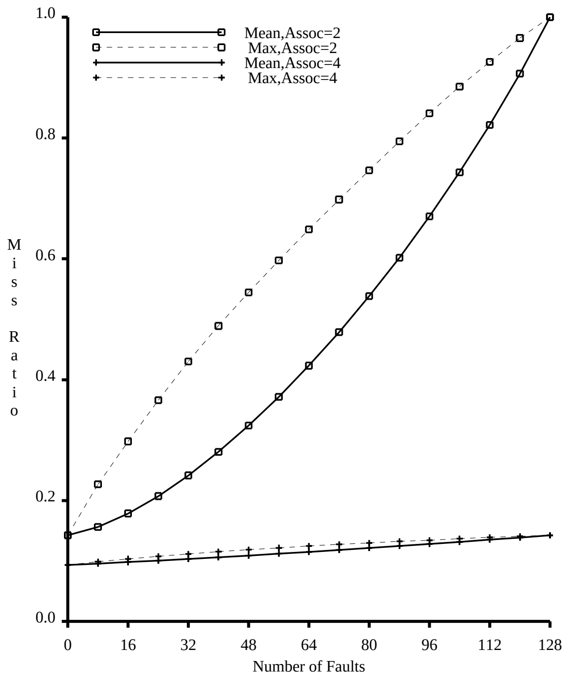

Figure 5b Mean and maximum miss ratios with numerous faults, but with at most two faults per set

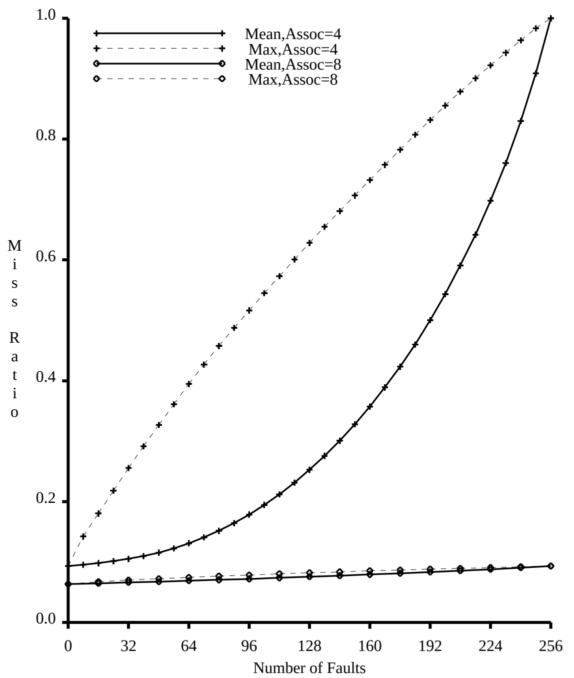

Figure 5c Mean and maximum miss ratios with numerous faults, but with at most four faults per set

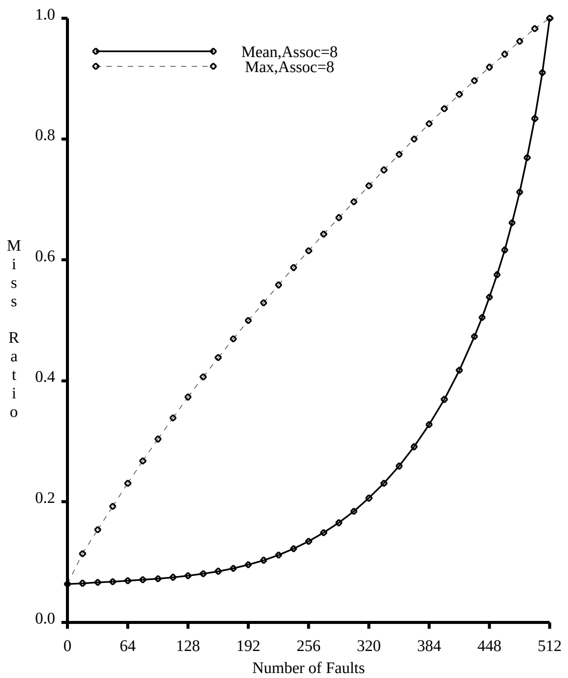

Figure 5d Mean and maximum miss ratios with numerous faults, but with at most six and eight faults per set; associativity is 8

Two faults in the same set (Figures 4b and 4d) are likely to arise only if a single failure causes both faults (e.g., a control signal to both blocks fails). The principal effect of moving the two faults from different sets to the same set is on two-way set-associative caches. Since these caches now have an entire set disabled, they tend to behave like direct-mapped caches with one fault (which also have an entire set disabled). As these caches get larger, mean relative increases no longer approach zero (Figure 4b) and maximum increases get larger.

4.3. Multiple Faults

Figure 5 examines the impact of many faults on the maximum and mean miss ratios for a cache with 16-byte blocks and 64 sets. The maximum number of faults allowed per set is one, two, four and eight in Figures 5a, 5b, 5c and 5d, respectively. Each figure varies associativity from the maximum number of faults allowed per set to eight, and we calculate each data point from all possible relevant distributions of faults. Data for smaller numbers of faults is more important than that for large numbers, since chips with large numbers of faulty bits are more likely to have destructive failures that force them to be discarded. Since the graphs depict caches with the number of sets constant, caches of associativity $2n$ are twice as large as those with associativity n. This limits the utility of comparisons between different associativities.

Figure 5a displays data for at most one fault per set. As predicted by Equation (5), mean miss ratio increases linearly with the number of faults at a slope of $[(m(n-1) - m(n))]/s$. The slope for the direct-mapped cache is the largest, since an entire set is disabled with each fault. Maximum miss ratios are equal to mean miss ratios at the endpoints, where none or all of the sets have a faulty block. Between the endpoints, maximum values are modestly worse than mean values.

Mean miss ratios with more faults per set begin with the same slope as above (since with the first fault there is a maximum of one fault in a set), but than increase according to a polynomial with degree bounded by the maximum number of faults per set. Maximum miss ratios are much worse than mean ratios whenever the maximum number of faults per set equals the associativity, allowing an entire set to be disabled.

5. Conclusions

We have attempted to provide insight into the effect on cache miss ratio of tolerating non-critical faults in an on-chip microprocessor cache. Since a cache is a non-critical resource − it is primarily used to increase the performance of, not ensure the correct operation of, a processor − it becomes a reasonable alternative to use a microprocessor chip that contains processing faults in the data blocks or tag bits. Doing so increases effective chip yield, and therefore reduces chip cost. While chips with disabled cache blocks will suffer larger miss ratios, they may still produce a faster memory system than chips that tolerate cache faults by suffering the access time overhead introduced by error correcting codes or redundant row or columns.

We first showed how the miss ratio of a cache with any fault pattern can be calculated from the number of references to the j-th block in the i-th of s sets. We then described how to extend all-associativity simulation [5] to calculate these metrics for many caches with a single pass through an address trace. Finally, we applied this technique to the ATUM traces [1].

Results suggest that the *mean relative miss ratio increase* from a few faults is negligible if no sets are completely disabled and is small in any case (< 5% per fault). Consequently, it is likely that the effective access time of a cache with some blocks marked faulty will be less than that for the alternate methods of tolerating cache faults.

The *maximum* relative miss ratio increase for a single cache fault, or for two cache faults in distinct sets, is acceptable if the associativity of the cache is two or greater and the block size is 8 or 16 bytes. Larger block sizes suffer greater penalties with permanent block invalidations. With a direct-mapped cache, however, there is a probability (albeit small with a large number of sets) that the executing program heavily references the faulty block(s), severely degrading the cache's performance. We expect that the overall impact of this worst-case behavior will not be significant for machines used to run many different programs.

6. Acknowledgments

We thank the Condor project at the University of Wisconsin [9] for providing us with the computational resources required for the large number of lengthy simulations performed and G. Sohi for reading and improving drafts of this paper.

References

1. Agarwal, A., Sites, R., and M. Horowitz, ''ATUM: A New Technique for Capturing Address Traces Using Microcode,'' *Proc. the 13th Annual Symposium on Computer Architecture*, pp. 119 - 129, Tokyo, Japan, June 1986.

2. Agarwal, A., Horowitz, M., and J. Hennessy, ''An Analytical Cache Model,'' *ACM Trans. on Computer Systems*, vol. 7, no. 2, pp. 184 - 215, May 1989.

3. Berenbaum, A. D., Colbry, B. W., D. R. Ditzel, R. D. Freeman, H. R. McLellan, K. J. O'Connor, and M. Shoji, ''CRISP: A Pipelined 32-bit Microprocessor with 13-kbit of Cache Memory,'' *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 5, pp. 776 - 782, October 1987.

4. Goodman, J. R., ''Using Cache Memory to Reduce Processor-Memory Traffic,'' *Proc. Tenth International Symposium on Computer Architecture*, pp. 124 - 131, Stockholm, Sweden, June 1983.

5. Hill, M. D. and Smith, A. J., ''Evaluating Associativity in CPU Caches,'' *IEEE Trans. on Computers*, vol. C-38, no. 12, pp. 1612 - 1630, December 1989.

6. Horowitz, M., Chow, P., D. Stark, R. T. Simoni, A. Salz, S. Przybylski, J. Hennessy, G. Gulak, A. Agarwal, and J. M. Acken, ''MIPS-X: A 20-MIPS Peak, 32-bit Microprocessor with On-Chip Cache,'' *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 5, pp. 790 - 799, October 1987.

7. Jouppi, Norman P., ''Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,'' *Proc. 17th Annual Symposium on Computer Architecture, Computer Architecture N ews*, vol. 18, no. 2, pp. 364-373, ACM, June 1990.

8. Kadota, H., Miyake, J., I. Okabayashi, T. Maeda, T. Okamoto, M. Nakajima, and K. Kagawa, ''A 32-bit CMOS Microprocessor with On-Chip Cache and TLB,'' *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 5, pp. 800 - 807, October 1987.

9. Litzkow, M., Livny, M., and M. W. Mutka, ''Condor — A Hunter of Idle Workstations,'' *Proceedings of the 8th International Conference os Distributed Computing Systems*, San Jose, California, June 1988.

10. Mattson, R. L., Gecsei, J., D. R. Schultz, and I. L. Traiger, ''Evaluation Techniques for Storage Heirarchies,'' *IBM Systems Journal*, vol. 9, no. 2, pp. 78 - 117, 1970.

11. Patterson, D. A., Garrison, P., M. Hill, D. Lioupis, C. Nyberg, T. Sippel, and K. Van Dyke, ''Architecture for a VLSI Instruction Cache for a RISC,'' *The Tenth Annual Symposium on Computer*

Architecture, vol. 11, no. 3, pp. 108 - 116, Stockholm, Sweden, June 13-17, 1983.

12. Phillips, D., ‘‘The Z80000 Microprocessor,’’ *IEEE Micro*, pp. 23 - 36, December 1985.
13. Pour, Farid and Hill, Mark D., ‘‘Performance Implications of Tolerating Cache Faults,’’ Computer Sciences Technical Report #991, Univ. of Wisconsin, January 1991.
14. Smith, A. J., ‘‘Cache Memories,’’ *Computing Surveys*, vol. 14, no. 3, pp. 473 - 530, September 1982.
15. Sohi, G., ‘‘Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors,’’ *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 484 - 492, April 1989.