Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors

GURINDAR S. SOHI, MEMBER, IEEE

Abstract—**High-performance VLSI processors make extensive use of on-chip cache memories to sustain the memory bandwidth demands of the CPU. As the amount of chip area devoted to on-chip caches increases, we can expect a substantial portion of the defects/faults to occur in the cache portion of a VLSI processor chip.**

This paper studies the tolerance of defects/faults in cache memories. We argue that, even though the major components of a cache are linear RAM's, traditional techniques used for fault/defect tolerance in RAM's may neither be appropriate nor necessary for cache memories. We suggest a scheme that allows a cache to continue operation in the presence of defective/faulty blocks. We then present the results of an extensive trace-driven simulation analysis that evaluates the performance degradation of a cache due to defective blocks. From the results we see that the on-chip caches of VLSI processors can be organized such that the performance degradation due to a few defective blocks is negligible. We conclude that by tolerating such defects without a noticeable performance degradation, the yield of VLSI processors can be enhanced considerably.

Index Terms—**Cache performance, defect/fault conditions, on-chip cache memories, trace-driven simulation, VLSI processors.**

I. Introduction

ADVANCES in semiconductor technology have led to the development of high-performance single-chip VLSI processors. For such processors, an increase in CPU speed must be coupled with an increase in memory bandwidth. By far the most popular technique for improving memory bandwidth in general purpose processors is the use of cache memories. For a single-chip processor, this translates into the use of on-chip cache memories. Many recent single-chip processors use some form of on-chip cache to provide adequate memory bandwidth and reduced memory latency for the CPU [3], [4], [6], [7], [9], [12], [14], [17]. For example, the MIPS-X processor devotes more than half its chip area to an on-chip instruction cache [9]. We expect that, in the future, most single-chip VLSI processors will devote a sizeable fraction of their chip resources to cache memories.

An increase in the circuit density of single-chip processors is coupled with an increase in defects. Since a large fraction of chip area will be devoted to cache memories in the near future, we expect that a large fraction of defects in a VLSI processor chip will be present in the cache memory portion of the chip. Application of yield improvement models (such as the model presented in [10]) suggests that, if defects/faults in the area occupied by the on-chip cache can be tolerated without substantial performance degradation and/or increase in cost, the yield of single-chip VLSI processors can be enhanced considerably.

The crux of defect- and fault-tolerance techniques is the use of redundancy. Generally some form of redundancy is provided explicitly for defect/fault tolerance. Sometimes, redundancy may be introduced in a processor not for defect/fault tolerance but for performance enhancement. A cache memory is an example of such a "redundant" resource. Cache memory is "redundant" because the correctness of processor operation is not dependent upon the presence of the cache. A processor can still operate correctly, albeit with severely degraded performance, in the absence of an architecturally-invisible cache memory.

When "redundant" components are present in a processor, two important and interesting questions arise: 1) what defect/fault-tolerance technique should be used for "redundant" portions of a processor such as the cache memory and 2) what are the performance implications of such a scheme? The focus of this paper is to evaluate the effects of defects/faults in a cache memory and provide answers to the above questions in the context of a cache memory.

The outline of this paper is as follows. We start off by discussing the nature of defects in cache memories and see how they affect cache operation. We discuss the use of techniques that use additional redundancy to tolerate defects and see why their use may not be a good choice for cache memories, especially for the on-chip caches of VLSI processors. We present a technique that allows a cache to continue operation even though some of its blocks may be defective. Then we evaluate the performance of cache memories to determine if defective blocks cause any appreciable loss in performance. Next, we present a discussion of the related issues of a sector cache organization and operational faults. Finally, we present concluding remarks.

II. Defects in Cache Memories and Their Tolerance

A cache memory consists of several *blocks* or *lines* of data. Each cache block is occupied by data elements from a block of the memory. A block consists of several contiguous bytes of memory. As data are referenced by the processor, they are brought from the memory into the cache. Data from a memory block are present in the *data memory* or *data array* portion of the cache. Each block in the cache has an associated tag which is kept in the *tag memory* or *tag array* portion of the cache. The tag is used to distinguish between one of several memory

Manuscript received June 28, 1988; revised November 7, 1988. This work was supported in part by NSF Grant CCR-8706722

The author is with the Computer Sciences Department, University of Wisconsin, Madison, WI 53706.

IEEE Log Number 8826375.

blocks that map onto a cache block. A simplified description of cache operation follows; a detailed description can be found in a survey paper by Smith [19].

When the CPU generates a memory request, a portion of the address is used as a tag and is compared to the tag(s) stored in appropriate locations in the tag array. If a match results, we have a cache *hit* and the data are accessed from the corresponding location in the data array. If no match results, we have a cache *miss*. On a cache miss, the entire memory block is transferred from the memory to the cache and is then accessed from the cache. In more sophisticated *sector cache* organizations, a block could be subdivided into several *transfer blocks* [11], [19]. To simplify the discussion, we shall initially assume a nonsector cache organization (indeed, this is the more common case). Then, in Section IV-A, we shall discuss the implications of a sector cache organization.

A. Types of Cache Defects

Components of a processor such as registers, buses, control logic, and the ALU are critical to the functioning of the processor. Defects in such components are *critical defects* because the defect will lead to incorrect processor operation unless some action is taken to tolerate and/or correct such defects. Consider, for example, a defect in a register. Instructions that utilize the defective register have no alternate modes of operation without violating the architectural definition of the instruction and will fail unless means are provided to tolerate the defect. Likewise, an ability to tolerate defects in the main memory must also be provided. As mentioned earlier, cache memory is not an "essential" component of the processor as far as correct operation is concerned. Cache memory is present in a processor mainly for performance reasons. The processor will be able to operate in a correct, but degraded, fashion if parts (or all) of the cache memory are unavailable and if alternate means are provided to recover and access correct data. If data cannot be accessed from a defective cache block, it can always be recovered from the memory without violating the architectural definition of the instruction. We call defects in noncritical components, such as the cache, *noncritical defects*.

A majority of fabrication defects can be classified as random *spot defects* [20]. Our defect model assumes random spot defects. We also assume that the defective area is small enough so that a single defect affects only one block of the cache (though more than one bit in each block may be defective). If the defect occurs in the tag array of the cache, we call it a *cache tag defect* and if it occurs in the cache data array, we call it a *cache data defect*. A cache tag defect will not pollute the data stored in the cache, i.e., it will not pollute the contents of the cache data array, but it will affect the cache *hit* operation. Examples of incorrect operation due to a cache tag defect include: 1) a miss indication even though data for the block are present in the cache, 2) a hit indication even though the block is not present in the cache, and 3) a "multiple" hit resulting from several tags matching. A cache data defect does pollute the data in the cache data array but does not affect the tag array. Such a defect does not affect the cache hit operation but results in the access of incorrect data.

B. Use of Redundancy to Tolerate Defects in Cache Memories

Since redundancy is a popular way of enhancing the yield and reliability in several contexts, one might be tempted to use redundancy in the cache portion of the VLSI processor to enhance the yield. Both the major portions of the cache, i.e., the tag and the data arrays are linear RAM's and redundancy techniques that are useful for RAM's could easily be applied to a cache. These techniques fall into two broad categories: 1) spare resources and a reconfiguration mechanism to substitute the defective resource with a defect-free resource and 2) use of error checking and correction (ECC) codes to mask out defects within a resource. Below, we discuss both of these options in some more detail.

1) Spare Cache Blocks and Reconfiguration: A cache memory could be designed with spare cache blocks in the data and tag arrays. If a block is defective, it can be switched out and a spare block substituted in its place using electrical or laser fuses [13]. The overhead for doing so includes the additional chip area for the spare blocks and the additional logic needed to implement the reconfiguration.

While this overhead is not very significant, we would like to emphasize that there is no reason to have a "full" cache in order to ensure correct operation of the processor. A "full" cache is a cache with the same number of defect-free blocks available for use in the caching operation as a completely defect-free cache. If the cache can be designed to operate in the presence of defective blocks with a negligible performance degradation, the use of spare cache blocks is wasteful.

2) Error Checking and Correction (ECC): ECC techniques have been used widely to tolerate faults in memory systems [18]. A typical memory system uses a single error correcting double error detecting (SECDED) Hamming code to correct single errors and detect double errors in the memory system. To carry out the detection and correction process, redundancy in the form of check bits must be incorporated into the memory data word. The fault-tolerance capability of an ECC technique is determined by the number of check bits used.

ECC techniques can also be used to enhance the yield of memories by masking out defective bits [13]. Since the tag and data arrays of the cache are essentially linear RAM's, one might be tempted to use an ECC scheme to tolerate cache defects. However, ECC techniques have two forms of overhead: 1) the time penalty introduced by the ECC logic and 2) the additional RAM required to store the check bits. Let us consider the implications of these overheads.

Since the degradation in memory access time is a good indicator of the degradation in performance of a VLSI processor, let us consider the degradation in memory access time due to the ECC logic. A typical processing system that uses a high-performance VLSI processor would have at least three levels in its memory hierarchy (excluding backing store). These levels are: 1) the on-chip or level 1 cache that serves to reduce the latency of CPU requests, 2) an off-chip or level 2 cache that serves both to reduce the latency of off-chip memory requests and, in the case of a multiprocessor, to reduce the traffic on the interconnect [8], and 3) the main

memory. For such a three-level memory hierarchy, the effective memory access time as seen by the CPU is

$$T = h_1 t_1 + (1 - h_1) h_2 t_2 + (1 - h_1)(1 - h_2) t_m \quad (1)$$

where h_1 is the hit ratio and t_1 is the access time of the level 1 cache, h_2 is the hit ratio and t_2 is the access time of the level 2 cache, and t_m is the access time of the main memory.

Using (1), let us see how the use of ECC at various levels in the memory hierarchy affects the overall effective memory access time. Let us consider a processing system built using high-performance VLSI processors in which the relative access times of the level 1 cache, the level 2 cache, and the main memory are 1, 3, and 10 time units, respectively. These values are typical of processing systems built using single-chip processors such as the NS32532 processor [3]. To determine the time penalties due to ECC at various levels, we carried out a VLSI layout of an on-chip cache and a paper design of a level 2 cache and a main memory. A timing analysis indicated that SECDED ECC degraded the the access time of level 1 cache, the level 2 cache, and the memory by 20, 15, and 10 percent, respectively. Assuming these degradations in access times to be representative of a large class of high-performance processing systems, we computed the average memory access time as seen by the CPU.

Table I presents the average access time as seen by the CPU for three cases: 1) a small level 1 cache ($h_1 = 0.6$), 2) a medium level 1 cache ($h_1 = 0.8$), and 3) a relatively large level 1 cache ($h_1 = 0.9$). In all cases, the on-chip level 1 cache is backed up by a typical medium-sized off-chip level 2 cache ($h_2 = 0.95$).[1] The results in Table I are presented for varying degrees of ECC usage.

From Table I we can see that, in all cases, the use of ECC in the main memory does not affect the overall memory access time to any appreciable extent (a degradation of about 1 percent). If ECC is used in the level 2 cache, the overall memory access time is degraded slightly (3.9–9.8 percent over the no-ECC case) but the degradation is less severe if the level 1 cache is larger and has a higher hit ratio. Note that the level 2 cache and the main memory are built from several chips and ECC would be necessary for fault tolerance. Indeed, the use of ECC for the main memory is very desirable. The use of ECC is also desirable for the level 2 cache especially if the level 2 cache is a copy-back cache and is built from DRAM's.

However, the use of ECC in the level 1 cache degrades the overall memory access time significantly (17–18 percent over the no-ECC case). Therefore, the use of ECC in the on-chip cache for yield enhancement does not seem to be an attractive option for high-performance VLSI processors. Furthermore, in the absence of adequate cache-coherence algorithms for on-chip caches, the on-chip caches are generally used to cache read-only information (such as instructions) or are write-through caches. For a read-only or a write-through cache, correct information always exits elsewhere in the system (the level 2 cache or the memory) at all times. Therefore, a simple error-detection capability is all that is needed even for fault-tolerant operation of the level 1 cache (see discussion in Section IV-B) and a more complex ECC scheme for fault tolerance is wasteful.

[1] For a justification of the hit ratios of the level 1 caches, see Section III-B and for a justification of the hit ratios of the level 2 cache, see [19].

TABLE I
AVERAGE MEMORY ACCESS TIME FOR VARYING ECC USAGE

ECC Usage	$h_1 = 0.6$; $h_2 = 0.95$	$h_1 = 0.8$; $h_2 = 0.95$	$h_1 = 0.9$; $h_2 = 0.95$
No ECC	1.940	1.470	1.235
ECC only in Main Memory	1.960	1.480	1.240
ECC in Memory and Level 2 Cache	2.131	1.566	1.283
ECC at all levels	2.251	1.726	1.463

It is possible that the degradation in memory access time due to ECC could be reduced for the on-chip cache [15]. For example, data could be read from the on-chip cache assuming that no error exists and supplied directly to the CPU. The ECC computation could be carried out in parallel with the CPU's use of the data. If the ECC computation indicates an error, the CPU would be informed and the computation aborted. However, the additional RAM overhead still exists and let us consider that.

Ideally, ECC must be provided on the smallest writeable unit [15]. Since the smallest writeable unit in most processors is a byte, this implies the use of 1 parity bit for single error detection, 4 check bits for single error correction, and 5 check bits for SECDED for each byte in the cache. This per bit overhead can be reduced by maintaining ECC check bits at the word (16-bit) or double word (32-bit) level. However, doing so can complicate the access of data in the cache when only a byte needs to be accessed since ECC information must be computed for more than a byte [15].

Because of the time and space overheads associated with it, ECC techniques to tolerate defects in an on-chip cache may be of limited utility. Indeed, if a defect affects more than a single bit in a cache block (as our defect model allows), the RAM overhead for storing the check bits for a multiple-error-correcting ECC code can be very large. Even if the overheads are tolerable, we would like to know if they are worthwhile. Therefore, we would like to see how a cache can operate in the presence of defective blocks and how the performance of the cache would be degraded in such a case.

C. Operation with Defective Blocks

To operate in the presence of defective blocks, the cache control logic must be able to distinguish between defective and defect-free blocks. To do so, we append to each block of the cache an *availability bit*. This bit is similar to the *fault-tolerance* bit proposed for the RISC-II instruction cache [16]. When the cache is tested, the availability bit for a block is set if the block is free of defects and is reset if a defect exists in the block. The defect can either be a cache data defect or a cache tag defect (in Section IV-A, we shall see how these bits and their setting change for a sector cache organization). The cache control logic makes use of the availability bit when it makes decisions during cache operation. The defective block is excluded from cache operation, that is, it is never chosen as the target block by the cache placement algorithm. If a

reference maps onto a defective block, the reference is treated as a miss.

When data are fetched from the memory, they are normally supplied to the CPU through the cache. If the cache is set associative and there is at least one defect-free block in each set of the cache, data transfer between the CPU and the memory can be carried out through a defect-free block. However, if all the blocks in a particular set are defective, data references that map onto the defective set cannot be carried out through the cache. This problem is particularly significant in a direct mapped cache where there is only one block in each set. To overcome this problem, selective bypass of the cache must be possible. We believe that this selective bypass capability is not a significant problem. Most processors have an option to turn off the on-chip cache thereby bypassing the cache for all memory references. Since the basic data paths already exist, extending the capabilities to allow selective bypass is straightforward.

The approach of using an availability bit to allow cache operation has little overhead—a single bit for each cache block. The cache can continue operation in the presence of cache tag and cache data defects. However, correct operation cannot be guaranteed in the rare case in which the defect exists in an availability bit. If the cache can be organized so that the degradation in performance due to defective blocks is negligible, this approach can be used profitably to enhance the yield of a VLSI processor with an on-chip cache.

III. Cache Performance Under Defect Conditions

In this section, we evaluate the performance of various cache organizations in the presence of defective blocks, where cache performance is measured by the miss ratio. First, we see how sensitive a cache organization is to a defective or missing block and then we carry out a detailed performance evaluation using trace-driven simulation.

A. The Sensitivity of a Cache Organization to Defective Blocks

Let us suppose that the memory consists of M blocks, the cache consists of C blocks, and the set associativity (number of blocks per set) of the cache is S. For this organization, there are $(M \times S)/C$ blocks in an *equivalence* or *congruence* class. All blocks from the same equivalence class are mapped onto the same set of the cache, i.e., the $(M \times S)/C$ blocks of an equivalence class are mapped onto one of S blocks in the cache.[2]

In a direct mapped cache, the set associativity is one ($S = 1$) and if a cache block is defective, M/C memory blocks are excluded from the cache. Consider, for example, the cache-memory system of Fig. 1. The cache has four blocks ($C = 4$) and the memory has 16 blocks ($M = 16$). If the cache were direct mapped ($S = 1$), under normal operation four memory

[2] So far we have assumed the presence of two levels of cache in the memory hierarchy. The reader should note that if the caches in the hierarchy have inclusion properties [5], that is, the contents of the level 1 cache are a subset of the contents of the level 2 cache, the presence of the level 2 cache does not affect the mapping of memory blocks in the level 1 cache. Therefore, to simplify our examples in this section, we shall use a single level cache in the memory hierarchy.

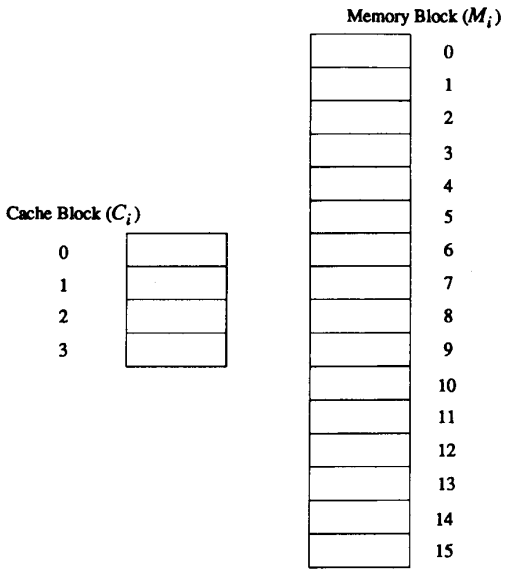


Fig. 1. An example cache-memory system.

blocks, namely $\{M_0, M_4, M_8, M_{12}\}$, map onto cache block C_0. If cache block C_0 is defective, four memory blocks $\{M_0, M_4, M_8, M_{12}\}$ will be excluded from the cache.

Since memory blocks $\{M_0, M_4, M_8, M_{12}\}$ cannot be present in the cache, references to these blocks must be serviced by the CPU–memory interface directly without passing the data through the cache. Therefore, any reference to these blocks would be a miss. In general, if there are D defective blocks in the direct mapped cache, $(D \times M)/C$ memory blocks would be excluded from the cache. Therefore, we can expect the miss ratio of a direct mapped cache to degrade linearly with the number of defective blocks.

A set associative cache is less restrictive. A single defective block does not automatically exclude any memory block from the cache. In fact, as long as every set in the cache has at least one defect-free block, no memory block is excluded from the cache. Blocks from a congruence class are excluded only if all the cache blocks of the corresponding set are defective. However, the miss ratio will degrade because the probability of interference among the blocks that map onto a set with defective blocks increases. For example, suppose that the cache of Fig. 1 were two-way set associative ($S = 2$) and cache blocks $\{C_0, C_1\}$ comprised set 0 of the cache. Under normal operation, memory blocks $\{M_0, M_2, M_4, M_6, M_8, M_{10}, M_{12}, M_{14}\}$ could be present in either cache blocks C_0 or C_1. A defect in cache block C_0 will not exclude any memory block from the cache completely; however, the probability of interference among the memory blocks that map onto set 0 of the cache will increase. Since no memory block is excluded from the cache unless all blocks in a set are defective, we can expect the degradation in miss ratio for a set associative cache to be less than the degradation in miss ratio for an equivalent direct mapped cache.

A fully associative cache ($S = C$) always allows every memory block to be cached (unless the entire cache is defective). Furthermore, all memory blocks are treated uniformly and no set of memory blocks experiences a greater

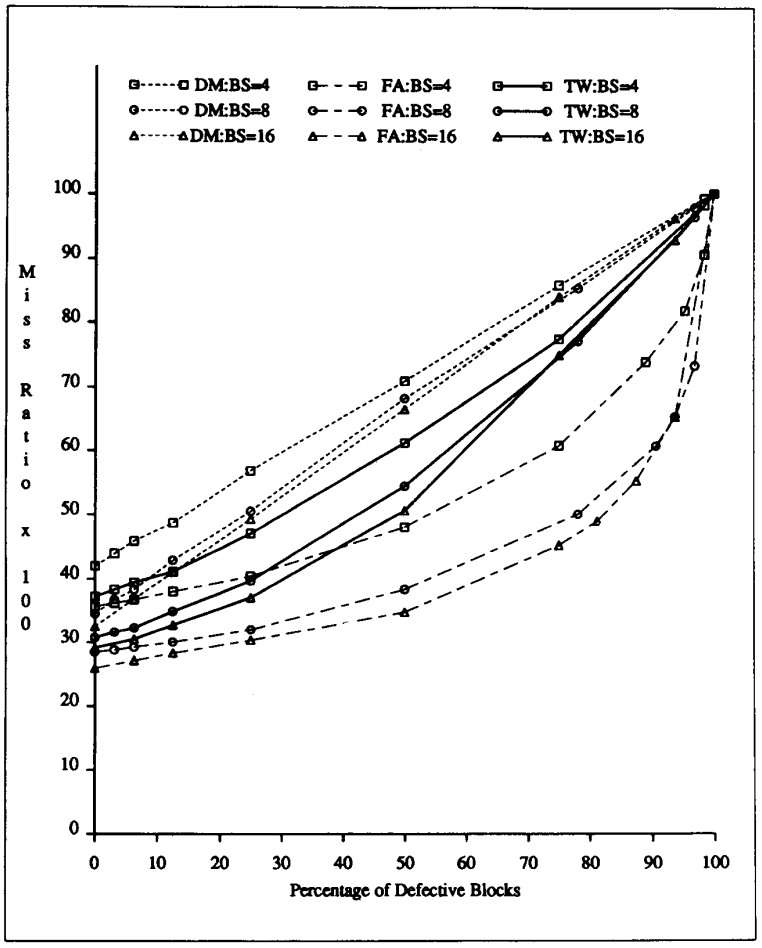


Fig. 2. Miss ratios; cache size = 256 bytes.

interference than another set. The degradation in the miss ratio, therefore, will be due solely to the increased probability of interference. The probability of interference is small, especially for larger cache sizes. Therefore, we expect a fully associative cache to have little degradation in miss ratio because of defective blocks, especially if the number of available defect-free blocks is large.

In summary, based on our understanding of cache operation in the presence of defective blocks, we expect that a direct mapped cache will be more sensitive to defects than a set associative cache with the same cache and block size. Furthermore, we also expect a cache with a larger number of blocks to be less sensitive to defective blocks than a cache with fewer number of blocks.

B. Simulation Methodology

To get an accurate estimate of degradation in cache performance (as measured by the miss ratio) due to defective blocks, we carried out an extensive trace-driven simulation analysis. Trace-driven simulation is the most popular way of evaluating cache memory performance. We simulated three different cache sizes: 1) a 256 byte cache which is a typical on-chip cache size for VLSI processors of the early- to mid-1980's (such as the Motorola 68020), 2) a 1K byte cache which is a typical on-chip cache size for VLSI processors of the mid- to late-1980's, and 3) an 8K byte cache which we expect be a typical on-chip cache size for high-performance VLSI processors of the near future. A direct mapped, a two-way set associative, and a fully associative organization were simulated for each cache size. A least recently used (LRU) replacement strategy was used for the set and fully associative organizations. The block size was also varied for each cache.

The benchmark programs used to simulate the caches were taken from the widely-used traces generated for a VAX-11/780 using the ATUM trace technique [1]. Each cache organization was simulated for approximately 1 million references. The caches were unified instruction and data caches. The simulations were carried out using a software cache simulator. We assume that the defects in the cache occur randomly, i.e., there is no clustering of defects. The simulator injects defects at random. A defect has the effect of preventing any data from being cached in the defective block. Since various blocks of the cache are not accessed precisely in the same fashion, two different caches with the same number of defective blocks (but different defective blocks) may differ slightly in performance. In order to overcome this problem, we simulated each cache organization several times for the same number of defective blocks but with a different set of defective blocks for each run and averaged the miss ratios.

The results of our simulation are presented in Figs. 2–4.

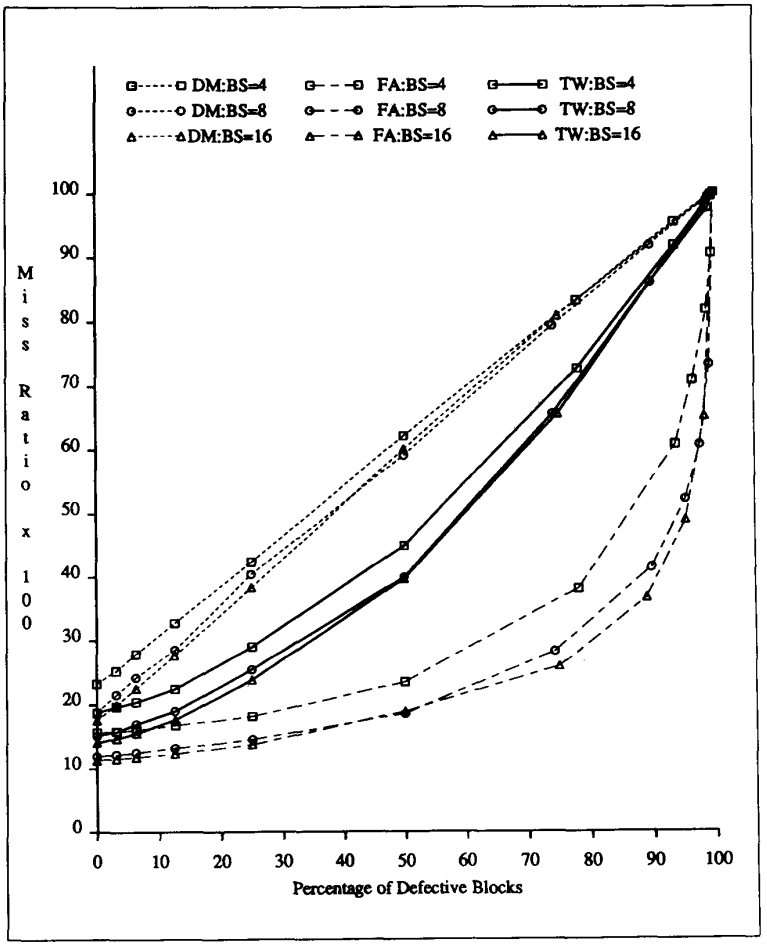


Fig. 3. Miss ratios; cache size = 1K bytes.

The figures plot the cache miss ratio (averaged over all traces) versus the percentage of blocks that are defective for direct-mapped (DM), two-way set associative (TW), and fully associative (FA) caches with various block sizes (in bytes). For the 256 byte and 1K byte caches, we have plotted the complete range of defective blocks. For the 8K byte cache, we have truncated the curves at 50 percent defective blocks to allow for a better look at the miss ratio degradation, especially for a fully associative cache.

C. Discussion of the Simulation Results

Consider the results for a 256 byte cache (see Fig. 2). If the cache is organized as a direct mapped cache, the miss ratio would degrade almost linearly with the number of defective blocks. This is indeed what we had expected. Thus, if the block size was 16 bytes (BS = 16), four defective blocks would imply that 25 percent of the blocks were defective and the miss ratio would degrade from about 0.325 to about 0.493. If, on the other hand, the cache were organized as a fully associative cache, the miss ratio would degrade only from about 0.259 to about 0.304. The degradation in miss ratio for other set associative organizations would be in between the two limits. For a two-way set associative organization, the miss ratio would degrade from about 0.307 to about 0.397. If the block size is small (4 bytes) and there is only a single defective block (1.56 percent of all blocks are defective), the miss ratio would degrade by only 0.002 (from 0.357 to 0.359) for a fully associative cache and by 0.01 (from 0.42 to 0.43) for a direct-mapped cache.

The results for cache sizes of 1K and 8K bytes (Figs. 3 and 4) follow a similar pattern. However, because of a large number of total blocks, more defect-free blocks are available for caching operation and the absolute degradation in miss ratio is much smaller. For example, in a direct mapped 8K byte cache with a block size of 8 bytes, four defective blocks (1.56 percent of all blocks) would degrade the miss ratio by only 0.0037 (from 0.0689 to 0.0726). If the 8K byte cache is two-way set associative with 8 byte blocks, the degradation in miss ratio due to four defective blocks would have only been 0.001. For a fully associative organization, the degradation is negligibly small.

An interesting point to note from Fig. 4 is that for a fully associative 8K byte cache, a loss of 50 percent of its blocks would only degrade the miss ratio from 0.054 to 0.064 if the block size is 8 bytes. Since a fully associative 8K byte cache with 50 percent of its blocks defective is essentially the same

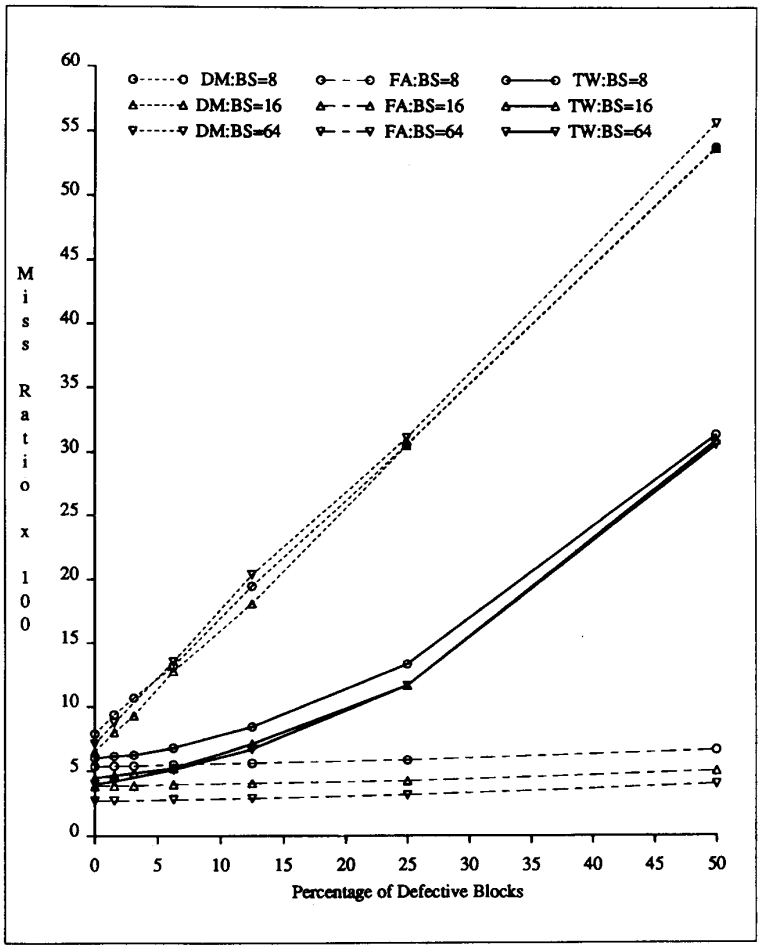


Fig. 4. Miss ratios; cache size = 8K bytes.

TABLE II
RELATIVE MEMORY ACCESS TIME IN THE PRESENCE OF DEFECTS

Block Size (bytes)	Number of Defective Blocks	Direct Mapped Cache Size (bytes)			2-way Set Associative Cache Size (bytes)			Fully Associative Cache Size (bytes)		
		256	1K	8K	256	1K	8K	256	1K	8K
8	0	1.812	1.443	1.185	1.722	1.356	1.142	1.669	1.281	1.126
	1	1.864	1.458	1.188	1.743	1.360	1.142	1.676	1.282	1.126
	2	1.901	1.475	1.190	1.759	1.363	1.142	1.686	1.284	1.126
	4	2.007	1.507	1.194	1.820	1.372	1.142	1.706	1.287	1.126
16	0	1.764	1.413	1.154	1.684	1.332	1.106	1.608	1.266	1.090
	1	1.866	1.427	1.158	1.717	1.336	1.106	1.637	1.267	1.090
	2	1.963	1.441	1.162	1.769	1.340	1.106	1.686	1.268	1.090
	4	2.158	1.469	1.171	1.872	1.346	1.106	1.713	1.271	1.090

as a fault-free, fully associative 4K byte cache, the reader might be tempted to conclude that a 4K byte fully associative cache is adequate; additional chip resources utilized in going to an 8K byte cache might be better utilized elsewhere. However, we would like to point out that the results for the fully associative cache have been presented as an upper bound. In many cases, fully associative caches are prohibitively expensive to implement and most practical caches employ a set associative organization.

Since the results presented in Figs. 2–4 do not indicate the degradation in the memory access time as seen by the processor, we converted the degradation in the miss ratio of the level 1 cache to a degradation in the relative memory access time as seen by the processor. We assumed the same system parameters as in Section II-B-2, i.e., the level 2 cache has a hit ratio of 0.95 and a relative access time of three cycles and the main memory has a relative access time of ten cycles. The results for block sizes of 8 and 16 bytes and various cache organizations are presented in Table II.

From the results presented in Figs. 2–4 and Table II, we can make two observations about cache behavior in the presence of defective blocks. First, for an arbitrary cache organization, the performance degradation due to defect is larger if the fraction of cache resources that it affects is larger. For example, the

degradation due to a single defective block in a cache with a block size of 8 bytes is smaller in a 1K byte cache (0.78 percent defective blocks) than in a 256 byte cache (3.125 percent defective blocks). The fraction of cache blocks that are unavailable due to defects can be reduced by increasing the cache size and/or decreasing the block size. However, smaller block sizes mean more tags and consequently more chip area. As we shall see in Section IV-A, a *sector cache* organization provides a good middle ground.

Second, for an arbitrary cache size, the performance degradation is smaller if the cache has a higher degree of set associativity. Increasing set associativity not only improves defect-free cache performance, it also enhances the ability of the cache to tolerate defects independent of or in conjunction with the cache size and the block size.

IV. Related Issues

A. Sector Caches

The cache organizations that we have discussed so far have a single block size and fetch the entire block into the cache on a miss. In general, large block sizes are preferable because they reduce the number of tags required (and consequently the size of the tag array RAM). However, large block sizes have two problems that limit their use: 1) additional traffic is generated on the cache–memory interconnection and 2) the time penalty incurred on a miss to fetch the block into the cache is larger. The first problem is significant for a multiprocessor with private cache memories [8] and the second is significant for a VLSI processor with an on-chip cache memory [2]. To alleviate these problems, a sector cache organization can be used.

In a sector cache, an *address block* is divided into several *transfer blocks* [11], [19]. Tags for an entire address block are maintained in the tag array. An additional bit called a *presence bit* is maintained for each transfer block in the data array. A reference is a miss either if the address block is not present in the cache or if the desired transfer block within the desired address block is not present in the cache. On a miss, only the desired transfer block is brought into the cache. By having large address blocks and small transfer blocks, we can reduce the cache–memory traffic as well as minimize the penalty incurred on a miss.

A sector cache can be made defect-tolerant in the same way as a regular cache—by the use of a single availability bit for each address block. The availability bit is reset if a defect exists in the tag portion (address block) or in either of the data portions (transfer blocks) of a particular block. However, if a cache data defect is confined to a small number of transfer blocks from a particular address block, the performance degradation can be reduced even further by associating an availability bit with each transfer block in the data array. In this case, there is a total of $T + 1$ availability bits where T is the number of transfer blocks in an address block. Now, in case of a single cache tag defect, an entire address block (T transfer blocks) is unavailable for use and the performance degradation that we can expect would be similar to the performance degradation for a nonsector cache. However, in case of a single cache data defect, only a single transfer block is unavailable for use (the other $T - 1$ transfer blocks of the address block can still be used for caching). Since the "scope" of the defect is much smaller, the degradation in performance will also be smaller.

B. Operational Faults

So far we have focused our attention on defects that can be detected by a testing procedure that is applied before the VLSI chip is put into operation. During operation, faults can occur and, if tolerating such faults is important, a fault-tolerance mechanism must be provided. The mechanism must be able to detect that a fault has occurred and take corrective action.

For our purposes, we assume that an operational fault manifests as a single bit error. Under this assumption, detecting a fault is quite straightforward and a simple parity scheme can be used to do so. If the cache is a copy-back cache, a correct copy of data is not guaranteed to exist elsewhere in the system at all times and, therefore, a correction capability is necessary if fault-tolerant operation is to be guaranteed. However, in the case of a cache for read-only data or a write-through cache, correct information can always be recovered from elsewhere and, therefore, a correction capability in the cache is not absolutely necessary in order to guarantee fault-tolerant operation.

The availability bit scheme coupled with an error detection mechanism can be used to guarantee fault-tolerant operation in write-through and read-only caches. When data are accessed from the cache, the error detection mechanism is triggered. If the mechanism indicates an error, the access is treated as a miss. The data are read from the memory (or level 2 cache) into the cache and the access operation is resumed. If an error condition is flagged again, a permanent (or intermittent) error must exist in the cache block. Since a permanent error is functionally equivalent to a defect, the availability bit of the block is reset and all future references to the block are treated as misses. If the error is transient, it is automatically scrubbed by the cache miss operation. In this manner, the cache can be used for fault-tolerant operation and its performance degradation due to faults is no worse than the performance degradation due to an equivalent number of defects.

For copy-back caches, correct operation cannot be guaranteed unless an ECC scheme is used. However, we can reduce the probability of an irrecoverable situation as follows. Organize the cache as a sector cache with a parity bit, a dirty bit, and an availability bit for each transfer block. A dirty bit is a bit which indicates if the contents of the block have been updated and if the contents of the block present in the cache differ from the contents that are present elsewhere in the processing system. As before, the availability bit is used to indicate a permanent error. Of the four possible combinations of the parity and dirty bits, only in one case can correct data not be recovered. This is the case when the dirty bit is set and the parity bit indicates an error. If the parity bit indicates no error, correct data exist in the cache block. If the parity bit indicates an error and the dirty bit is not set (indicating that the block has not been modified since it was last brought into the cache and a correct copy of the block exists elsewhere in the

system), correct data can be recovered from elsewhere. By having smaller transfer blocks, the probability of having an error in a dirty cache transfer block can be minimized and consequently the probability of an irrecoverable error can be minimized.

V. Conclusions

To achieve high performance in VLSI processors, the use of an on-chip cache memory is highly desirable. Since the on-chip cache is expected to consume a large portion of the chip area, techniques that allow a cache to continue operation in spite of defective blocks can profitably be used to enhance the yield of VLSI processors. In this paper, we investigated techniques to enhance the yield of on-chip cache memories. We saw that traditional techniques that use additional redundancy may neither be appropriate nor necessary since a cache can continue operation in the presence of defects. Then we suggested a scheme that allows a cache to continue operation in the presence of defective blocks.

We evaluated the degradation in cache performance due to defects using an extensive trace-driven simulation analysis. Our evaluation indicates that the performance degradation due to defective blocks is small in cases where: 1) the cache is large, 2) the block size is small, and 3) the set associativity is high. In many cases, the performance degradation is insignificant.

Two major conclusions can be drawn from the work presented in this paper: 1) by choosing an appropriate cache organization, the need for additional redundancy in a cache to allow defect/fault-tolerant operation can be avoided and 2) defects in the on-chip cache portion of a VLSI processor can be tolerated with a minimum performance loss and very small area overhead and consequently, the yield of high-performance VLSI processors can be enhanced considerably.

References

[1] A. Agarwal, R. L. Sites, and M. Horowitz, "ATUM: A new technique for capturing address traces using microcode," in *Proc. 13th Annu. Symp. Comput. Architecture,* Tokyo, Japan, June 1986, pp. 119–127.

[2] A. Agarwal, P. Chow, M. Horowitz, J. Acken, A. Salz, and J. Hennessy, "On-chip instruction caches for high performance processors," in *Proc. Conf. Advanc. Res. VLSI*, Stanford, Mar. 1987.

[3] D. Alpert, J. Levy, and B. Maytal, "Architecture of the NS32532 microprocessor," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 1987.

[4] D. W. Archer *et al.*, "A CMOS VAX microprocessor with on-chip cache and memory management," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 849–852, Oct. 1987.

[5] J.-L Bear and W.-H. Wang, "On the inclusion properties for multi-level cache hierarchies," in *Proc. 15th Annu. Symp. Comput. Architecture.* Honolulu, HI, June 1988, pp. 73–80.

[6] A. D. Berenbaum *et al.*, "CRISP: A pipelined 32-bit microprocessor with 13-kbit of cache memory," *IEEE J. Solid-State Circuits,* vol. SC-22, pp. 776–782, Oct. 1987.

[7] P. W. Bosshart *et al.*, "A 553K-transistor LISP processor chip," *IEEE J. Solid-State Circuits,* vol. SC-22, pp. 808–819, Oct. 1987.

[8] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," in *Proc. 10th Annu. Symp. Comput. Architecture*, June 1983, pp. 124–131.

[9] M. Horowitz *et al.*, "MIPS-X: A 20-MIPS peak, 32-bit microprocessor with on-chip cache," *IEEE J. Solid-State Circuits*, vol. SC-22, pp. 790–799, Oct. 1987.

[10] I. Koren and D. K. Pradhan, "Modeling the effect of redundancy on yield and performance of VLSI systems," *IEEE Trans. Comput.*, vol. C-36, pp. 344–355, Mar. 1987.

[11] J. S. Liptay, "Structural aspects of the System/360 Model 85 Part II: The cache," *IBM Syst. J.*, vol. 7, pp. 15–21, 1968.

[12] D. MacGregor, D. Mothersole, and B. Moyer, "The Motorola MC68020," *IEEE Micro,* pp. 101–118, Aug. 1984.

[13] W. R. Moore, "A review of fault-tolerant techniques for the enhancement of integated circuit yield," *Proc. IEEE,* vol. 74, pp. 684–697, May 1986.

[14] A. Patel, "An inside look at the Z80,000 CPU: Zilog's new 32-bit microprocessor," in *Proc. AFIPS Nat. Comput. Conf.*, July 1984, pp. 83–91.

[15] D. A. Patterson and C. H. Sequin, "Design considerations for single-chip computers of the future," *IEEE Trans. Comput.*, vol. C-29, pp. 108–116, Feb. 1980.

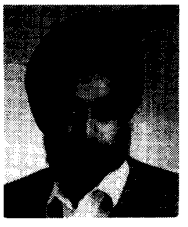
[16] D. A. Patterson, P. Garrison, M. Hill, D. Lioupis, C. Nyberg, T. Sippel, and K. Van Dyke, "Architecture of a VLSI instruction cache for a RISC," in *Proc. 10th Annu. Symp. Comput. Architecture*, June 1983, pp. 108–115.

[17] D. Phillips, "The Z80000 Microprocessor," *IEEE Micro*, pp. 23–26, Dec. 1985.

[18] D. K. Pradhan, *Fault Tolerant Computing: Theory and Techniques.* Englewood Cliffs, NJ: Prentice-Hall, 1986.

[19] A. J. Smith, "Cache memories," *ACM Comput. Surveys,* vol. 14, pp. 473–530, Sept. 1982.

[20] C. H. Stapper, F. M. Armstrong, and K. Saji, "Integrated circuit yield Statistics," *Proc. IEEE,* vol. 71, pp. 453–470, Apr. 1983.



Gurindar S. Sohi (S'85–M'85) received the B.E. (Hons.) degree in electrical engineering from the Birla Institute of Science and Technology, Pilani, India in 1981, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana-Champaign, in 1983 and 1985, respectively.

Since September 1985, he has been with the Computer Sciences Department at the University of Wisconsin–Madison where he is an Assistant Professor. His interests are in computer architecture, parallel and distributed processing, and fault-tolerant computing.