N94-18346

Performance of Defect-Tolerant Set-Associative Cache Memories[1]

J. F. Frenzel
Department of Electrical Engineering
University of Idaho, Moscow, Idaho 83843
jfrenzel@groucho.mrc.uidaho.edu, 208-885-7888

Abstract- **Increased use of on-chip cache memories has led researchers to investigate their performance in the presence of manufacturing defects. Several techniques for yield improvement are discussed and results are presented which indicate that set-associativity may be used to provide defect -tolerance as well as improve the cache performance. Tradeoffs between several cache organizations and replacement strategies are investigated and it is shown that token-based replacement may be a suitable alternative to the widely-used LRU strategy.**

1 Introduction

The dramatic increase in cache memory size and diminishing geometries has resulted in lower yields. Today's high performance processors often have on-chip cache and consequently the yield of these memories can be a significant factor in determining the ultimate cost of the processor. One way of increasing yields is to provide defect-tolerance through the use of redundant resources. Two methods for achieving defect-tolerance are commonly employed in the design of dynamic RAM (DRAM) memories, namely the use of error correcting codes and spare rows and columns [5]. However, both techniques result in increased circuitry and possible increases in access times.

Associative memories offer an alternative approach. By design, associative memories have the flexibility necessary to function in the prescence of defects. With the inclusion of control logic it is possible to force the memory to operate "around" the defect and use alternative locations, albeit with a reduction in storage capacity. In the following sections we will describe basic cache memory operation and then discuss the different techniques for providing defect-tolerance.

2 Cache Operation

A cache memory is a fast intermediary memory positioned between a processor and main storage. The goal of a hierarchical memory system is an average access time close to that of the cache memory, at a cost per bit approaching that of the main memory. To achieve the former the cache must be designed to keep the most frequently referenced items in the cache. A system may designed with separate caches for data and instructions or a single (unified) cache.

[1]This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

2.1 Organization

A cache memory is organized as sets of blocks, where each block is typically 4 to 16 bytes of data from main storage. In a direct-mapped cache each set consists of only one block, whereas in a n-way, set-associative (SA) cache each set contains n blocks. The total cache size is the product of the block size, the number of sets, and the associativity, n.

2.2 Address Translation

Address references to the cache are split into three fields, the widths of which depend upon the cache size and organization. The block field is used to index a particular item within a block and is $\log_2 b$ bits wide, where b is the number of addressable items within a block. If s is the number of sets in the cache, then the set field is $\log_2 s$ bits wide and is used to indicate a particular set for access. The remaining bits are referred to as the *tag* and are used to distinguish between other blocks of main memory which may be stored in the same set. Each block in a set has storage to hold both the block data and the tag associated with that block. The collection of tag storage for the cache is referred to as the *tag directory*. During a memory access, the tag field for the address is compared with all entries in the tag directory corresponding to the referenced set. If there is a match, the data from the matched block is sent to the processor. If there is no match, referred to as a miss, then the missed data must be loaded from main memory.

2.3 Replacement Policy

On a miss, the cache must decide where to place the block from main storage which caused the miss. For a direct-mapped cache the decision is trivial, as each block from main storage maps to a single block in the cache. However, with a set-associative cache, assuming the referenced set is full, there are n possible blocks to replace. One of the best replacement algorithms is referred to as *least recently used* (LRU), where the set is treated as a stack and accessing a particular block moves that block to the top of the stack. The least recently used block is always at the bottom of the stack and a miss will load the data into this block and move it to the top of the stack. Efficient implementations of the LRU replacement algorithm require $n(n-1)/2$ bits of storage per set to maintain the $n!$ possible stack configurations. Consequently, a 4-way SA cache requires 6 bits of storage per set, while an 8-way SA cache requires 28 bits per set. Additional circuitry is needed to update the stack configuration as a result of an access. Alternative replacement strategies are first in, first out (FIFO), and random. The FIFO algorithm is implemented using a modulo b counter for each set, incremented on every miss to that set. One technique for implementing a pseudo-random replacement strategy is to use a single modulo b counter for the entire cache and increment it on every miss, regardless of the set. This will be referred to as token-based replacement.

2.4 Discussion

Several observations may be made in comparing direct-mapped caches to set-associative caches. First, for a given cache size, the tag field and subsequently the tag directory will be larger for the SA cache. This is because the n-way, set-associative cache will have $1/n$ the number of sets as the direct-mapped cache, needing fewer bits in the set field, and increasing the number of bits in the tag field. Second, for the set-associative cache, n comparisons must be conducted in parallel between the tag field and the entries in the tag directory. Furthermore, the set-associative cache has an additional delay over the direct-mapped cache as a result of the need to multiplex the data from each of the blocks in the referenced set to the output. Lastly, the SA cache has additional circuitry needed to implement the replacement algorithm.

3 Defect-Tolerance Strategies

3.1 Spare Resources

There are several methods for implementing memory reconfiguration in the presence of defects: electrically programmable links, electron-beam programmable fuses, and laser cutting/welding [6]. These techniques can be employed to bypass faulty resources and activate spare units. The most common technique for increasing memory yields is to include spare rows and/or columns in the data array and sufficient programmable decoders. While all implementations increase the circuit area, some methods may also increase the access times and power dissipation [5]. Furthermore, unless special circuitry is added it is usually not possible to test the spare rows/columns without first programming the decoders.

It has recently been observered that manufacturing "throughput", measured in usable chips per unit time, is dominated by the delay associated with repairing defective parts rather than the process yield [2]. These researchers argue that efforts should be directed at maximizing the throughput, rather than the yield, and propose algorithms for achieving this by balancing repair time and yield of repaired parts. Previously, production experience with a 64K DRAM indicated that the repair algorithm typically took several seconds and represented roughly half of the entire test time [10]. The next two sections describe methods which eliminate the time needed to execute a repair algorithm.

3.2 Error Correcting Codes

Error correcting codes can be used to correct single or multiple errors in the tag directory and data array caused by manufacturing defects. Codes may be selected to provide a guaranteed level of protection at a corresponding increase in circuit area and access time. A 16-bit word would require 6 extra check bits to *detect* and *correct* all single errors.

In addition to storing the check bits, additional circuitry is needed to encode or decode during memory accesses. For large words, where the use of check bits is most efficient, the delay associated with this circuitry can be significant. Results of a timing analysis are presented in [11] which indicate a 20% increase in access time using single error correction, double error detection coding of the tag directory and data array. For these reasons, error correcting coding is generally reserved for applications which require tolerance of transient errors incurred during normal operation. However, Mostek built a 1-Mbit ROM with a 32-bit word that achieved a 3-fold improvement in yield at a 20% increase in area [8].

A distinct advantage of error correcting coding is the lack of any "repair" time. As mentioned earlier, the delay associated with this process can severely affect the manufacturing throughput for the part.

3.3 Associativity

Sohi observed that a cache memory does not have to be defect free to meet its objective, namely reduce the average memory access time of a hierarchical memory system [11]. A direct-mapped cache memory with a defective block will never be able to hold items from main memory which map to that set in the cache. For a cache to operate properly under this condition two things are necessary: one, the cache must be able to recognize a defective block and generate a miss and two, must have the capability of performing a load through, so that the processor can access the item. An associative cache has alternate locations within a set which can be used when there is a defective block present. Ideally, the circuitry which implements the replacement algorithm would be modified at test time to exclude defective blocks from selection during replacement. Provided each set has at least one good block all items from main memory can map to a good location in the cache.

4 Related Work

Patterson et al. described the implementation of a cache memory in which each cache block was provided with a *fault tolerant* bit, which could permanently invalidate a cache block. Set-associativity was achieved through the use of multiple chips and block replacement was directed by a token [7]. Accessing a bad block would result in a miss.

More recently, Bergh et al. designed a fully associative fault-tolerant memory. Extra logic, amounting to a 2% increase in area, allowed the memory to completely bypass defective locations transparent to the user [1].

Finally, Sohi investigated the performance under defects, as measured by miss ratio, of three different cache organizations: direct-mapped, 2-way set-associative, and fully-associative [11]. His research illustrated that it is possible for a 2-way set-associative cache, using a LRU replacement strategy, to outperform a direct-mapped cache of equivalent size in the presence of defects.

This paper attempts to extend the work of Sohi in evaluating the benefits of associativity for the purpose of defect-tolerance. In this paper I focus upon set-associative cache memories for the following reasons:

- fully-associative caches are generally not required for many applications and are prohibitively expensive;

% Change in Hits Compared to 2K, DM Cache							
Size	DM	2-way, SA			4-way, SA		
(words)		LRU	FIFO	Token	LRU	FIFO	Token
2K	0	2.2	1.7	1.8	3.5	2.8	2.9
4K	5.0	7.6	7.2	7.2	8.8	8.2	8.2
8K	9.6	12.0	11.7	11.8	13.1	12.6	12.7

Table 1: Performance of Defect-free Caches

- set-associative caches possess the flexibility necessary to reduce the impact of defects.

Specifically, this paper investigates set-associative caches of various organizations under three different replacement strategies, least recently used (LRU), first-in, first-out (FIFO) and token-based. The LRU strategy is widely accepted as the superior strategy, although costlier to implement [9].

5 Simulation Methods

Performance evaluation was conducted using address trace simulation. The address traces were generated from runs of SPICE, gcc, and TEX, for a total of over 2.8 million references, approximately 75% of which were instruction references [3]. All address references were assumed to reference items of the same size, namely one word. A wide variety of caches were studied; however, in all cases the block size was held at 8 words and the cache was treated as a unified cache (instructions and data).

Three different cache sizes were simulated, ranging in size from 2K words to 8K words. For each size, three different cache structures were investigated: direct-map, 2-way set-associative, and 4-way set-associative. Each associative cache was simulated using three different replacement strategies: least recently used (LRU), first in, first out (FIFO), and token-based. Lastly, each associative cache was simulated under three different levels of defects, ranging from zero to 25%.

During defect simulation each cache was simulated forty times, each iteration using a random distribution of defects. Furthermore, defect-levels were limited and the defects distributed such that each set was guaranteed to have at least one good block. The replacement strategies were modified from the traditional descriptions to prevent loading a missed block into a defective location.

6 Results

6.1 Defect-free Performance

Table 1 shows the percent change in the total number of hits for various cache organizations, relative to the total number of hits for a 2K, direct-mapped (DM) cache. From this data we can make several observations regarding the relative performances of various organizations under defect-free operation:

% Change in Hits Compared to 2K, DM Cache						
Size	2-way, SA			4-way, SA		
(words)	LRU	FIFO	Token	LRU	FIFO	Token
2K	-1.0	-1.4	-1.6	1.9	1.2	1.3
4K	5.7	5.4	5.2	7.5	6.9	6.8
8K	10.3	10.1	9.9	12.0	11.4	11.5

Table 2: Performance with 12.5% Defect-Level

- As cache size doubled there was approximately a 5% increase in hits, relative to a 2K, DM cache, across all structures and replacement algorithms. However, this effect would eventually diminish as the cache size approached that of the workload's working set.

- The token-based replacement algorithm was virtually identical in performance to the FIFO algorithm for all cache organizations. While at first this may seem surprising, neither algorithm is a "usage based" algorithm and consequently their performance is roughly equivalent.

- LRU was the best replacement strategy, increasing the performance by roughly 0.5% over the other algorithms. For a fixed cache size, the performance difference increased with associativity. As the number of blocks per set increased, LRU's superior management of those resources became more apparent. For a fixed associativity, *n*, the improvement decreased with increasing cache size. This may be attributed to reduced contention in the cache.

- Doubling the associativity increased performance by approximately 2%, with the improvement diminishing as the associativity increased. Again, this may be attributed to reduced contention within the cache.

As cache size increases, there is less contention for space in the cache and performance differences due to associativity and replacement strategies tend to diminish. The same is true for a fixed cache size as associativity increases, particularly under LRU replacement.

6.2 Performance under Defects

Tables 2 and 3 detail the results of simulating various cache organizations under two different defect-levels. As in the first table, the numbers represent the percent change in the total number of hits, relative to a defect-free, 2K, DM cache. At the 12.5% defect-level, there was a drop in performance that was a function of both cache size and associativity, but *not* replacement strategy. For example, all 2K, 4-way, SA caches experienced a decline of approximately 1.6% from their defect-free performance. This can be attributed to the fact that each replacement algorithm was modified such that missed data would never be loaded into a defective block. The average number of bad blocks per set is equal to the product of the associativity and the defect-level. So at a 12.5% defect-level a 2-way cache

% Change in Hits Compared to 2K, DM Cache						
Size	2-way, SA			4-way, SA		
(words)	LRU	FIFO	Token	LRU	FIFO	Token
2K	-4.1	-4.4	-4.7	0.2	-0.5	-0.4
4K	3.8	3.6	3.4	6.0	5.4	5.3
8K	8.5	8.4	8.0	10.7	10.2	10.1

Table 3: Performance with 25% Defect-Level

will have, on average, 0.25 bad blocks per set, or one bad block for every four sets, while a 4-way cache will average one bad block for every other line. Consequently, the effect of defects is to decrease the associativity. At low defect-levels, and particularly for low values of associativity, the decrease will be minor and thus the performance differences between replacement algorithms will remain approximately constant.

In general, the larger the associativity or the total cache size, the smaller the drop in performance due to defects. Increasing associativity or size are two methods for reducing contention in a cache and consequently it is expected that defects would have a lesser effect on these caches. Another important observation, is that all 4-way, SA, 2K caches, regardless of replacement algorithm, outperformed the defect-free, DM, 2K cache. Furthermore, for caches larger than 2K, all associative caches with a 12.5% defect-level outperformed a defect free DM cache of equivalent size. This is a clear example of using associativity to provide defect-tolerance *and* a performance improvement. At a defect-level of 25%, only the 4-way, set-associative caches outperformed the defect-free, DM caches.

Other researchers have suggested that the use of associative cache memory may be on the decline because as cache memories increase in size the performance difference between direct-mapped and set-associative will decrease [4]. Furthermore, a DM cache is always smaller and faster than a SA cache of equivalent capacity, due to the extra circuitry required to implement the associativity. From our limited trials it is difficult to validate such a trend in performance. An 8K, DM cache had 9.6% more hits than a 2K, DM cache, whereas the 2-way, SA cache had 12% more and the 4-way had 13% more. These differences are similar to the differences observed for 2K caches. Of course, common sense dictates that as the cache size approaches the size of the working set the differences will diminish. While this may occur soon for board level cache memories, the author suspects that on-chip cache will continue to benefit from the use of associativity, due to size limitations. Doubling the associativity and halving the number of sets requires less area than doubling the cache capacity.

7 Summary

The results indicate that a set-associative cache can experience a significant number of defects and still exceed the performance of a direct-mapped cache of equivalent capacity. Secondly, although the LRU replacement strategy performed better than FIFO or token-based replacement, the modest improvement, particularly at lower associativities and large cache sizes, may not warrant the increase in control logic.

The fundamental question is: "Should associativity be used to increase manufacturing yields instead of spare rows and columns?" To answer this, one needs to develop a cost function capable of reflecting the impact of manufacturing throughput, circuit characteristics (power, size, speed), and cache performance as measured by miss ratio. Several observations may be made:

- If the cache access time is critical and the application can not tolerate the additional delay imposed by associativity then spare rows and columns are the only alternative for increasing yields.
- If the chosen technology has matured to the point where manufacturing throughput is not severely affected by the time needed to repair devices, then spare rows and columns are probably the logical selection. A repaired part will be guaranteed to have a full set of defect-free blocks and will have known performance characteristics.
- If, on the other hand, manufacturing throughput is poor, due either to low yields or lengthy repair times, then using associativity may be viable alternative to using spare rows and columns. By doubling the associativity and halving the number of sets, cache performance can be improved even in the presence of defective blocks. Repair time will be minimal and simply involve marking defective blocks as unusable. Research is being considered to evaluate the area overhead associated with enhancing the replacement algorithms to avoid defective blocks.

Perhaps the biggest deterrent to using this approach may be the difficulty in marketing such a device. Customers expect devices to be 100% defect-free and might be unwilling to order parts which are guaranteed to have "a maximum defect-level," particularly as two devices with the same defect-level will not perform identically on the same workload.

References

[1] Harald Bergh et al, " A fault-tolerant associative memory with high-speed operation, " *IEEE Journal of Solid-State Circuits*, pages 912 – 919, August 1990.

[2] Ramsey W. Haddad et al, " Increased throughput for the testing and repair of RAM's with redundancy, " *IEEE Transactions on Computers*, pages 154 – 166, February 1991.

[3] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[4] Mark D. Hill, " A case for direct-mapped caches, " *IEEE Computer*, pages 25 – 40, December 1988.

[5] Will R. Moore, " A review of fault-tolerant techiniques for the enhancement of integrated circuit yield, " *Proceedings of the IEEE*, pages 684 – 698, May 1986.

[6] R. Negrini et al, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, chapter 4, The MIT Press, 1989.

[7] David A. Patterson et al, " Architecture of a VLSI instruction cache for a RISC, " In *Proceedings of the Tenth Annual Symposium on Computer Architecure*, pages 108 – 116, June 1983.

[8] T. Shinoda et al, " A 1Mb ROM with on-chip EEC for yield enhancement," In *IEEE International Solid State Circuits Conference*, pages 158 – 159, February 1982.

[9] Alan Jay Smith, " Cache memories, " *ACM Computing Surveys*, pages 473 – 530, September 1982.

[10] Robert T. Smith et al, " Laser programmable reduncancy and yield improvement in a 64K DRAM," *IEEE CJounral of Solid-State Circuits*, pages 506 – 514, October 1981.

[11] Gurindar S. Sohi, " Cache memory organization to enhance the yield of high-performance VLSI processors, " *IEEE Transactions on Computers*, pages 484 – 492, April 1989.