Performance Recovery in Direct - Mapped Faulty Caches via the Use of a Very Small Fully Associative Spare Cache

H. T. Vergos & D. Nikolos

Computer Technology Institute, Kolokotroni 3 , Patras, Greece
&
Computer Engineering and Informatics Department,
University of Patras, 26500 Rio, Patras, Greece.

Abstract

Single chip VLSI processors use on-chip cache memories to satisfy the memory bandwidth demands of CPU. By tolerating cache defects without a noticeable performance degradation, the yield of VLSI processors can be enhanced considerably.

In this paper we investigate how much of the lost hit ratio due to faulty block disabling in direct-mapped caches can be recovered by the incorporation of a very small fully associative spare cache. The recovery percentage that can be achieved as a function of the primary cache's parameters (cache size, block size), the number of faulty blocks and the size of the spare cache is derived by trace driven simulation. The results show that when the number of the faulty blocks is small the use of a spare cache with only one block offers a hit ratio recovery of more than 70%, which increases further with cache size. A spare cache with two blocks is justified only in the case of a large number of faulty blocks.

1. Introduction

Single-chip VLSI processors use on-chip cache memory to provide adequate memory bandwidth and reduced memory latency for the CPU [4 - 10]. The area devoted to some on-chip caches is already a large fraction of the chip area and is expected to be larger in the near future. For example, in the MIPS-X processor [6] more than half of the chip area is devoted to an on-chip instruction cache.

Since in the near future a large fraction of the chip area will be devoted to on-chip caches, we expect that in a large fraction of VLSI processor chips the manufacturing defects will be present in the cache memory portion of the chip. Application of yield improvement models [11] suggests that, by tolerating cache defects without a substantial performance degradation the yield of VLSI processors can be enhanced considerably.

A technique for tolerating defects is the use of redundancy [25]. The use of redundancy to tolerate defects in cache memories was discussed in [1, 2]. Redundancy can have the form of spare cache blocks where if a block is defective it can, after the production testing, be switched out and substituted by a spare block using electrical or laser fuses. Instead of spare cache blocks, spare word lines and/or bit lines may exist that are selected instead of faulty ones. The overhead of these techniques includes the chip area for the spare blocks or word lines/bit lines and logic needed to implement the reconfiguration. Another form of redundancy is the use of extra bits per word to store an error correcting code [26]. Sohi [1] investigated the application of a Single Error Correcting and Double Error Detecting (SEC-DED) Hamming code in an on-chip cache memory and found out that it degrades the overall memory access time significantly. Therefore the classical application of a SEC-DED code in the on-chip cache for yield enhancement does not seem to be an attractive option for high-performance VLSI processors. In [27] it was shown that the defects in the tag store of a cache memory may cause significantly more serious consequences on the integrity and performance of the system than similar defects in the data store of the cache. To this reason a new way of the SEC-DED code exploitation well suited to cache tag memories was proposed. During fault free operation this technique does not add any delay on the critical path of the cache, while in the case of a single error the delay is so small that the cache access time is increased by at most one CPU cycle. Unfortunately, this technique is effective only in the case that the defects cause single errors per word as for example in the case of a bit line defect.

Another technique to tolerate defects in cache memories is the disabling of the faulty cache blocks that was investigated in [1, 2]. It has been shown in [1, 2] that the

0-8186-7059-2/95 $04.00 © 1995 IEEE

mean relative miss ratio increase due to disabling a few defective blocks decreases with increasing cache size and is negligible unless a set is completely disabled. Therefore, the effectiveness of the method of disabling the faulty blocks depends on the number of faulty blocks and whether a set is completely disabled. Unfortunately, as we will show in the following, a large number of faulty blocks may exist. In the case of random spot defects [20] we have to consider a very small number of defects because chips with a large number of defects will usually suffer defects in other critical resources. However, we can easily see that even a very small number of defects in the tag store of a cache memory can affect a large number of tags, leading to disabling of a large number of cache blocks. Also, taking into account the fact that the area devoted to the cache may be a large portion of the chip (50% or more) and the clustering of manufacturing defects [20 - 24] we conclude that it is possible in VLSI processor chips a large number of defects to appear in the caches while all the critical resources of the chips are defect-free. Besides the above, in direct-mapped caches, a set contains just one block, thus disabling a faulty block means the disabling of a set. Direct-mapped caches offer smaller average access time than set-associative ones for sufficiently large sizes [3]. Thus as the size of the on-chip caches increases, the use of direct-mapped caches is favored [12].

Block disabling does not increase the cache access time and permits all non-faulty blocks to be used. However it increases both the mean and the variability of cache's miss ratio [2]. The main performance metric of a memory hierarchy is the average memory access time, T_{av}. For a non-faulty cache with a miss ratio m,

$$T_{av} = T_{cache} + m\, T_{memory},$$

where T_{cache} is the access time of the cache and T_{memory} the average access time of the subsequent memory hierarchy levels. In a faulty cache where faulty block disabling has occurred, the average memory access time will be :

$$T'_{av} = T_{cache} + m'\, T_{memory},$$

where m′ is the resulting cache miss ratio after faulty block disabling. That is,

$$m' = m + \delta,$$

where δ is the increment of miss ratio due to the disabled blocks. Combining the above relations we get :

$$T'_{av} - T_{av} = \delta\, T_{memory}.$$

Since T_{memory} is large the above relation implies that faulty block disabling increases the average memory access time significantly, even when δ is very small. The value of δ depends heavily on the cache associativity, block size and the number of faulty blocks [1, 2]. For direct-mapped caches even a very small number of disabled blocks can result in a substantial value of δ.

Jouppi [12] has proposed the use of a small fully associative (called victim) cache for enhancing the hit ratio of non-faulty direct-mapped caches. The function of the victim cache is to buffer the blocks that leave the main cache due to replacement. The reason of incorporating the victim cache in [12] is for direct-mapped cache performance enhancement and not for performance recovery, since faulty conditions are not considered.

In this work we investigate the subject of recovering the performance of a direct-mapped cache lost due to some disabled faulty blocks by incorporating a very small fully associative spare cache -hereafter called spare cache- *that only serves as spare for the disabled faulty blocks.* More specifically the following problems are examined :

1. How does the performance recovered relate to the primary cache's size, block size and the percentage of faulty blocks ?
2. What is the size of the spare cache required and how does it relate to the above mentioned factors ?

The use of the spare cache described later in Section 2 does not add any delay to the operation of the cache. The size of the spare cache will be measured by the number of blocks that it contains.

The results presented and analyzed in this paper are based on extensive trace driven simulation that is regarded the best way to determine a cache hit ratio [13]. Results for the ATUM traces [14], indicate that hit ratio recovery of more than 70% is feasible by a spare cache with only one block when the number of faulty blocks is small and increases with the size of the cache. The results of trace driven simulation also indicate that the incorporation of a spare cache with two blocks is justifiable only for large number of faulty blocks.

The paper is organized as follows. Section 2 presents the function of the spare cache. Section 3 discusses the simulation methodology, the traces used and presents the simulation results along with an investigation of them. Section 4 concludes our discussion.

2. The suggested cache operation

In this section we will describe the use of the spare cache along with the primary on-chip cache for hit ratio recovery.

Figure 1 presents the block diagram of the primary on-chip cache along with the spare cache. As it is obvious from this figure, the only changes to the original configuration reside in the cache control logic. Suppose that the primary on-chip cache has suffered a number of defects and that during production testing the faults caused by those defects have been discovered. One approach to implement the disabling of cache blocks is to use a second valid (availability) bit [15]. The blocks that contain one or more faults will be disabled by resetting their availability bit. All the non-faulty blocks will have the corresponding bit set and can be used during cache operation.

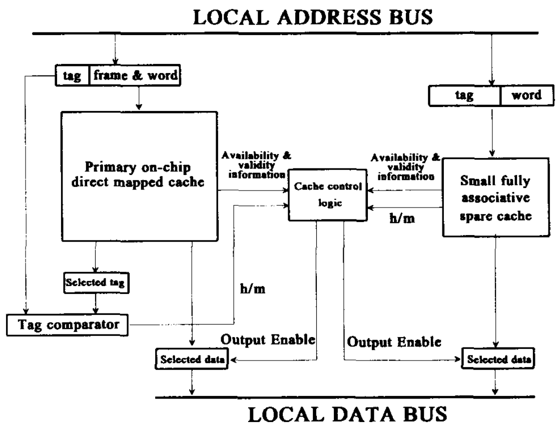

Figure 1. Block diagram of the primary and the spare on-chip caches.

All accesses (either reading or writing) are in parallel propagated to both the primary and the spare caches. Any access that does not refer to a faulty block (the cache controller logic can discover if that is the case by examining the availability bit) is serviced by the primary cache as in the case that no spare existed. This modification does not increase the cache's access time.

Suppose now that an access to a faulty block happens. The cache controller logic will take notice of it due to the status of the availability bit. Off-chip memory access would be required in this case when only block disabling was performed. Such an access may not have to take place even when using a very small spare cache. The controller logic in this case takes account of the hit or miss signal generated by the spare cache along with the validity and availability information. In a hit case the control logic only needs to activate the tristate output data buffer of the spare cache and long off-chip memory access is avoided. LRU updating for the spare cache can be done in parallel with sending the required data to the microprocessor. The spare cache is a very small fully associative cache consisting from (as will be shown in the next section) only one or two blocks thus the hit/miss signal (h/m signal in figure 1) of it will be asserted much faster than the hit/miss signal of the primary on-chip cache [16].

When a disabled primary cache faulty block is referenced and the access of the spare cache results in a miss, the control logic has all the time required to discover the LRU block, update the LRU contents and replace the LRU block data with those from the slower off-chip memory hierarchy when they become available on the local data bus. No overhead delay is imposed to the non-faulty cache's miss penalty.

We must notice that the description given above can equally well be applied to both real and virtual addressed caches. Since the primary cache's organization remains undisturbed the above description is also free from any specific implementation details.

3. Simulation and results

The best way of determining the miss ratio of a certain cache configuration is through trace driven simulation [13]. In our simulations we used the ATUM traces because they include both operating system references and multiprogramming effects. Moreover, the way that those traces were gathered introduces fewer distortions of the address trace [14]. Table II in [2] lists the features of each individual trace used. We present results only for the combined trace, denoted by *all* in [2] due to the large number of traces. For this specific trace we combined the individual ATUM traces by concatenating them and inserting cache flushes before a new trace was appended. The combined trace is about 8 million references long. Since the individual traces are up to 400000 references we simulate cache sizes up to 32 Kbytes. Larger cache simulation would be impossible without inserting much error [2, 17].

Trace driven simulation is known to be a time-consuming process [13] and several techniques have in the past been proposed to shorten the required simulation time (see for example [2, 17, 18]). These techniques though can not be applied in our case. For caches with large numbers of blocks and/or enough large number of faulty blocks, all the possible faulty blocks combinations can not be taken into account in the simulation because then the required time is prohibitively large. In such cases simulation for a subset of all the possible faulty blocks combinations should be performed.

Therefore, we have to determine how many faulty blocks combinations should be examined so that the miss ratio measurement to be close enough to the values measured by simulating all possible faulty blocks combinations (exhaustive simulation). To this end, for several cache sizes, we examined 10, 100 and 1000 faulty blocks combinations and compared both the accuracy of the results and the required simulation time against the results and the required time of the exhaustive simulation. We found the one hundred limit to be a very good compromise in producing the average miss ratios but improper for the minimum and maximum values. The extra simulation time required by the one thousand limit was not justified, since the average miss ratios were extremely close to that of the one hundred and both minimum and maximum values of miss ratio although closer, still far enough from those of the exhaustive simulation. The one hundred limit was therefore chosen. Exhaustive simulation was used when affordable.

Figure 2 shows the percentage of the lost hit ratio due to block disabling, recovered by using a spare cache with only one block, for block sizes 8, 16 and 32 bytes. Specifically the

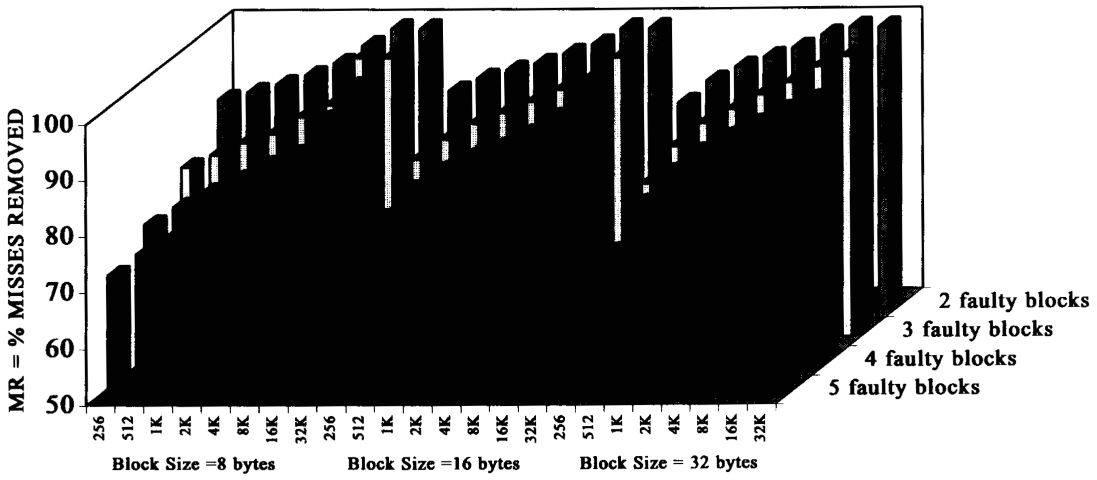

Figure 2. Percentage of the lost hit ratio due to block disabling, recovered by one spare block for 2 to 5 faulty blocks, for caches with block sizes 8, 16 and 32 bytes.

percentage of the Misses Removed (MR) is equal to :

$$MR = (m_1 - m_2) / m_1 * 100$$

for $m_1 = m' - m$,

where m are the average misses of the non-faulty primary cache.

m' are the average misses of the primary cache with the faulty blocks disabled.

and $m_2 = m'' - m$,

where m'' are the average misses of the faulty primary cache supported by a spare cache.

The number of faulty blocks is varied from two to five (results for one faulty block have no meaning since recovery of 100% would be the outcome in any case).

In all cases the recovery percentage exceeds 64 %. When the primary cache size is greater than 1 Kbytes the recovery percentage is well over 75 %. (This is the most common case in today's commercial microprocessor chips). Three more observations must be made from this figure :

a. The percentage of the misses removed (MR) increases as the number of faulty blocks decreases for the same primary cache and block sizes. This is because less faulty blocks means less possible candidates for the single spare block and hence less conflict misses.

b. MR for the same number of faulty blocks and block size increases as the cache size increases. This is due to the fact that a larger primary cache has a bigger number of addressable blocks than any smaller with the same block size, and so references to specific blocks are more rare. For example, when the block size is 32 bytes, a 256 bytes cache, only has 8 distinct blocks that may be referenced, while a 4 Kbytes cache with the same block size has 128 distinct blocks. If we assume that two faulty blocks exist in any of these caches, the probability of interchanging references to the faulty blocks (that cause conflict misses in the only spare block available) in the first cache is a multiple of the corresponding probability of the second cache.

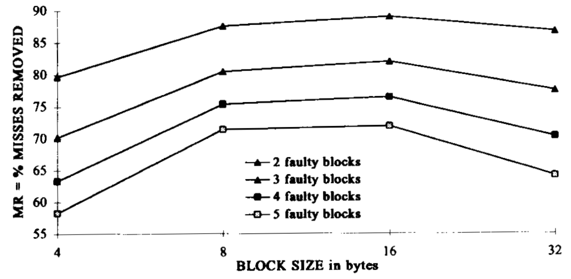

Figure 3. Percentage of the lost hit ratio due to block disabling, recovered by one spare block for 2 to 5 faulty blocks. Cache size = 256 Bytes.

c. The relation of MR to the cache's block size though is more complicated. According to (b) above one would expect that MR will drop when moving to larger block sizes, because then we have a smaller number of blocks and thus increased conflict misses. But it has also been shown in [2], that, for a specific number of faulty blocks, the hit ratio loss is greater for caches with larger block sizes and thus the spare cache will offer a greater recovery percentage. Figures 3, 4 and 5, present MR versus block size, for cache sizes of 256 bytes, 1 and 8 Kbytes respectively and for two to five faulty blocks. We can see that for small block sizes the second of the above mentioned factors dominates and MR is increased when moving from 4 to 8 or 16 bytes block. On the other hand for large block sizes the first factor is the dominant and

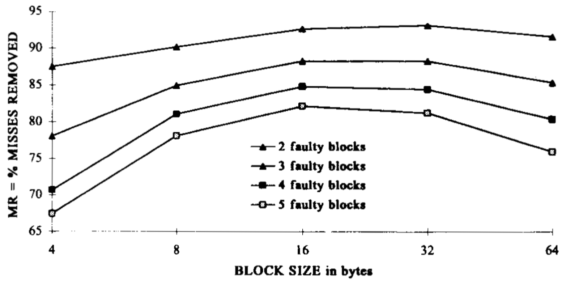

Figure 4. Percentage of the lost hit ratio due to block disabling, recovered by one spare block for 2 to 5 faulty blocks. Cache size = 1K Bytes.

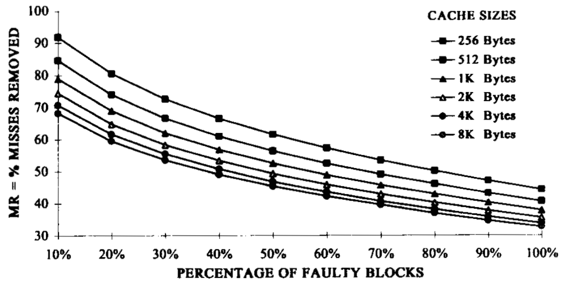

Figure 6. MR vs. the percentage of faulty blocks in caches up to 8K, with block size of 16 bytes.

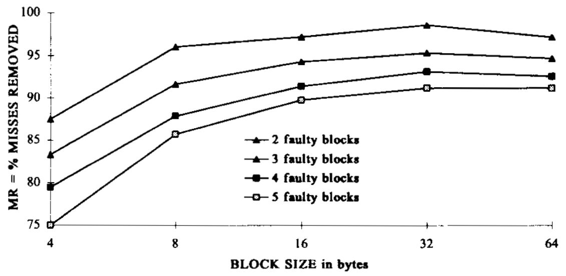

Figure 5. Percentage of the lost hit ratio due to block disabling, recovered by one spare block for 2 to 5 faulty blocks. Cache size = 8K Bytes.

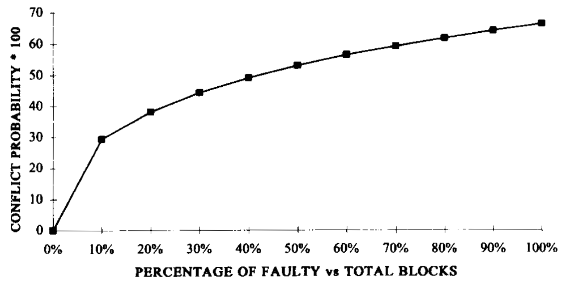

Figure 7. Conflict probability for one spare block vs. the percentage of faulty blocks. Cache size = 4 Kbytes, Block size = 16 bytes.

MR drops when moving from 32 to 64 bytes block. Depending on the cache size, a block size of either 16 or 32 bytes can maximize MR, for constant number of faulty blocks.

We will hereafter discuss how effective is the usage of the spare cache to chips with a large number of cache faulty blocks.

Figure 6 plots the recovery percentage of a single block spare cache for cache sizes up to 8 Kbytes versus the percentage of faulty blocks. For the same percentage of faulty blocks caches with larger size have a greater number of faulty blocks, thus the percentage of misses removed is smaller. Even when 20% of the total blocks are faulty, 60% of the lost hit ratio can be recovered. (A fraction of 20% of the total blocks, in a 8 Kbytes cache, means that more than 102 blocks are unavailable).

The form of the curves presented in figure 6 should have been expected, because any new faulty block that is added to the faulty blocks list will cause many more conflict misses at the spare cache. Suppose for example that two blocks A, B of the primary cache are marked as faulty. Then conflicts at the spare cache occur only by references of the form *A*B* or *B*A*, where * denotes references to any other except the faulty blocks. If another primary cache block, suppose C, is also faulty, then any combination of the form *A*B*, *B*A*, *A*C*, *C*A*, *B*C*, *C*B*, would cause conflict misses at the spare cache.

In figure 7 the conflict probability in the competition for the single spare block is plotted against the fraction of the total blocks that are faulty in a 4 Kbytes cache. This curve was formed by counting the references and the conflict misses at the spare cache. We can observe that when the fraction of faulty blocks is less than 10% there is a great probability (> 0.7) that the spare block caches the required data. When 50% or more of the primary blocks are faulty the spare cache access will probably result in a conflict miss.

Incorporation of a second spare block is examined hereafter. Figure 8 plots the average miss ratio of a 4 Kbytes cache for three cases, namely when only block disabling is performed and when the primary cache is backed up by either one or two fully associative spare blocks, versus the number of faulty blocks. We can see that for less than 16 faulty blocks the miss ratio of the faulty cache with a spare cache of one or two blocks is almost equal to that of the non-faulty cache. The second spare block becomes attractive only when a large number of blocks is faulty. The percentage of misses

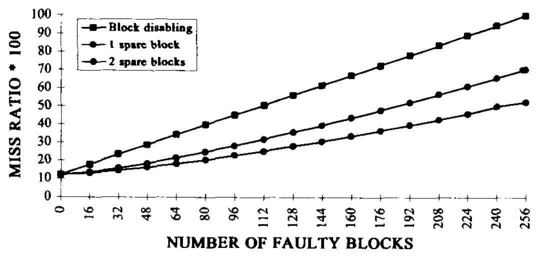

Figure 8. Miss ratio of block disabling and the use of a spare cache of 1 or 2 blocks vs. the number of faulty blocks. Cache size = 4 Kbytes, Block size = 16 bytes.

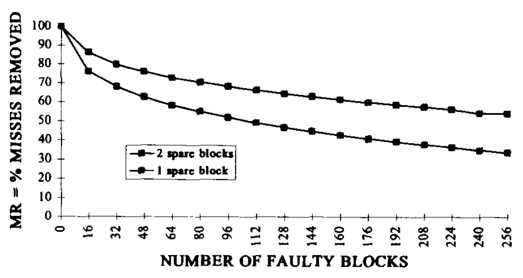

Figure 9. Percentage of lost hit ratio due to block disabling, recovered by a spare cache of 1 or 2 blocks vs. the number of faulty blocks. Cache size = 4 Kbytes, Block size = 16 bytes.

removed is plotted in Figure 9. The returns of the second spare are much smaller than those achieved by the first one and only justified for either :

a. When a great portion of the total blocks are faulty, or
b. When hit ratio sustainment is too important. Multiprocessor caches usually service misses via a multistage interconnect or a bus. When a multistage interconnect is used the miss latency can be large due to contention existence. Bus miss times with low utilization can be small, but delays due to contention among processors can become large and are sensitive to cache miss ratio [19].

Using the average memory access time, T_{av}, as a metric of performance we will now compare block disabling, and the use of spare cache. In the introduction we have shown that block disabling increases T_{av} by :

$$\delta\ T_{memory}.$$

If we denote with r the misses removed due to spare cache incorporation, using the same procedure we can get that T_{av} in this case is increased only by :

$$(1\text{-}r)\ \delta\ T_{memory}.$$

As an example suppose that T_{memory} = 6 cycles (a second level cache is supposed), T_{cache} = 1 cycle. From our simulations, for a 4 Kbytes cache with 16 bytes block and 16 faulty blocks :

$$\delta = 0.0551,$$
$$r = 76.4\%,$$
$$h = 0.8784 \text{ (non-faulty cache's hit ratio).}$$

The overhead imposed on T_{av} in this case by block disabling is approximately 19.11%, whereas in the case of a spare cache with only one block, T_{av} increases by only 4.51%. Using a spare cache with two blocks we have an increase in T_{av} of only 2.68%. In the above example, we assumed that a second level board cache exists. In low cost systems with no board cache, the difference between the increase on T_{av} when only faulty block disabling is used and in the case of a spare cache is increased further.

4. Conclusions

To achieve high performance in single chip VLSI processors, on-chip cache memories are used. As the chip area devoted to on-chip caches increases, we expect that in a large fraction of VLSI processor chips the manufacturing defects will be present in the cache memory portion of the chip. Disabling the cache defective blocks was shown to be an attractive technique for yield enhancement of single chip VLSI processors with on-chip cache. It has been shown in [1, 2] that the mean relative miss ratio increase due to disabling a few defective blocks decreases with increasing cache size and is negligible unless a set is completely disabled. Unfortunately, the number of faulty blocks may be large and in direct-mapped caches, disabling a block means the disabling of a set. Direct-mapped caches offer the fastest access time and for sufficiently large sizes smaller average access time than the set-associative ones [3]. Thus as the size of the on-chip caches increases, the use of direct-mapped caches is favored.

In this paper we have investigated, by extensive trace driven simulation, how much of the lost hit ratio due to faulty block disabling in direct-mapped caches can be recovered by the incorporation of a very small fully associative spare cache. Four conclusions can be drawn from the work presented in this paper :

1. A spare cache with just one block is sufficient for a small number of faulty blocks.
2. A spare cache with two blocks is justified only in the case of a large number of faulty blocks.
3. For constant block size and number of faulty blocks the percentage of the removed misses increases as the cache size increases.
4. For constant number of faulty blocks a block size of either 16 or 32 bytes, depending on the cache size can maximize the misses removed.

The use of a small (with size two or four blocks) fully associative cache which will operate as a victim cache for the non-faulty blocks and as a spare cache for the faulty blocks of the primary cache is under investigation.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their valuable comments.

References

[1] 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.

[2] Pour F. and Hill M. D., "Performance Implications of Tolerating Cache Faults," *IEEE Transactions on Computers*, Vol. 42, no. 4, pp. 257-267, March 1993.

[3] Hill M. D., "A Case for Direct-Mapped Caches," *IEEE Micro*, pp. 25-40, December 1988.

[4] Phillips D., "The Z80000 Microprocessor," *IEEE Micro*, pp. 23-26, December 1985.

[5] Berenbaum A. D., et al., "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.

[6] Horowitz M., et al., "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] Kadota H., et al., "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.

[8] Shoemaker K., "The i486 Microprocessor Integrated Cache and Bus Interface," *In Proc. of the COMPCON '90 IEEE International Conference*, pp. 248-253.

[9] Dopperpubl D. W., et al., "A 200-MHz 64-bit Dual-issue CMOS Microprocessor," *Digital Technical Journal*, Vol. 4, no. 4, pp. 35-50, Special Issue 1992.

[10] Intel, *i860 XP Microprocessor, Multimedia and Supercomputing Microprocessors Data Book*, Intel 1992.

[11] Koren I. and Pradhan D. K., "Modeling the Effect of Redundancy on Yield and Performance of VLSI Systems," *IEEE Transactions on Computers*, Vol. C-36, no. 3, pp. 344-355, March 1987.

[12] Jouppi N. P., "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *ACM Computer Architecture News*, Vol. 18, no. 2, pp. 364-373, June 1990.

[13] Smith A. J., "Cache Memories," *ACM Computing Surveys*, Vol. 14, no. 3, pp. 473-530, September 1982.

[14] Agarwal A., et al., "ATUM : A New Technique for Capturing Address Traces Using Microcode," *In Proc. of the 13th Annual Symposium on Computer Architecture*, pp. 119-127, June 1986.

[15] Patterson D. A., et al., "Architecture for a VLSI Instruction Cache for a RISC," *In Proc. of the 10th Annual Symposium on Computer Architecture*, pp. 108-116, June 1983.

[16] Wilton S. J. E. and Jouppi N. P., "An Enhanced Access and Cycle Time Model for On-Chip Caches," *Technical Report 93/5*, DEC Western Research Laboratory.

[17] Stone H. S., *High-Performance Computer Architecture*, Addison-Wesley, October 1987.

[18] Mattson R. L., et al., "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, 9, 2, pp. 78-117, 1970.

[19] Goodman J. R., "Using Cache Memory to Reduce Processor-Memory Traffic," *In Proc. of the 10th Annual Symposium on Computer Architecture*, pp. 124-131, June 1983.

[20] Stapper C. H., Armstrong F. M., Saji K., "Integrated Circuit Yield Statistics," *Proceedings of the IEEE*, Vol. 71, no. 4, pp. 453-470, April 1983.

[21] Stapper C. H., "Large-Area Fault Clusters and Fault Tolerance in VLSI Circuits : A Review," *IBM Journal of Research and Development*, Vol. 33, no. 2, pp. 162-173, March 1989.

[22] Stapper C. H., "Simulation of Spatial Fault Distributions for Integrated Circuit Yield Estimations," *IEEE Transactions on CAD*, Vol. 8, no. 12, pp. 1314-1318, December 1989.

[23] Stapper C. H., "Statistics Associated with Spatial Fault Simulation Used for Evaluating Integrated Circuit Yield Enhancement," *IEEE Transactions on CAD*, Vol. 10, no. 3, pp. 399-406, March 1991.

[24] Blough D. M., "On the Reconfiguration of Memory Arrays Containing Clustered Faults," *In Proc. of the 21st International Symposium on Fault-Tolerant Computing*, pp. 444-451, June 1991.

[25] Moore W. R., "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," *Proceedings of the IEEE*, Vol. 74, no. 4, pp. 684-698, May 1986.

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

[27] Vergos H. T. and Nikolos D., "Efficient Fault Tolerant CPU Cache Memory Design," *to appear in Microprocessing and Microprogramming - The Euromicro Journal.*