Re-evaluation of Fault Tolerant Cache Schemes

Huan-yu Tu and Sarah Tasneem
Eastern Connecticut State University
Willimantic, CT 06226
{tuh, tasneems}@easternct.edu

ABSTRACT – *In general, fault tolerant cache schemes can be classified into 3 different categories, namely, cache line disabling, replacement with spare block, and decoder reconfiguration without spare blocks. This paper re-examines each of those fault tolerant techniques with a fixed typical size and organization of L1 cache, through extended simulation using SPEC2000 benchmark on individual techniques. The design and characteristics of each technique are summarized with a view to evaluate the scheme. We then present our simulation results and comparative study of the three different methods.*

***Keywords*:** fault tolerant, cache, disabling, spare, PADded

1. INTRODUCTION

High-performance VLSI processors results increasing demand of memory bandwidth which the memory technology never be able to satisfy [2]. Thus, extensive use of on-chip cache memories became essential to sustain memory bandwidth demand of the CPU. Meanwhile, advances in semiconductor technology and continuous down scaling of feature size creates extra-space for additional functionality on a single chip. The most popular way to make use of this extra-space is integrating bigger size of cache so that a microprocessor is able to gain higher performance. We can observe that the area occupied by cache in current processor already exceeded 50% of total area of CPU die. However, an increase in the circuit density is closely coupled with an increase in probability of defect. Furthermore, the increased defects can mostly be in the on-chip cache area since the area occupied by cache grows larger and larger. Consequently, the defect level of the cache has a significant impact on the defect level of overall CPU. Therefore, the first fault tolerant cache design was proposed in [10] for the purpose of enhancement in yield of micro-processors.

To design fault tolerant cache, we first need to observe that cache is a redundant structure which is employed to enhance the performance of CPU. Thus, the CPU can work correctly without cache-memory. Among many components in microprocessor, redundant structures, such as cache is called *non-critical component* and defects in that structure is called *non-critical defects* [10]. The *non-critical defects* can be easily tolerated by simply disabling *non-critical component* which contains defect. Thus, disabling defective part of caches were investigated in [7] [10] [8]. However, simply disabling the defective part of cache will result degradation of overall performance of CPU. Thus, use of redundancy to tolerate defect in cache memory is studied in the literature. The redundancy techniques that are used for RAM can easily applied to cache. Using a SEC-DED code [12] codes can mask out defective bits in cache as well as main memory. However, a detailed investigation in [10] showed that employing SEC-DED code for on-chip cache is not appropriate due to the delay introduced by SEC-DED hardware. Using redundancy and reconfiguration logic is another method to tolerate faults in cache by providing spare cache blocks [4] [13]. The defective block is switched to spare block by reconfiguration mechanism. The reconfiguration can be done either electrical or laser fuses to permanently replace defective blocks [4]. In [13], instead of permanent replacement, they employed small fully associative cache to dynamically replace the faulty block.

Yet another technique which called *PADded cache* [9] is presented recently. This new technique can sooth the degradation of cache performance without spare cache block. Instead of using explicit spare blocks, the physical or logical neighborhood blocks play a role of spare block.

In this paper, we re-examine and compare three different fault tolerant schemes, namely, *cache line disabling* [10] [8] [7], spare cache [13], and *PADded cache* [9]. In section 2, a brief overview of the organizations of the different schemes as well as summary of previous results are presented. In section 3, we re-evaluate each technique with realistic, unbiased setup for fair comparison. Also, the results of our comparisons of the schemes are reported.

2. FAULT TOLERANT CACHES

2.1 Cache Block Disabling

Although disabling the faulty cache will not affect the correct operation of CPU, disabling the entire cache will significantly degrade computer performance. Thus, one may consider disabling some portion of cache [7]. Also, purging the entire way is wasteful since all the other fault free blocks in the same unit cannot be utilized. Therefore, one solution is to disable faulty byte or word in data array to maximize the utilization of fault free bits. However, most cache implementation fetches data from the main memory by the size of multiple words, instead of by a byte or a word, which is usually the same as block size of cache. Hence, disabling a single block containing fault/defect was investigated in [10] [8], where a single bit is used to indicate the presence of fault in a block. This indicator bit is called the *availability bit* [10], the *purge bit*, or the *second valid bit* [8]. In the present paper, we will call this indicator bit as *faulty-bit*.

The faulty blocks can be identified either by a) manufacturing test to enhance the yield of micro-processors

[8] [10] or b) error detection code to tolerate permanent fault occurred even during normal operation [6] [12]. The *faulty-bit* of a faulty block will be set once the block is identified as faulty. When access to the certain address of cache occurs, the cache control logic makes use of the *faulty-bit* by treating the access as a miss and also excludes the block from cache replacement algorithm. Figure 1 illustrates simplified block diagram of this scheme. The effect of disabling faulty block is first presented in [10] and mathematically extended in [8].

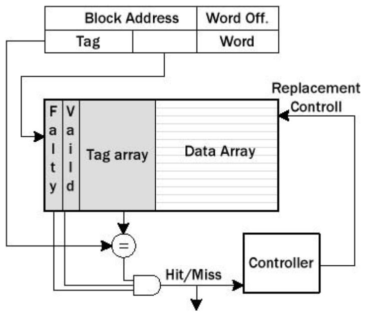

Figure 1. Cache Line Disabling with Faulty Bit

According to their result and analysis, the associativity of cache has big impact on the performance degradation incurred by disabling defective block. In case of direct mapped cache, all the memory blocks that are mapped onto defective block will be excluded from the cache. Thus, one can expect the linear degradation of performance on fraction of faulty block. On the other hand, set associative cache has less degradation ratio to the fraction of faulty block. Suppose *M*-way set associative cache. If one block among *M*-way in a same set is defective, the remaining *(M-1)* healthy blocks are still able to accommodate the corresponding memory blocks that are mapped onto the set. However, the replacement rate on a faulty set will increase due to higher probability of conflict miss by decreased number of ways in that set. A fully associative cache always allows every memory block to be cached in every cache blocks. Therefore, the degradation of cache performance would solely depend on the probability of conflict miss. To overcome this, the idea of using small fully associative spare cache has been evolved which is described in the next sub-section.

2.2 Replacement Using Small Spare Cache

To recover the performance loss due to disabling faulty blocks, a replacement scheme called the *Memory Reliability Enhancement Peripheral* (MREP) is discussed in [4]. The main idea is to provide extra words which can replace any faulty words in memory. A similar method is proposed in [13] in which while MREP replaces a faulty block with the a spare which is dedicated for the specific faulty block, it uses small fully associative cache as a spare for the faulty blocks of direct mapped primary cache. Since any blocks in fully associative spare cache can store data of all possible indexes that is used for its primary cache, it can temporary replace more faulty blocks than the number of spare blocks in *spare cache*.

The organization of spare cache scheme is illustrated in Figure 2 and the use of *spare cache* for direct-mapped cache and its performance recovery are extensively studied in [13]. In order to limit our scope of study on various fault-tolerant cache schemes, it is worth to make some observation from their study. First, they defined a matrix called MR (*Miss-recovery Rate)* to measure the effectiveness of *spare cache* which is equal to:

$$MR = \frac{Misses\ removed\ by\ spare\ cache}{Misses\ caused\ by\ \ faults} \times 100\% \qquad (1)$$

Based on MR, their study on *spare cache* can be summarized as follows:

1. MR decreases as number of faulty block increases.
2. For a constant block size and number of faulty blocks, MR decreases as total cache size decreases.
3. Block size of 16 or 32 maximizes MR.
4. One or two blocks are sufficient for a small number of faulty blocks.

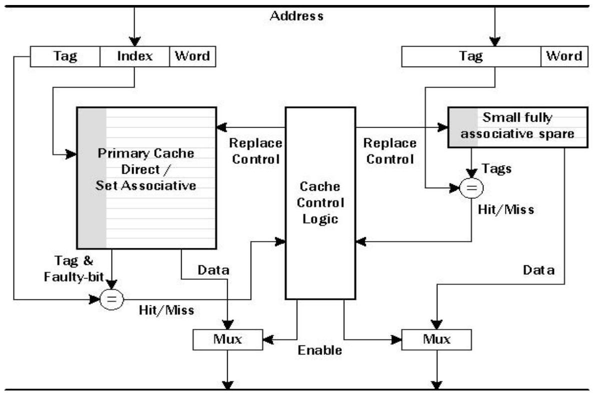

Figure 2. Block Diagram of a Fully Associative Spare Cache

It is obvious to see that the first and second results are due to the increased conflict miss in *spare cache*. As the number of faulty block increases, more blocks will contend to occupy limited number of blocks in *spare cache* resulting more conflict misses in *spare cache*. Also, as the size of the primary cache decreases, the number of addresses which is mapped onto faulty block will increase and eventually will increase the conflict miss in spare cache. The third observation is more complicated. For the block size of less than 16, MR increases as the cache's block size increases because the misses caused by fault is greater for caches with larger block size and thus the spare caches with same block size will cover those misses. However, if block size becomes too big, the addressable space in primary cache will decrease and results into more conflict misses in *spare cache*. The fourth observation is related to the temporal locality [2] on the access to the faulty blocks in primary cache. Due to the presence of temporal locality, the number of spare blocks does not have to be proportional to the number of faulty blocks in primary cache to achieve reasonable MR. Based on the above observations, we re-examined the use of the *spare cache* scheme from various aspects. First, we investigate the effectiveness of *spare cache* for bigger size of primary cache and spare cache, since previous simulation results and conclusion are outdated, regarding the size of caches. It might require more than one or two spare blocks to achieve reasonable MR. Moreover, we examine the effectiveness of spare cache on set-associative cache. This is valuable to study because most of today's micro-processors employ set associative cache to increase hit rate. Furthermore, since cache line disabling on set associative cache already has acceptable degradation for fault tolerance with a very little hardware overhead, it is worthy to investigate the effectiveness of *spare cache* with set-associative primary cache. The new simulation results and analysis on extended study is presented in next section.

2.3 Programmable Decoder (PADded Cache)

As mentioned before, caches have an intrinsic redundancy since the purpose of caches is to improve performance; it should not be relative to the preciseness of operation. Many architectures can work without any cache at the cost of degraded performance. Therefore, adding extra redundancy, using spare blocks, could be inefficient. There is a phenomenon to cut this second redundancy. Because of the spatial and temporal locality of memory references not all of the sets in a cache are hot at the same time. Thus, there must be some cache sets which can substitute the *spare blocks*. A special *Programmable Address Decoder* (PAD) [8] is introduced to exploit this nature.

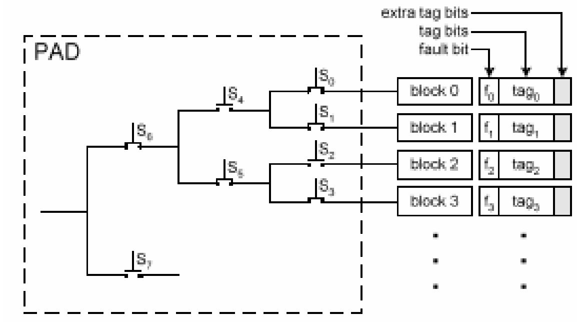

Figure 3. 1-level Programmable Address Decoder (PAD)

When a memory reference happens, a decoder maps it to the appropriate block. A *PAD* is a decoder which has programmable mapping function for the fault tolerance. Once a faulty block is identified, a *PAD* automatically redirects access to that block to a healthy block in the same primary cache. For example, let's consider a *PAD*ded cache, which is equipped with *PAD.* If *PAD*ded cache has *n* cache blocks and one of the blocks is faulty, the cache will work as if it has *n-1* cache blocks. *PAD* re-configures the mapping function so that a healthy block acts as a spare block. The method to find suitable defect-free block is predefined and implemented in hardware. Figure 3 shows one implementation of *PAD* where $S0 = f'0 \cdot a'0 + f1$ and $S1 = f'1 \cdot a0 + f0$ (*a0* is the least significant bit of the index). In the case of where *block 0* is faulty, *S0* will be always on. That is, index for both *block 1* and *block 0* will be directed to *block 1*. However, during the remapping process, one bit of index information will be lost. Therefore, an extra tag bit (shaded boxed in Figure 3) is required to determine whether the contents in block 1 is its own or not. Otherwise, bogus hits can be generated in *block 1*. For instance, suppose that memory address *000* is originally mapped to *block 0* but redirected to *block 1* because *block 0* is faulty. If after the reference to *block 0,* memory address *001* is given, it will be a hit even though the contents come from memory address *000*. More details about multiple-level *PAD*s can be found in [8] and Figure 4 illustrates how the *PAD*s work. The Shaded block in *Set 1* substitutes the faulty block in *Set 0*. The shared block belongs to both of *Set 0* and *Set 1*; the faulty block does not belong to any set. That is, *Set 0* and *Set 1* have three their own blocks and they share one block.

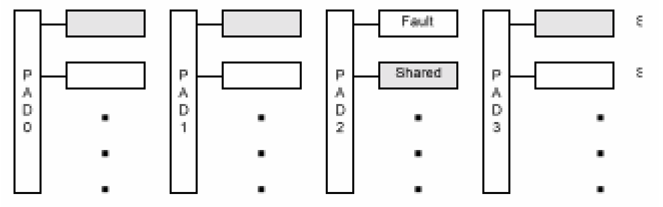

Figure 4. 4-way Associative Cache Employing PAD

To evaluate *PAD*ded caches, they compared this technique with the cache block disabling method for several configurations: cache size (2, 4, 8, 16 and 32KB), block size (8, 16, 32 bytes) and associativity (1, 2 and 4). Many sets of traces were used: ATUM traces, traces from SPEC92 and SPEC95, and the IBS traces. All *PAD*s were assumed that they are programmable for all levels and reverse ordered. The results of their simulation showed that the miss rate of *PAD*ded caches stay relatively flat and increase slowly towards the end. Simulations of caches with different sizes show that when half of the blocks are faulty, the miss rate of a *PAD*ded cache is almost the same as a healthy cache of half the size. The authors claimed that the full capacity of healthy blocks is utilized and the performance decreases with almost minimum possible degradation. Hardware overhead of *PAD*ded cache is also estimated to be 11%, in terms of area [9]. According to their calculation, *PAD*s cost 3%, extra tag bits cost 7%, and the remaining 1% is due to faulty bits.

3. RE-EVALUATION AND COMPARISONS

In this section, we first discuss our simulation set up to reevaluate above surveyed techniques and present the results. Results are divided into two sub-sections. Each fault tolerant schemes are re-evaluated separately and their results are given in section 3.1 and the comparisons of different schemes are presented in sub-section 3.2.

We modified the *SimpleScalar* [2] execution-driven architecture simulator to simulate each technique. Five benchmark programs (i.e. *bzip2, gcc, mcf, vortex,* and *parser*) from SPEC2000 benchmark suite were simulated. For each benchmark, 5 million instructions were simulated (total 25 million). Moreover, caches were flushed on every system calls to mimic realistic operating system environment.

We total cache size and block size was fixed to 32KB and 16 byte, respectively, which is a reasonable configuration for most L1 cache of modern days. Although the total size of cache and the size of block can affect the performance of each technique, the manageable amount of result should be used to focus on the comparison of each technique. However, we varied associativity (1, 2 and 4) since the performance of surveyed techniques were significantly sensitive to the associativity. All simulated caches used write-back, allocate-on-write-miss, and LRU replacement algorithm. No pre-fetching or sub-blocking was used. Also, unified instruction/data cache was used since simulation on separated caches showed no significant difference with unified caches' results.

The *random spot defect model* [11] was assumed to inject permanent faults in random locations in the cache. We assume that power up BIST identifies faulty block with 100% coverage. The *fraction of faulty block* (hereafter called *FFB*) was set to range from 0% to 100% for all simulations. The *FFB* lower than 15% was examined more closely for realistic evaluations.

Miss rate is used, to measure effectiveness of individual techniques. The MR (*miss recovery rate*, *equation 1*) is used to compare relative performance of spare-cache and PADded cache to cache block disabling method.

3.1 Re-evaluations

3.1.1 Cache Line Disabling

Cache line disabling is examined first since it is the most primitive way of tolerating fault in cache and will be used as the basis of comparing effectiveness of other techniques. In addition to direct mapped, a 2-way and 4-way set associative cache, and a fully associative cache is also simulated to get the lower bound of degradation for cache block disabling. Total size of cache and block size is fixed to 32KB and 16 bytes, respectively.

Figure 5 depicts the cache miss ratio versus *FFB*. For the direct mapped cache organization, the miss ratio increases almost linearly with *FFB*. Each block in direct mapped cache is mapped to its congruent memory blocks. In other words, a specific address can be mapped to no more than one block. Thus, the memory blocks those are mapped to disabled block can not be cached. All the references to these blocks will be missed in cache, resulting linear degradation.

On the other hand, disabling faulty blocks in fully associative cache shows 0.07 increase only, even with 90% of faulty blocks. As discussed in previous section, blocks in fully associative cache can accommodate every possible memory blocks and the miss rate depends only on the increased replacement rate. Two-way and four-way set associative cache showed their degradation somewhere in between direct mapped cache and fully associative cache.

Our result shows that the fully associative cache can be the best solution for cache line disabling. However, the implementation of fully associative cache is prohibitively large and impractical. Although there is a large probability for cache to be faulty, we believe that considering more than 15% of *FFB* is impractical. Thus, we closely re-examined the degradation of each cache with *FFB* lower than 15% and the results are plotted in Figure 6. While the relative degradation between 2- or 4-way set associative cache and fully associative cache is significant for large number of faulty blocks, there was only up to 0.01 difference in miss rate between 4-way and fully associative cache when FFB is less than 15%. Furthermore, there was only up to 0.015 miss rate difference between 2-way associative cache and fully associative cache with FFS less then 15%.

In summary, the cache block disabling can be more effective when the associativity of cache is larger. However, for reasonable value of FFB and feasibility, 2- or 4-way set associativity is sufficient to get the advantage of associative organization of cache. The results presented for cache block disabling will be used in subsequent subsections as the basis for comparing effectiveness of other schemes.

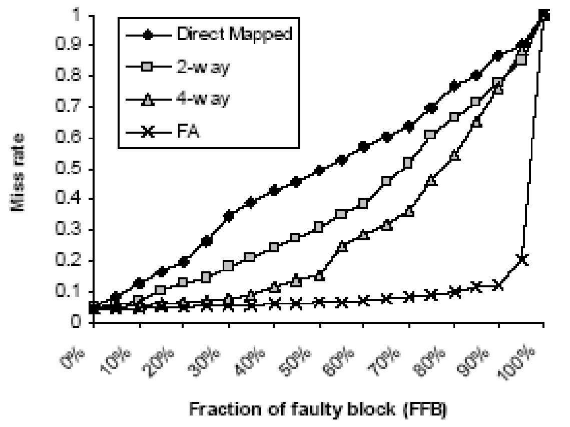

Figure 5. Miss Rate of Cache Line Deletion (32KB)

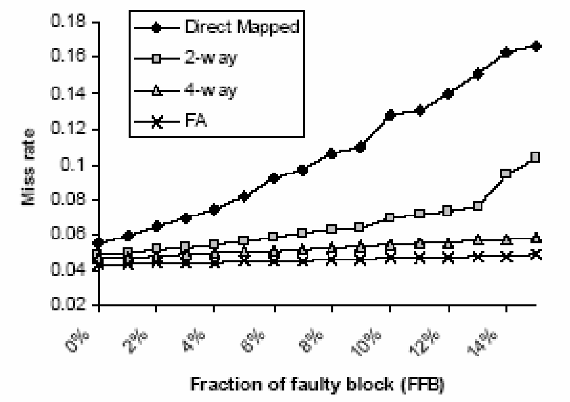

Figure 6. Miss Rate of Cache Line Deletion (32KB)

3.1.2 Spare Cache

Extensive simulation on *spare cache* for the direct mapped primary cache is done in [13] and their results are summarized in previous section. However, the simulation set up is obsolete since the simulated cache size was too small (less than 16KB) and the effectiveness of spare cache in the conjunction with set associative primary cache is not considered. Thus, we extended our simulation for larger cache size (32KB). Moreover, a 2- and 4-way set associative primary cache as well as a direct mapped cache was simulated. First, the effects of varying the spare cache size for three different primary caches are considered. Figures 7, 8, and 9 plots miss rate versus *FFB* for Direct-mapped, 2- and 4-way set associative cache, respectively. For each of the cases a 2, 4, and 8-block sized spare cache was considered. In those figures, DM, 2W, 4W denotes direct-mapped cache, 2-way set associative, and 4-way set associative cache, respectively, "-Del" means cache line disabling or deletion, and "-*x*S" denotes *spare cache* of *x* blocks.

For the entire primary cache configuration, the sensitivity to the size of spare cache is noticeable. The direct mapped cache is more sensitive to spare cache size as opposed to both a 2- and 4-way set associative caches. While there is more than 0.05 miss rate difference between 2-block spare and 8-block spare with direct-mapped primary cache at 15% of *FFB*, 2-way and 4-way set associative caches showed less than 0.005 miss rate difference between 2-block and 8-block spare cache. Especially, the number of blocks in spare cache showed negligible miss rate difference for the 4-way set associative cache.

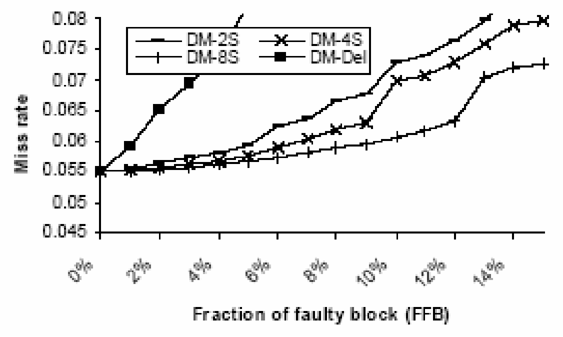

Figure 7. Spare Cache with Direct-Mapped Cache

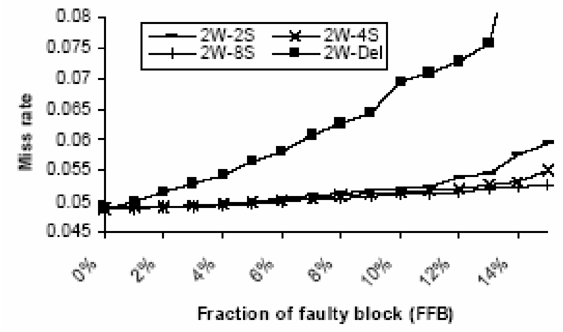

Figure 8. Spare Cache with 2-Way Set Associative Cache

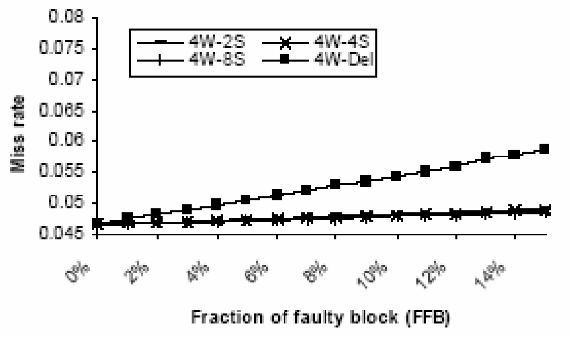

Figure 9. Spare Cache with 4-Way Set Associative Cache

In case of direct-mapped cache, we notice that employing spare cache improves the miss rate significantly as compared to set associative caches. While the miss rate of cache line deletion already exceeds 0.08 with only 5% of FFB, spare cache suppress the miss rate under 0.08 for the *FFB* of up to 14%. Similar but less improvement over cache line disabling can be observed in 2-way set associative cache. On the other hand, there was no significant miss rate improvement (less than 1% difference of miss rate) for *FFB below*15% in 4-way set associative cache. This is because the degradation due to cache line disabling on set-associative cache is already small compared to that of direct-mapped cache. This result brings up new question: Is spare cache effective only for a direct mapped or a small associativity cache? To answer this question, we need to compare MR (*Miss-recovery Rate*) for each of the primary cache organizations. Figure 10 compares the MR for three different primary cache organizations for a fixed spare cache size of 4 blocks. The size of spare cache is fixed to solely compare the effectiveness of employed spare cache on primary caches with different associativity. For small *FFB*, the misses recovered by 4-block spare cache is higher for lower associativity. However, the MR for direct mapped cache quickly drops when *FFB* exceeds 8% resulting less *MR as* compared to 4-way associative caches. The similar drop of *MR* for 2-way set associative was also observed.

The following observation can be made from our results:

- Primary cache with larger associativity is less sensitive to the size of spare cache. This is because the associativity of primary cache can already be able to reduce the misses caused by faulty block access. If misses caused by faulty block access is lower, the contention in the spare cache will be lower, therefore, a very small number (2 or 4-block) spare cache size is enough to tolerate *FFB* of less than 15%.
- The *MR* is higher for smaller associative cache for a lower value of *FFB*, then quickly drops below the *MR* of higher associative primary cache when *FFB* increases further. The reason is that there are more misses caused by fault in smaller associativity cache than larger associativity cache, while the quick drop occurs when the contention in the spare cache causes more replacement in *spare cache*.

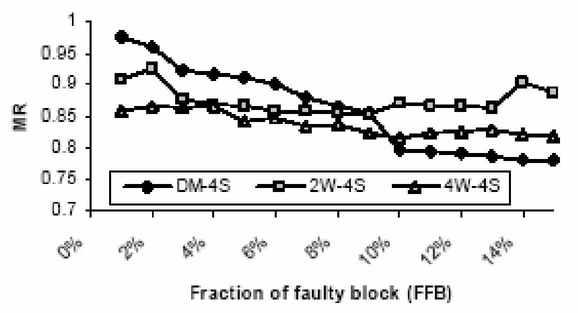

Figure 10. Miss Recovery Rate

3.1.3 PADded Cache

In [9], *PAD*ded caches are simulated extensively on the range of …… % to 100% to emphasize the effectiveness of *PAD*ded cache in case of large *FFB*. Although *PAD*ded cache has this nice property, we believe that considering more than 15% of *FFB* would be impractical. Figure 11 plots miss rate versus *FFB* for Direct-mapped cache from 0% to 15% of *FFB.* Cache block disabling is added to compare the effectiveness. As shown in Figure 11, *PAD*ded caches show very low and flat miss rate for one level of *PAD* is used,. However, there is no significant change of miss rate after level two, which implies employing more than two levels could be extravagant for small range of *FFB*. Therefore, we have used only level two *PAD*ded caches to evaluate *PAD*ded caches, although other levels have been simulated to verify our result.

From observation on Figure 11, it seems that *PAD*ded caches successfully dissolve spare blocks in themselves as they are supposed. However, there is another point. Inserting spare blocks into caches cause the change of locality in the caches. For example, there is a 2-way associative *PAD*ded cache. It has a faulty block, *block A,* so that the references to *block A* are redirected to an adjacent healthy *block B*. Then, it is hard to exploit the spatial locality of the set which contains *block b*. As expected, this phenomenon can be reduced by different ordering of address bits in the *PAD.* The simulation results confirm that reverse order indexed *PAD*s have better performance than normal ordered *PAD*s. Thus, the result presented in this report is based on reverse-ordered index *PAD*.

To assess the effectiveness of *PAD*ded caches, we also simulated 2-way and 4-way associated caches. Figure 12 depicts the variation of MR with FFB for both 2-way and 4-way associated caches. In Figure 12, *PAD*ded caches seems that it takes full advantage of associative caches. Figure 13 shows MR vs. FFB for three different schemes. The results shows that the *PAD*ded cache has less effectiveness when it is used for set-associative cache as opposed to direct mapped cacahe.., *MR* decreases as associativity increases. The conflict of these two figures comes from another locality issue. Suppose there are two sets of cache block: *A* and *B*. Set *A* has a faulty block and its references are re-mapped to set *B*. Even though set *B* is far away from set *A* in address space, this redirection impedes set *B* to exploit the temporal locality. Increasing associativity gives *PAD*ded caches more chances to break the temporal locality. As a result, *PAD*ded caches do not take full advantage of associative cache, which indicates that the performance improvement (see Figure 12) is due to mainly because of primary cache's associativity. *PAD*ded caches have the less portion of contribution to improved performance for the higher associativity. In an extreme case of fully associated cache, *PAD*ded caches have obviously no effect.

For direct-mapped caches, MR stays close to 1 for the entire range of *FFB*, indicating that *PAD*s recovers all most all of misses due to faulty cache blocks. Since there is no other redundancy to accommodate the misses, if some of them are recovered, it must be done by *PAD*s. For associative caches, MR decreases as *FFB* increases. For low *FFB*, associativity contributes more than *PAD*s. However, as *FFB* increases, *PAD*s start to surpass associativity since *PAD*s have more candidates for faulty blocks than associativity. This result seems not in accordance with that of spare caches. Nonetheless, it is quite in accord. The number of spare blocks is very small compared to the number of total cache blocks. Opposite to spare, *PAD*s have huge amount of spare blocks up to 50% of the total number of cache blocks depend on *PAD* level. That is, the capacity of *PAD*s is much larger than that of the spare the cache method. The following is the summary of the simulation results of the *PAD*ded caches:

- Two-level is enough for *PAD*ded caches from a practical view point. For small percentage of faults, small number of levels is actually utilized. Therefore, surplus *PAD* level is not desirable; high level of *PAD*s is expensive.
- The order of inputs to *PADs* is important to its performance. which affects the spatial locality of caches.

*PAD*ded caches contribute to the performance of direct-mapped caches more than that of associativity caches. Since PADs influence the temporal locality of caches, high associativity caches suffer from second conflict miss due to *PADs*.

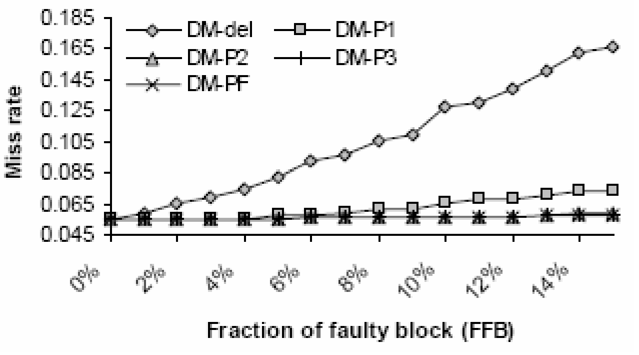

Figure 11. Miss Rates for Different PADded Levels

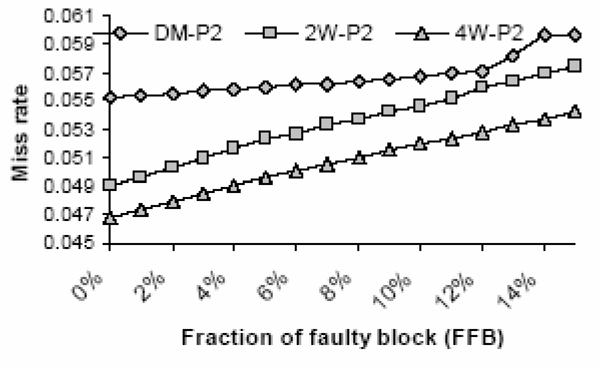

Figure 12. Miss Rates for Different Associativities

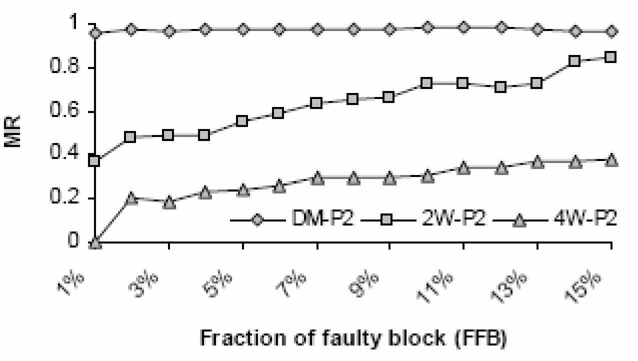

Figure 13. MRs for Different Associativities

3.2 Comparisons

In this sub-section we will compare individually reexamined schemes together in terms of their characteristics, advantages, effectiveness and hardware overhead.

We compare the simulation results of the cache line disabling, 2-level *PAD*ded cache, and 4-block spare cache for each of direct-mapped and 4-way set associative caches. Since 2-level *PAD*ded cache showed close result to its ideal case (i.e. full-level PAD) and there was no significant difference between 4-block and 8-block spare cache, these two configurations of each scheme would be good candidates for the comparison. Direct-mapped and 4-way set associative cache is chosen to clearly compare the characteristics of each scheme when they are applied to the primary cache with different associativities.

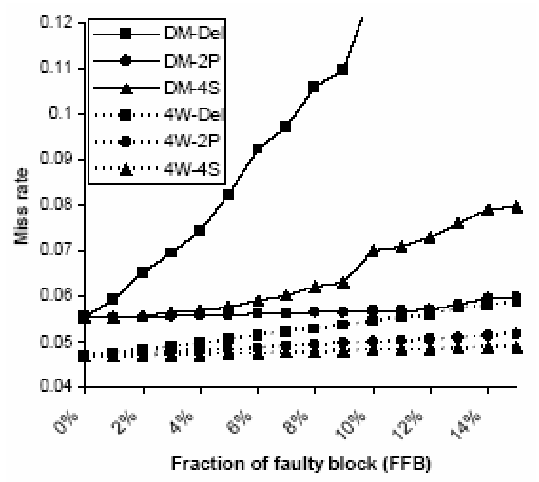

Figure 14. Comparison of Disabling, Spare Cache, and 4-way Set Associative Cache

In Figure 14, miss rate for each configuration is compared for varying FFB. First of all, we can easily observe that all the techniques with direct-mapped primary cache has higher miss rate than those with 4-way set associative cache. Even cache block disabling method has better performance than the best case of direct-mapped primary cache. Thus, one might conclude that using higher set associativity cache would be the best solution when the hardware cost is the main issue. However, if the latency of cache is the primary consideration, there is situation where directmapped cache is preferred [3] as well as fault tolerant features. In this case, *PAD*ded cache seems to be the best solution since the miss rate of 2-level PADded cache has much less degradation for more than 10% of *FFB* in Figure 14. Furthermore, the hardware cost for *PAD*ded cache can be minimized when it is applied for direct-mapped cache. On the other hand, the spare cache scheme can achieve the minimum degradation with 4-way set associative cache. Although there is a slight difference between the 2-level *PAD*ded cache and the 4-block *spare cache*, *PAD*ded cache may not be a good solution for 4-way set associative cache since PADded cache will require 4 separate decoders for each ways of set-associative primary cache.

	Cache Line Disabling	Spare Cache	Padded Cache
Suitable Primary Cache	Large associative cache	Any	Direct mapped or 2 way set associative cache
Hardware Overhead	Lowest	High	Low for Direct-mapped Higher for set associative cache

Table 1. Comparison on Three Different Fault Tolerant Cache Techniques

4. CONCLUSION

As VLSI technology and performance of micro-processor advances, on-chip cache memory becomes essential and continues to grow in size. This trend results more chance of defect in cache area. Consequently, many fault tolerance scheme had been presented in literature.

In this paper, we present the results of extensive simulation study to investigate and compare three different fault tolerant cache schemes**.** Our simulation results for individual technique expose their characteristics and indicate ways to achieve low degradation in system performance. In addition, the result demonstrates that each of fault tolerant techniques has its own advantages and there is no one scheme which is better than the other in all the situations considered as shown in Table 1. However, more thorough investigation on hardware cost of each technique should be done to obtain more precise comparison.

REFERENCE

[1] Austin, T.M., “The SimpleScalar Architectural Research Tool Set”, fttp://www.cs.wisc.edu/~mscalar/simplescalar. html, Rel. 2, Jan. 1998.
[2] Hennessy, J.L., and D.A. Patterson. Computer Architecture. A Quantitative Approach, 2nd edition, Morgan Kaufmann Pub., Inc., San Mateo, CA, 1996.
[3] Hill M.D. “A Case for Direct-Mapped Caches”, IEEE Micro, pp 25-40. December 1988.
[4] Lucente, M.A., C.H. Harris and R.M. Muir, “Memory System Reliability Improvement Through Associative Cache Redundancy,” Proc. IEEE Custom Integrated Circuits Conf., pp. 19.6.1-19.6.4, May 1990.
[5] Nikolos, D. and H.T. Vergos, “On the Yield of VLSI Processors with On-Chip CPU Cache,” Proc. 2nd European Dependable Computing Conference, pp.214-229, Oct. 1996.
[6] O’Leary, B.J., A.J. Sutton, “Dynamic Cache Line Delete,” IBM Tech. Disclosure Bull., Vol. 32, No. 6A, pp. 439, Nov. 1989.
[7] Ooi, Y., M. Kashimura, H. Takeuchi, and E. Kawamura, “Fault-Tolerant Architecture in a Cache Memory Control LSI,” IEEE J. of Solid-State Circuits, Vol. 27, No. 4, pp. 507-514, April 1992.
[8] Pour, A.F. and M.D. Hill, “Performance Implications of Tolerating Cache Faults,” IEEE Trans. Comp.. Vol. 42, No. 3, pp. 257-267, March 1993.
[9] Shirvant, P.P., and E.J. McCluskey, “PADded Cache: A New Fault-Tolerance Technique for Cache Memories,” IEEE VLSI Test Symp., pp. 440-445, Aprril 1999.
[10] Sohi, G, S., “Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors,” IEEE Trans. Comp., Vol. 38, No. 4, pp. 484-492, April 1989.
[11] Stapper C.H., Armstrong F.M. Saji.K. “Integrated circuit yield Statistics” Proc. IEEE vol. 71 pp. 453-470, April. 1983.
[12] Turgeon, P.R., A.R. Stell, M.R. Charlebois, “Two Approaches to Array Fault Tolerance in the IBM Enterprise System/9000 Type 9121 Processor,” IBM J. Res. Develop., Vol. 35, No. 3, pp. 382-389, May 1991.
[13] Vergos, H.T., and D. Nikolos, “Performance Recovery in Direct-Mapped Faulty Caches via the Use of a Very Small Fully Associative Spare Cache,” Poc. Int’l Comp. Performance and Dependability Symp., pp. 326-332, April 1995.