Run Time Management of Faulty Data Caches

Michail Mavropoulos
Dept. of Computer Eng. & Informatics
University of Patras
Patras, Greece
mavropoulo@ceid.upatras.gr

Georgios Keramidas
Dept. of Informatics
Aristotel University
Thessaloniki, Greece
gkeramidas@csd.auth.gr

Dimitris Nikolos
Dept. of Computer Eng. & Informatics
University of Patras
Patras, Greece
nikolosd@ceid.upatras.gr

***Abstract*—As the technology continuous to shrink, power consumption appears to be the main design parameter. Operation on low voltage negatively affects mainly the operation of on-chip memories, resulting in multiple malfunctioning memory cells. As a reaction many cache fault tolerance (CFT) mechanisms have been proposed targeting the mitigation of performance degradation. The challenge is to devise mechanisms that are tailored to the memory access patterns of the executing applications.**

In this work we initially investigate the impact of the granularity of cache line disabling scheme in the first level data caches. Based on our analysis, we propose a run time adaptive mechanism that is able to opt the cache (sub-)block taking into account the diverse memory characteristics of the application. The proposed mechanism is based on the widely used block (sub-block) disabling scheme, and dynamically selects the appropriate sub-block granularity during the execution of the applications. Our evaluation results reveal that the proposed dynamic approach is able to offer significant benefits over a faulty cache design with a monolithic (sub-)block granularity.

Keywords— Cache Fault Tolerance, Re-configurable caches

I. INTRODUCTION

Reducing the supply voltage in today's process technologies introduces significant reliability challenges for on-chip SRAM arrays. This is particularly true as silicon industry moves into the near threshold region characterized by high fault probabilities [7]. On-chip caches are built with minimum sized (to reduce leakage power), thus more prone to failure SRAM cells [3]. Resilience roadmaps pinpoint the vulnerability problem in SRAM cells [16]. As a result, a vast portion of the on-chip memory resources will become unreliable leading to stochastic designs due to increases in static [4] and dynamic [5] variations, wear-out failures [24], and manufacturing defects [3].

Therefore, it becomes critical to investigate new CFT techniques [13][14][17][22]. Obviously, these techniques have to be both lightweight and performance effective, especially when the target caches are close to the core (e.g., L1 caches). A broad category of CFT designs, named after the term graceful degradation [18], has gained much attention by the researchers as well as the industry. The underlying idea is to disable cache portions, such as cache ways, that include malfunctioning memory cells, and apply several schemes to reduce the consequences of the disabled cache portions [2][13][17][22][23]. A detailed analysis about the already proposed techniques is presented in Section III.

A particularly attractive, due to its simplicity, scheme was presented in [1]. The authors introduced the concept of subblock disabling. Instead of relying on complex cache restructuring or block remapping approaches, the cache lines are divided into four parts (called subblocks) and a separate bit (called fault bit) is assigned to each subblock. Sub-block disabling (called SBDIS hereafter) allows keeping data in cache lines even if they have some faulty subblocks. By tracking which subblocks are not faulty, hits in those subblocks can be detected. Obviously SBDIS is a low overhead CFT technique (with less than 0.19% area overheads) [1].

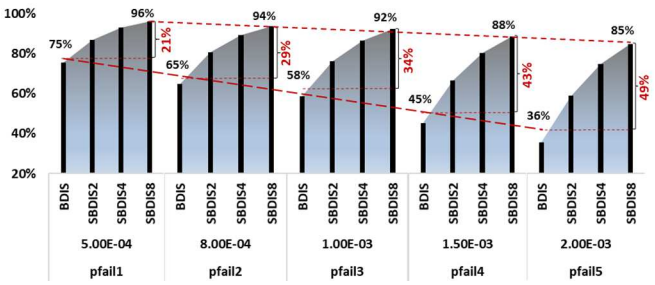

Figure 1: Effective (fault-free) cache size for different percentages of malfunctioning cells.

Figure 1 shows the impact of the granularity of the (S)BDIS scheme to the cache fault-free area (y-axis) for five percentages of failures (pfails) assuming a 32KB, 64-bytes block, 8-way cache. In BDIS, one fault bit is applied to the whole cache frame (1 fault bit per 64B). In SBDIS2/SBDIS4/SBDIS8, one fault bit is assigned to every 32/16/8B sub-block respectively. From Figure 1, it is obvious that moving to smaller block granularities, a larger effective cache capacity is available at the microarchitectural level and this trend is more pronounced as we shift to higher pfails. For example, in the 5e-04 pfail (five malfunctioning cells per 10^4 memory cells), BDIS results in 75% fault-free cache area (36% in 2e-03), while the SBDIS8 scheme manages to increase the sound area to 96% (85% in 2e-03).

The main contributions of this work are:

- We examine the cache behavior for a wide range of pfails and for two benchmark suites by applying the BDIS technique in different block granularities. Our analysis reveals that there is no unique BDIS granularity that performs the best across all pfails, fault maps, and studied benchmarks.

- Based on the above observation we propose a simple mechanism that is able to opt the best performing cache block granularity according to the application memory behavior. The proposed mechanism decides at run-time if there is an opportunity to reduce the number of misses and dynamically adjusts the BDIS granularity. Note that the SBDIS scheme in [1] relies only on four subblocks (statically defined).

- An inherent drawback of the proposed mechanism is that it relies on cache flushes in order to select the appropriate (S)BDIS scheme. To address this, we introduce an additional

prediction mechanism that manages to reduce significantly the number of required cache flushes.

- We evaluate our approach using a wide range of applications (from SPEC2000 and SPEC2006 suites), various cache organizations (32/64KB and 4/8 ways), a plethora of fault maps and five pfails in order to prove the validity of our proposal. Our results indicate that our run-time mechanism is able not only to dynamically select the most suitable block granularity, but it can also seize the opportunity to further optimize the cache performance by taking advantage of the different program phases that appear in the application.

Structure of this paper. Section II assesses the impact of defective cells in faulty data caches and motivates this study. Section III surveys related work and Section IV describes our evaluation framework. Section V presents our two-level, run-time management mechanism. Section VI provides our evaluation results and Section VII summarizes the paper.

II. Motivation

Cache design is a multi-parametric design process involving several parameters e.g., block size, associativity, lookup overheads etc. The centerpiece of the cache parameters is the block size. Small block sizes tend to fetch fewer unused words, but impose significant performance penalties by missing opportunities for spatial prefetching. Larger line sizes minimize tag overhead and effectively prefetch neighboring words (spatial prefetching).

Figure 2 depicts the effect of the block disabling scheme assuming two block granularities and five pfails. Omitting at this point the simulation details, the graph in the top shows the averaged values across all studied benchmarks. The y-axis shows the relative increase in the number of misses over a fault-free cache. The x-axis is divided into five main parts (one for each pfail) and each part contains separate statistics for the BDIS and SBDIS4 (each cache line is divided into four subblocks). In all cases, a 32KB, 8-ways, 64-bytes block DL1 cache is assumed. At the top of each bar, the disabled part of the cache is also shown.

As expected, finer granularities lead to larger effective cache capacities. Surprisingly, in the first four pfails the two schemes seem to perform equally well reporting almost the same number of misses. The difference between the two schemes is less than 5.5% until pfail4, while the SBDIS4 is the clear design choice in high failure rate situations (pfail5) e.g., when operating in the near-threshold region. Therefore, it is obvious that if we want to operate a cache memory in different pfails (e.g., in DVFS-enabled systems), a different subblock granularity must be selected.

While the top graph of Figure 2 pertains to the averaged values, the additional graphs in Figure 2 depict specific cases varying either the studied benchmark (for the same pfail) or the studied fault map (for the same benchmark and pfail). The graph in the middle shows six selected benchmarks corresponding to pfail3 (1e-03).

As we can see, different benchmarks exhibit a different behavior. Note that in pfail3, the difference between the two schemes is 4.2% (averaged across all studied cases). In bzip2/SPEC2000 and cactusADM/SPEC2006, the SBDIS scheme manages to significantly reduce the number of misses compared to BDIS by 53.8% and 32.9%, respectively. On the contrary, applu/SPEC2000 and wrf/SPEC2006 are not able to get advantage of the extra memory offered by SBDIS4. In fact,

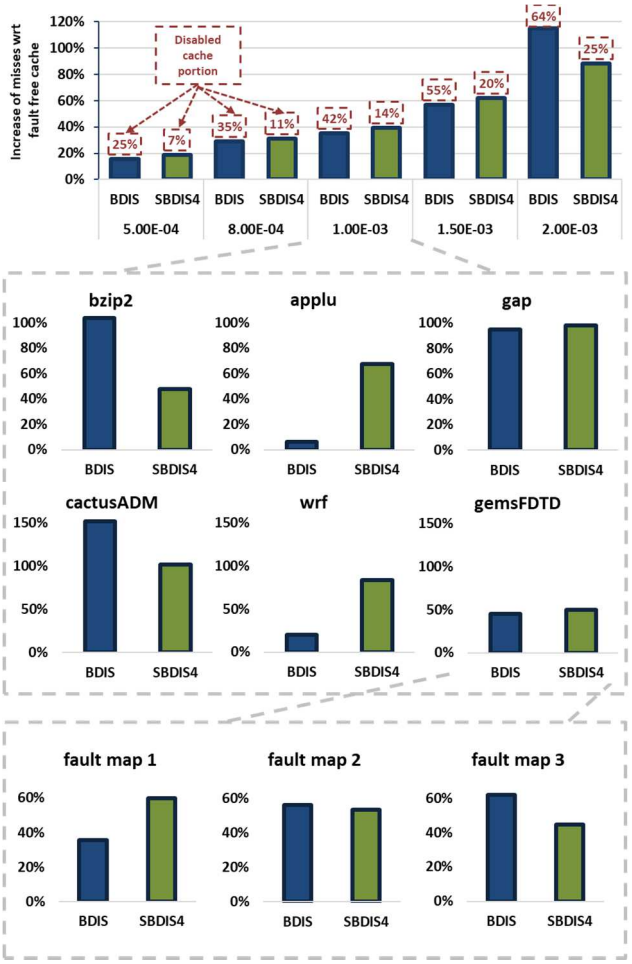

Figure 2: Impact of SBDIS granularity on the miss ratio for the SPEC2000 and SPEC2006 benchmarks. The percentages on top of each bar indicate the fraction of the cache faulty area.

BDIS reports more than 60% less misses in both cases. Finally, the other two benchmarks in the mid group of graphs of Figure 2 (gap/SPEC2000 and gemsFDTD/SPEC2006) show a memory behavior that lies in between the above extreme cases.

Our analysis reveals that the culprit of this non-intuitive case is that fine grain block disabling exhibit a hostile behavior to the spatial locality of the executing applications. In other words, while partially faulty blocks may offer the opportunity to exploit the temporal access patterns of the applications (by providing a limited space to host the requested blocks in the cache), they also limit the possibility to exploit spatial locality, simply because the neighboring memory locations are not cached.

Moreover, the graphs in the bottom part of Figure 2 highlight one more interesting behavior. These graphs correspond to representative cases for the same benchmark (gemsFDTD/SPEC2006) and pfail (1e-03), but for three different fault maps. As we can see, the same trend can be seen here as well: a different block granularity is the best-performing one for different fault maps.

To conclude, fine-grain block disabling schemes are able to substantially increase the available fault-free cache space in defective caches. However, there is no unique BDIS granularity that performs the best across all pfails, fault maps, and benchmarks. In the rest of this paper, we propose a

dynamic mechanism that is designed to dynamically select the best performing granularity according to the application memory behavior (spatial-temporal memory characteristics). The proposed mechanism decides at run-time if there is an opportunity to reduce the number of misses and dynamically adjusts the BDIS granularity.

III. Related Work

In this work we deal with the **cache reliability problem by adapting the block granularity** of faulty data caches. We will briefly overview prior work in both directions.

CFT Techniques. Many CFT techniques rely on redundancy [20] and on an appropriate logic to remap the defective cache parts to functional spare elements [12]. The spare resources are organized as extra placeholders intended to host both the data and tag parts of the defective blocks. However, this approach offers a limited scalability due to the limited number of faulty blocks that can be tolerated [12] (defined by the number of spare blocks). Moreover, when the number of the faults is relatively small (e.g., under nominal operation in DVFS systems) the spare resources are underutilized [14]. The work in [14] addresses the latter issue by combining the functionality of a typical spare cache [20] with the functionality of a victim cache [10]. Another recent work [13], called DARCA, presents an alternative usage of the redundant memory space. Instead of using the redundant elements for hosting the data and tag parts of the defective blocks (data-based redundancy), DARCA devotes the extra storage to hold information (extra tag and status bits) to control how the cache data array stores and disambiguates the cached data (control-based redundancy). Nevertheless, the main disadvantage of the two latter schemes stems from their basic functionality: the additional storage required to hold the required information.

In response to the problems with redundancy, a broad category of CFT designs are based on the graceful degradation concept [18]. In graceful degradation, there are no spares. The idea is to disable cache portions, such as blocks or words that contain defective bits, and reconfigure operational (physically or logically neighborhood) blocks to serve as substitutes. For example, the PADed cache redirects the accesses in faulty cache frames to sound ones by modifying the cache decoding logic [17]. The WDIS [23] scheme combines two consecutive faulty cache blocks into a single sound block. BFIX [23] sacrifices a sound block to repair defects in three other cache blocks. Finally, the FFT-cache [15] describes a multi-bank cache organization to avoid performance degradation when accessing the substituted block or word.

As noted, in [1], a subblock disabling based scheme was proposed. The main advantage of this technique is each low storage overheads i.e., one faulty bit for each subblock (four extra bits in a 64Bytes block). The main disadvantage of the technique in [1] was that a fixed subblock granularity is assumed. As we show in this work, by adapting the cache block granularity significant benefits can be reported. This is due to the unique and changing memory behavior of the different executing applications even for the same pfail and fault map, as well as of the same application for different pfails and/or fault maps.

Cache Line Adaptivity. The cache block defines the fundamental unit of data movement and space allocation in caches. Typically, the cache blocks are uniformly sized to simplify the insertion/removal of blocks and cache refill requests, and to support low complexity tag organization. Unfortunately, this one-size-fits-all approach results in poor cache efficiency. In order to palliate this, the superloading [9] or superblocking [21] techniques were proposed i.e., upon a cache miss, the missing line and surrounding lines are brought into the cache if it deemed profitable.

One of the first approaches for accommodating variable line sizes was in [19]. The authors fetched adjacent blocks into a separate buffer to exploit spatial locality and avoid cache pollution. A study for dynamically allocating data across caches with different line sizes (one optimizing temporal locality and the other one spatial locality) is described in [8].

Our approach also relies on the spatial locality characteristics of the application, but instead of adjusting the number of cache lines that will be fetched in a cache miss, we adjust the number of the active (powered-up) subblocks within a cache frame. To the best of our knowledge, this is the first work that proposes a dynamic mechanism to adapt at run-time the cache block granularity in faulty data caches.

IV. Evaluation Framework

Baseline faulty cache. As noted, our proposal assumes that DL1s are enhanced with the ability of disabling their defective parts at subblock granularity. The baseline faulty cache that we consider in this work is the four-subblock disabling scheme (statically defined) proposed in [1]. Exploring finer granularities is left for future work. As noted, subblock disabling allows keeping data in the cache lines even if they have some faulty subblocks. By tracking which subblocks are faulty, hits in those subblocks can be detected. The implementation of the baseline faulty cache is straightforward [1]. Whenever a hit happens in a faulty subblock, the hit/miss cache logic reports a false hit (the tag comparison logic indicates a match, but the address offset points to a faulty subblock). Then, the cache line is evicted and treated as a normal miss. Finally, it is assumed that faulty caches are already managed by a modified version of the LRU-based replacement policy [1]. More specifically, the cache defect map is exposed to the baseline LRU algorithm. Fully defective cache frames are totally excluded from the (re)placement decisions. However, partially defective cache frames serve as placement candidates, no matter if the just-requested-address (requested portion of the block) corresponds to the sound or the defective physical area of the target frame.

Failures. Cache sections containing malfunctioning cells are disabled, so it is not necessary to consider a specific fault model. Disabling takes place at block or subblock level, therefore we consider that each subblock is accompanied by a separate fault bit. All the fault bits constitute the cache fault map. We consider random distribution of malfunctioning cells (both at tag and data arrays of the cache), while for the valid bits, fault bits, and replacement bits we assume that are not affected by process variation-induced failures e.g., by employing more reliable SRAM cells [11].

Moreover, since our effort is to provide a dynamic and pfail-depended block resizing approach, we study five pfails: 5.00E-04 (pfail1), 8.00E-04 (pfail2), 1.00E-03 (pfail3), 1.50E-03 (pfail4), and 2.00E-03 (pfail5). Finally, we produce random fault maps and run simulations to ensure results are within an error of 5% and a confidence level of $(1 - \alpha) = 0.95$. For each benchmark, the averaged results across all fault maps are presented.

Simulation infrastructure. Our evaluations are based on trace-based simulations. The memory traces have been created using gem5/x86 simulator assuming a 8-stage pipeline, 2-issue, out-of-order processor with a 20-entries instruction window. The extracted memory traces are fed to a custom cache simulator[1]. The reason for using a custom trace driven simulator was to reduce the simulation times. More specifically, for each case (benchmark, cache configuration, and reliability technique) and in order to ensure the target confidence level in our measurements, each simulation was repeated for 150 random fault maps (on average). Due to the large number of required simulations (for 30 SPEC2000 and SPEC20006 benchmarks), performing cycle accurate simulations using gem5 would have led to huge simulation times.

Benchmarks. Two benchmark suites, namely SPEC2000 and SPEC2006, comprise our collection of benchmarks used in this work. For each suite, 15 applications with the highest DL1 miss ratios are presented. In all cases, we simulate 200M instructions using the reference inputs after skipping 1B instructions to avoid unrepresentative startup behavior.

BIST mechanism. We assume that the cache is equipped with a built-in-self-test (BIST) circuitry. The BIST mechanism detects the defective cache parts and updates the fault map either during production testing or under periodic testing in the field of the application in order to detect aging problems too. A description of the BIST mechanism is out of the scope of this work.

V. Organization of the Proposed Mechanism

Having demonstrated the need for adaptive cache line granularities in faulty caches, this section presents our proposal. Much like conventional adaptive computing, our proposal uses a set of parameters to monitor and dynamically to be adapted to changes in application cache behavior. To monitor cache performance, the execution time is divided into fixed windows measured in DL1 accesses. Also, the number of misses is used as the main cache performance metric.

Figure 3 illustrates our framework for dynamically tuning the cache block granularity. The proposed framework is inspired by the approach proposed in [6] where the authors relied on a lossy-compressed representation (signature) of the application working sets in order to detect new program phases. As Figure 3 shows, the time windows[2] are divided into a tuning phase (called sense intervals) in which the available configurations are run for a small interval and the cache misses of each option are recorded by two miss counters. The configuration with the lowest number of misses is selected to be used in the remaining window (execution interval in Figure 3). In the context of this work, our approach is formulated to select among two block granularities: BDIS and SBDIS4. Exploiting more block granularities is left for future work.

However, since our proposal targets to resize the block granularities at run-time, specific cache items must be flushed (invalidated) during transitioning from our cache granularity to another[3]. While this might sound detriment to cache interval performance, it is important to note that the flushing operation is: i) restricted only to the cache blocks that contain one or more malfunctioning SRAM cells and ii) required only during

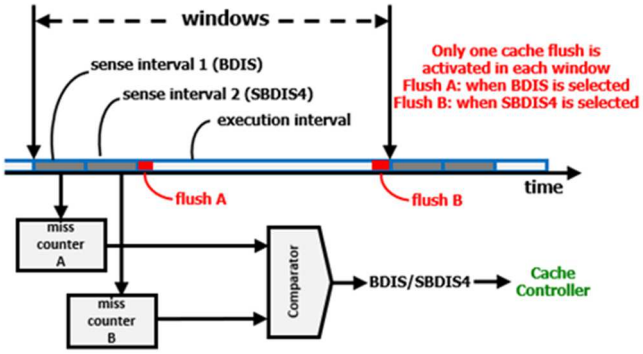

Figure 3: The proposal internal-based block selection scheme

block granularity upsizing (from SBDIS4 to BDIS). Moreover, in the rest of this section, we will present an additional prediction mechanism that is able to further reduce the needed number of cache flushes. In any case, independently of the selected block configuration only one cache flush must be enforced in each window.

The monitoring circuitry is responsible to formulate the sense and execution intervals and accordingly notify the cache controller about the choice of reconfiguration (BDIS or SBDIS4). As noted, for a cache block divided into four subblocks, each subblock (16 bytes) is enhanced by a fault bit (called sfault bit). However, an additional bit for each cache block is required (called bfault bit). The role of the bfault bit is to indicate if the corresponding cache block has one or more defective SRAM cells. This bit can be set either by OR'ing the four sfault bits or it can be part of the cache fault map. Finally, the cache controller is responsible to control the granularity of choice: either by conveying the per-subblock sfault bits (SBDIS4) or the per-block bfault bit (BDIS) in the read/write/invalidate operations of the target cache.

Reducing the cache flushes. To reduce the number of required cache flushes, a simple prediction mechanism is introduced in our design (not showed in Figure 3). If in two subsequent windows the same block configuration is selected, then the sense intervals (thus the cache flushes) are omitted in the consecutive (third) window. In this case, the previous block configuration is used. As we will show in the next section, this simple prediction mechanism is able to reduce the number of the required cache flushes by 30% on average with negligible impact in cache performance. Employing more sophisticated prediction mechanisms is left for future work. In any case, the impact of cache flushes (i.e., extra misses) are fully accounted in our evaluation results.

Time considerations and hardware overheads. With respect to the baseline (subblock-disabled) faulty cache, our proposal does not introduce extra delays in the cache access/lookup path. This is important since our target is to increase the reliability of the latency-sensitive DL1s. The extra components in Figure 3 are: two 32-bit cache-wide counters for measuring the three intervals (measured in cache accesses) + two 32-bit counters for measuring the misses in the two sense intervals + a comparison logic for comparing the misses collected in the previous sense intervals. Obviously, the hardware overheads of these extra components are negligible. In our analysis, we do not account for the

[1] https://github.com/mmavrop/faulty-cache-simulator

[2] Employing variable length windows to better capture the program phases of the applications is left for future work.

[3] During cache flushes the necessary write-backs of the dirty sections are also enforced.

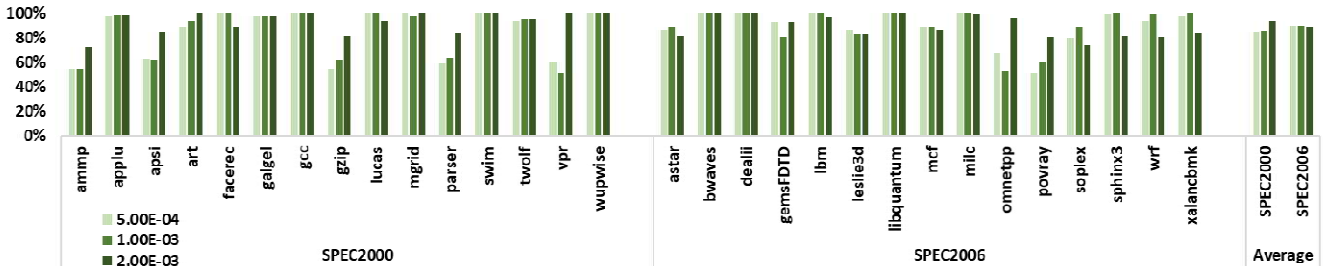

Figure 4: Accuracy in predicting the block granularity with respect to an oracle prediction scheme.

additional circuitry required for the cache flushes, since this feature is typically included in almost all contemporary cache controllers. Finally, the whole monitoring logic runs in parallel to the normal cache operation, while the selection between BDIS and SBDIS4 is activated only once per window and it does not reside in the critical path of the cache read/write/lookup operations.

VI. Experimental Results

This section presents our evaluation results when the proposed dynamic policy is employed in DL1s. Four different cache organizations are considered varying the cache size (32KB, 64KB) and associativity (4-way, 8-way). In all cases, a 64B cache block is used. We experiment with six different windows and eight different sense internal parameters. According to our results, the best performing combination is when the window is configured to 1.8M DL1 accesses while the sense intervals for 32K/4ways, 32K/8ways, 64K/4ways, and 64K/8ways are 12K, 18K, 22.5K, and 22.5K accesses respectively.

The first part of this section quantifies the efficiency of the two separate prediction schemes that employed in our proposal. In the second part, our mechanism is compared against its two main counterparts (BDIS and SBDIS4) in a per-benchmark basis for the 1.0e-03 pfail. The third part contains our overall range of results for all studied pfails, cache configurations, and block disabling granularities.

Prediction accuracy. Working with trace-driven simulations is possible to calculate the accuracy of an oracle predictor by post-processing the memory traces. Figure 4 depicts the accuracy (vertical axis) of our sense-interval based prediction scheme i.e., how many times the configuration selected, by comparing the misses obtained during the two sense intervals, matches the oracle block granularity. As Figure 4 shows in 73 out of 90 cases (30 benchmarks x 3 pfails) the achieved accuracy is more than 80%, but there are 7 cases in which it is less than 60% which means that there is ample room for improvement. Overall, our complexity effective proposal manages to report an 88% accuracy wrt. an oracle predictor.

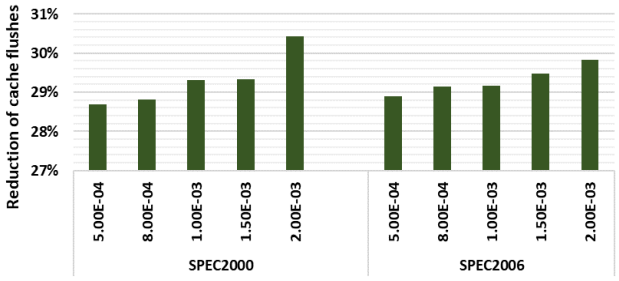

Figure 5: Reduction in cache flushes using the additional prediction mechanism.

Figure 5 quantifies the efficiency of the additional program-phase based predictor in reducing the number of cache flushes. Since the additional predictor relies on the information collected in two previous windows to avoid the cache flush in the subsequent window, the maximum number of the cache flushes that can be skipped is 33.3%. As Figure 5 indicates the reductions in flushes vary from 28.7% (SPEC2000/pfail1) to 30.4% (SPEC2000/pfail5). Employing more sophisticated prediction schemes is left for future work.

Per-benchmark statistics. Figure 6 illustrates our gathered results in a per-benchmark basis for the 1.0e-03 pfail and for the BDIS (dark green bars), SBDIS4 (light green), and the proposed mechanism (red). In all cases, the results (misses) are normalized to the fault-free case. As the graph indicates, the proposed adaptive mechanism not only manages to follow the best performing mechanism (among BDIS and SBDIS4), but also to outperform it by more than 5% in 6 benchmarks. Although the left part of figure 6 presents per benhmark results only for the 32KB/8ways, on the right part we give the averaged results for all four studied cache organizations.

Overall statistics. Figure 7 depicts our full range of results for all studied pfails and cache configurations. The x-axis is divided in five main parts (one of each pfail) and each part contains separate statistics for our proposal ("Dynamic") and four static disabling schemes of different block granularities.

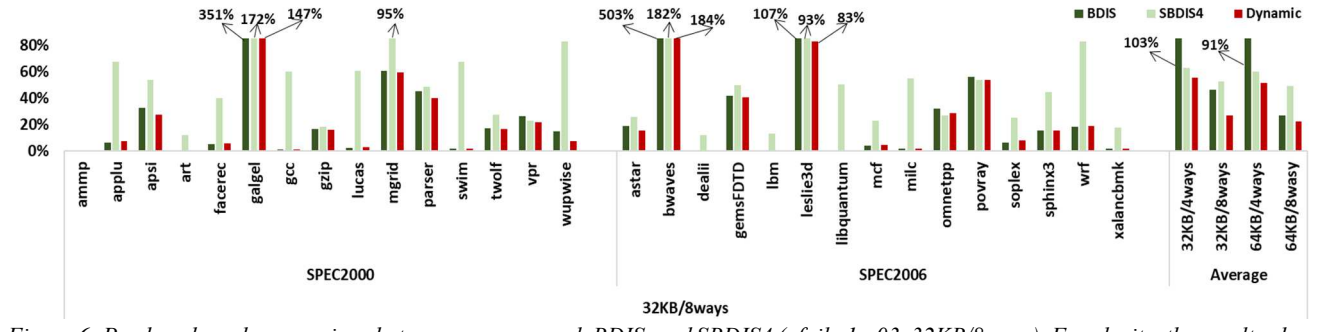

Figure 6: Per-benchmark comparison between our proposal, BDIS, and SBDIS4 (pfail=1e-03, 32KB/8ways). For clarity, the results above 80% are shown in the top of the bars.

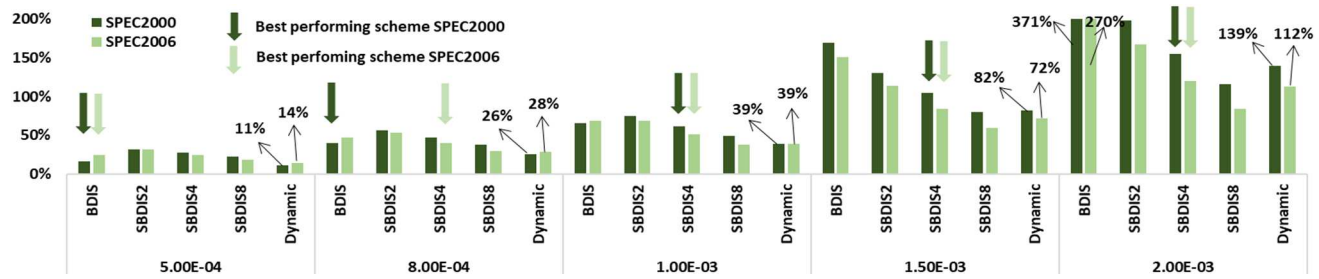

Figure 7: Comparison (avg. results) for 5 pfails between our proposal and 4 (static) block-disabling techniques of different granularities.

The dark/light green bars in Figure 7 correspond to the averaged results of SPEC2000/SPEC2006. Note that in SBDIS8 scheme, each cache block is divided into eight subblocks. Given that our mechanism still selects among the BDIS/SBDIS4 schemes, the SBDIS8 exhibits 57.8% (pfail = 1E-03) more fault-free area with respect to BDIS. As we can see, in the first three pfails, our proposal shows significant miss-saving capabilities over SBDIS8 (37.7%% in pfail1, 19.9% in pfail2, and 9.3% in pfail3). On the contrary, in pfail4 and pfail5, the proposed dynamic scheme fells victim of the extra fault-free cache space of BSDIS8. As expected, the larger difference between the two schemes (dynamic vs. SBDIS8) appear in pfail5 case (23% in SPEC2000 and 28% in SPEC2006). This can be addressed by extending our dynamic mechanism to select among a larger groups of block granularities and by further reducing the number of required cache flushes. Finally, the real strength of our dynamic approach is revealed if we compare our dynamic approach against the best performing mechanism among the BDIS and BDIS4 schemes (vertical arrows in Figure 7). In this case, the average relative gain of our scheme over the best performing scheme (among BDIS and SBDIS4) starts from 8.2% in pfail5 up to 36.3% in pfail1.

VII. Conclusions

A subblock disabling technique when compared to the block disabling increases the effective area of a data cache with malfunctioning cells. However, in this paper we reveal that depending on the percentage of the malfunctioning cells, the fault map and the running application, the use of the block disabling technique can lead to less misses than the use of a subblock disabling technique. To take advantage of this outcome for the benefit of performance, we propose a run time decision making mechanism for the dynamic selection of the block or a subblock disabling technique. The effectiveness of the proposed technique was shown with simulations.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 871738 - CPSoSaware: Cross-layer cognitive optimization tools & methods for the lifecycle support of dependable CPSoS.

References

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González. Low Vccmin Fault-Tolerant Cache with Highly Predictable Performance. Intl. Symp. on Microarchitecture, 2009.

[2] A. Ansari, S. Gupta, S. Feng, and S. Mahlke. Maximizing Spare Utilization by Virtually Reorganizing Faulty Cache Lines. Trans. on Computers, 2011.

[3] S. Borkar. Design Perspectives on 22nm CMOS and Beyond. Intl. Design Automation Conference, 2009.

[4] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. Intl. Design Automation Conference, 2003.

[5] K. Bowman, J. Tschanz, C. Wilkerson, S.L. Lu, T. Karnik, V. De, and S. Borkar. Circuit Techniques for Dynamic Variation Tolerance. Intl. Design Automation Conference, 2009.

[6] A.S. Dhodapkar and J.E. Smith Managing multi-configuration hardware via dynamic working prediction. Intl. Symp. on Computer Architecture, 2002.

[7] S. Gabapathy, J. Kalamatianos, K. Kasprak, and S. Raasch. On Characterizing Near-Threshold SRAM Failures in FinFET Technology. Intl. Design Automation Conference, 2017.

[8] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. Intl. Conf. on Supercomputing, 1995.

[9] T. Johnson, M. Merten, and W. Hwu. Run-time spatial locality detection and optimization. Intl. Symp. on Microarchitecture, 1997.

[10] N. P. Jouppi. Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers. Intl. Symposium on Computer Architecture, 1990.

[11] J.P. Kulkarni, K. Kim, and K. Roy. A 160 mV Robust Schmitt Trigger Based Subthreshold SRAM. Journal of Solid-State Circuits, 2007.

[12] H. Lee, S. Cho, and B.R. Childers. DEFCAM: A Design and Evaluation Framework for Defect-Tolerant Cache Memories. Trans. on Architecture and Code Optimization, 2011.

[13] M. Mavropoulos, G. Keramidas, and D. Nikolos. A Defect-Aware Reconfigurable Cache Architecture for Low-Vccmin DVFS-Enabled Systems. Intl. Conf. in Design, Automation, and Test in Europe, 2015.

[14] M. Mavropoulos, G. Keramidas, G. Adamopoulos and D. Nikolos. Reconfigurable Self Adaptive Fault Tolerant Cache Memory for DVS Enabled Systems. Intl. Symp. of Great Lakes on VLSI, 2015.

[15] A.B. Mofrad, H. Homayoun, N. Dutt. FFT-cache: A Flexible Fault Tolerant Cache Architecture for Ultra Low Voltage Operation. Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems, 2011.

[16] S.R. Nassif, N. Mehta, and Y. Cao. A Resilience Roadmap. Intl. Conf. in Design, Automation, and Test in Europe, 2010.

[17] P.P. Shirvani and E.J. McCluskey. PADded Cache: New Fault Tolerance Technique for Cache Memories. VLSI Test Symp., 1999.

[18] G.S. Sohi. Cache Memory Organization to Enhance the Yield of High-Performance VLSI Processors. Trans. on Computers, 1989.

[19] O. Temam and Y. Jegou. Using virtual lines to enhance locality exploitation. Intl. Conf. on Supercomputing, 1994.

[20] H.T. Vergos and D. Nikolos. Performance Recovery in Direct Mapped Faulty Caches via the Use of a Very Small Fully Associative Spare Cache. Computer Performance and Dependability Symp., 1995.

[21] P.V. Vleet, E. Anderson, L. Brown, J.L. Baer, and A. Karlin. Pursuing the performance potential of dynamic cache line sizes. Intl. Conf. on Computer Design, 1999.

[22] J. Wang, Y. Liu, W. Zhang, K. Lu, K. Qiu, X. Fu, and T. Li. Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing. Intl. Conf. on Parallel Processing, 2016.

[23] C. Wilkerson, H. Gao, A.R. Alameldeen, Z. Chishti, M. Khellah, and S.L. Lu. Trading off Cache Capacity for Reliability to Enable Low Voltage Operation. Intl. Symp. on Computer Architecture, 2008.

[24] S. Zafar, B.H. Lee, J. Stathis, A. Callegari, and T. Ning. A Model For Negative Bias Temperature Instability (NBTI) in Oxide and Kappa. Intl. Symp. on VLSI Technology, 2004.