Spatial Pattern Prediction Based Management of Faulty Data Caches

Georgios Keramidas, Michail Mavropoulos, Anna Karvouniari, Dimitris Nikolos
Department of Computer Engineering & Informatics
University of Patras
Patras, GR26500, Greece

***Abstract*—Technology scaling leads to significant faulty bit rates in on-chip caches. In this work, we propose a methodology to mitigate the impact of defective bits (due to permanent faults) in first-level set-associative data caches. Our technique assumes that faulty caches are enhanced with the ability of disabling their defective parts at cache subblock granularity. Our experimental findings reveal that while the occurrence of hard-errors in faulty caches may have a significant impact in performance, a lot of room for improvement exists, if someone is able to take into account the spatial reuse patterns of the to-be-referenced blocks (not all the data fetched into the cache is accessed). To this end, we propose frugal PC-indexed spatial predictors (with very small storage requirements) to orchestrate the (re)placement decisions among the fully and partially unusable faulty blocks. Using cycle-accurate simulations, a wide range of scientific applications, and a plethora of cache fault maps, we showcase that our approach is able to offer significant benefits in cache performance.**

I. Introduction

Over the last two decades, scaling of CMOS devises has provided remarkable improvements in performance of electronic circuits. However, as silicon industry is moving toward the "end of Moore's Law," increases in static [3] and dynamic [4] variations, wear-out failures [20], and manufacturing defects are affecting the yield and reliability of ICs [2]. As a result, the toolbox of the system designers must be always enhanced with new fault-tolerance techniques.

This is particularly true in on-chip caches for two reasons. First, an increasingly larger portion of the chip area is devoted to caches and this trend is expected to exacerbate in the many-core era. Second, caches are built with minimum sized, thereby prone to failure, SRAM cells. Recent technology roadmaps pinpoint the vulnerability problem in SRAM cells. According to [15], the predicted probability of failure (pfail) for SRAM cells is equal to 2.6e-04 due to random dopant fluctuations in 12nm technology. Furthermore, if we rely on aggressive voltage scaling techniques below safety thresholds, the SRAM pfail ramps up exponentially [19]. Therefore, it becomes critical to devise new fault-tolerant techniques to protect caches against hard faults. Obviously, these techniques have to be both scalable and performance effective, especially when the caches under consideration are close to core.

A significant amount of work targets to address the cache reliability concerns at the circuit level. Typically, those techniques are based on upsizing the memory transistors or by using alternative SRAM cell designs, such as eight- or ten-transistor cells [9]. Unfortunately, these techniques incur large area overheads and require considerable effort to redesign the underlying cache structure. A class of architectural level methods for masking out the memory faults relies on inserting redundant cache elements (blocks or columns) in the cache array [5,18]. However, due to the limited redundancy that can be afforded, this approach offers a limited capacity in the number of defects that can be tolerated (defined by the number of redundant elements) [1,10]. Another set of methods is based on the cache block/row/way disabling that is also suitable for low failure rate situations [12,13]. The usage of error correcting codes (ECC) to detect/correct the hard errors has also been proposed. However, the use of ECC is not practical for hard errors due to their large storage overhead and ECC repair time penalty rendering them not suitable for the latency-sensitive L1 caches. Furthermore, in the case of multi-bit hard errors, stronger ECC is required further prohibiting the in-line correction due to even larger overheads [19].

As industry moves into the near threshold region characterized by high fault probabilities, more scalable solutions are required. Many recent fault tolerant schemes are based on the concept of disabling cache portions, such as block or words that include defective bits, and reconfiguring operational ones (e.g., physical or logical neighborhood blocks). For example, PADed cache uses configurable address decoders that are programmed to select non-faulty blocks as replacements of faulty blocks [16]. The WDIS scheme combines two consecutive cache blocks into a single cache block, whereas the BFIX scheme sacrifices a sound cache block to repair defects in three other blocks [19]. The buddy cache pairs up two faulty blocks in a cache set to provide a functional one [8]. The salvage cache follows a similar approach by using a single faulty block to repair several others in the same cache set [7]. All those techniques require significant circuit modifications [1] which may pose a great burden to the designers of time-sensitive L1 caches.

In contrast to these approaches, in this work we propose to deal with the cache reliability problem at a very different level. Our proposal targets to better utilize the cache fault-free area by exploiting the low cache block utilization without resorting to non-scalable redundancy based solutions or to complex block remapping schemes. In contrast to prior techniques, striving to substitute every faulty subblock or word with a sound one is not always necessary, if the executing applications exhibit specific access patterns. More specifically, it is well known that caches are very inefficiently utilized because not all data, fetched into the cache to exploit spatial locality, is accessed [6,14]. Thus, if someone was able to predict the to-be-referenced words of a cache block and

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales "HOLISTIC". Investing in knowledge society through the European Social Fund.

978-3-9815370-2-4/DATE14/©2014 EDAA

accordingly fetch and place those data in the functional portions of a faulty cache frame (physical cache position), this will eliminate the need for extra redundant elements and/or the need for complex remapping mechanisms.

Based on those observations, our attempt to address the cache reliability challenge is to enhance the L1 faulty caches with a spatial pattern prediction mechanism that will provide the necessary information (unused portions of the requested blocks) to the underlying cache (re)placement logic. More specifically, we propose a fault tolerant aware (FTA) (re)placement policy that is able to take into account i) the faulty portions of the frames belonging into a specific cache set and ii) the spatial footprint predictions of the just requested blocks (misses) and issues informed (re)placement decisions in order to mask out the faulty cache parts. *To the best of our knowledge, this is the first approach that proposes dedicated fault tolerant aware cache (re)placement policies.*

Our approach relies on spatial footprint predictions (SFPs). While many sophisticated predictors are available [6,14], the proposed mechanisms are too fine grain, too complex and storage demanding for our purposes. As a result, we propose practical, instruction (PC) based, coarse grain spatial footprint predictors (CG-SFP) with a very limited range of prediction options. CG-SFP is designed to predict if a whole cache block or only the left or the right portion of a block will be accessed. The latter characteristic leads to frugal and scalable prediction structures requiring less than 128 bytes storage overhead.

Finally, we extensively evaluate our approach using cycle accurate simulations, a wide range of scientific applications (from SPEC2000 and SPEC2006 suite), and a plethora of fault maps in order to prove the stability of our proposal. Our results indicate a significant reduction in the resulting number of misses compared to a conventional cache managed with LRU replacement decisions. On average, 17.22% reduction in the reported misses is observed in a 16KB defective cache, 19.64% in a 32KB, and 21.18% in a 64KB cache.

Structure of the paper. Section II assesses the impact of defective bits in set-associative data caches and motivates this study. Section III describes our evaluation framework. Section IV presents our lightweight CG-SFP. Section V depicts our fault tolerant aware (re)placement scheme. Section VI provides our evaluation results. Section VII concludes this work.

II. Faulty Caches, Cache Associativity and Cache Utilization

Impact of defective bits. Fig. 1 depicts the effect of cache block disabling technique for various first-level data (DL1) cache sizes as averaged values for the two benchmark suites that we consider in this work. Details about the simulation and benchmark parameters are provided in Section III. The y-axis shows the relative increase in the number of misses with respect to a fault-free cache of the same configuration. Each line in the graph corresponds to a different cache size and benchmark suite, while three set-associative organizations are

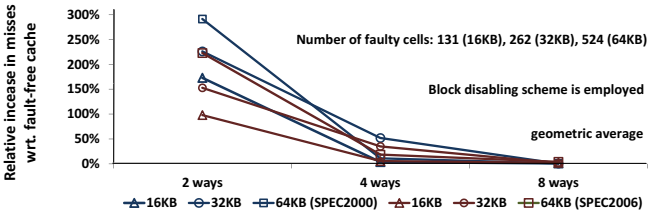

Fig. 1. The effect of faults in set-associative first-level data caches.

depicted (shown in x-axis). In addition, Fig. 1 shows also the number of faulty bit-cells for each cache size.

As Fig. 1 illustrates, low associative caches suffer the most from defective bits independently of the studied cache size (also noted by other researchers). In 8-way caches, the resulting increase in the number of misses due to faulty bits is far below 5% in all cases, whereas in the 4-way counterparts, the maximum average increase is relatively limited (51.89% in 32KB/SPEC2000 organization). In contrast, the 2-way configurations exhibit remarkable ramp-ups in the reported misses varying from 98.02% (16KB/SPEC2006) up to 291.3% (64KB/SPEC2000). As a result, in the context of this work, we opt to apply our fault-tolerant aware (re)placement methodology only in 2-way cache organizations, because 2-way caches experience the most serious performance degradation due to defective bits and most importantly this trend is independent of the executing benchmark suite (as depicted in Fig. 1). Obviously in direct map caches no-(re)placement decisions are applicable.

It is noteworthy to mention that the presented statistics reflect the average behavior of the corresponding suites hiding (for space reasons) the details of individual benchmarks. For example, in 32KB/4-way caches, 29 out of the 43 studied benchmarks exhibit an increase in the number of misses by more than 20% (which is not negligible!). Extending our approach to higher associative caches is left for future work.

Motivation. As noted, the cornerstone of this work is to explicitly manage faulty caches according to the spatial characteristics of the memory blocks. Partially faulty cache frames can host cache blocks with limited spatial footprints with minimal or no impact in performance. To quantify the potential of our approach, we analyzed the spatial locality footprints of a subset of SPEC2000 and SPEC2006 applications. Fig. 2 plots our profiling results as stacked bars.

The reported statistics are collected when the cache blocks are evicted from the cache assuming an LRU replacement policy and fault-free caches. In addition, we assume that the cache blocks are divided into two equally sized parts. The vertical axis depicts the percentages of the evicted blocks. The blue bars correspond to blocks that only 50% of the cached data is touched by the core, whereas the green bars corresponds to blocks with 100% utilization. Finally, there are three bars in each benchmark, one for each studied cache size.

As Fig. 2 indicates, different benchmarks exhibit different characteristics. More specifically, there are cases like *gap*,

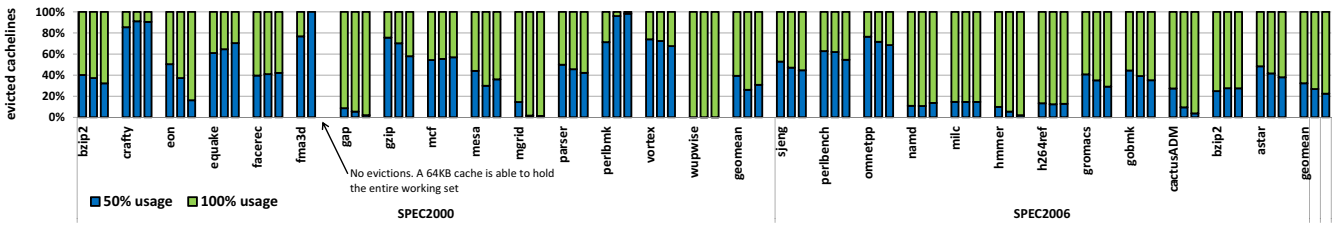

Fig. 2. Cache utilization breakdowns for various 2-way cache sizes (from left to right: 16KB, 32KB, and 64KB).

mgrid, *wupwise/SPEC2000, namd, hmmer, cactusADM/SPEC2006* that almost all the requested blocks are fully utilized (100% usage). At the opposite side, *perlbmk/SPEC2000/64KB* exhibits a very different behavior. All of its blocks have 50% utilization which eventually means that even cache frames with 50% fault-free parts are able to accommodate the whole working set without hurting performance. On average, for the 32KB organizations, in 18 benchmarks (out of 27) at least 35% of the requested blocks fall into the 50% usage category proving that it is ample room for optimizations. The question is how it is possible to identify the blocks with low spatial footprints at run-time and accordingly drive the (re)placement policy of the faulty caches. This is the subject of Section IV and Section V.

III. Evaluation Framework

Simulation infrastructure. To evaluate our approach, we used the SimpleScalar toolset. The simulated processor is a dynamic 2-issue core. In this work we use our FTA policy to manage the L1 data caches (we assume that faults are injected only in DL1s). We use various DL1 sizes, but we limit our analysis in 2-way caches. Our baseline machine includes a 1-cycle DL1, a 1-cycle, 16KB/4-way IL1, a 12-cycle, 256KB/8-way unified L2 cache, and a 200-cycle, 4GB main memory. The system bus is a 16 bytes wide split transaction bus. The block size is equal to 32 bytes in L1s and 64 bytes in L2.

Benchmarks. Two benchmark suites, namely SPEC2000 and SPEC2006, comprise our collection of benchmarks used in this work. However, due to space limitations, we present detailed results only for specific applications of these suites. The selected applications are chosen according to the following criterion: the increase in the number of misses compared to a non-faulty cache when the block disabling scheme is employed is more than 50%. This leaves at our disposal 27 applications (15 from SPEC2000 and 12 from SPEC2006). In all cases, we simulate 200M instructions after skipping 1B instructions to avoid unrepresentative startup behavior.

Baseline LRU (re)placement policy. As noted, we choose to apply our FTA cache management technique in DL1s. For a fair comparison, we assume that DL1s are already managed by a modified version of the LRU (re)placement algorithm (called baseline LRU). 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 can serve as replacement 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. Obviously, in the latter case, the just-requested cache word will not be eventually located into the cache, but the portion of the requested block that does not physically correspond to the faulty cache frame area will be cached. Finally, in all the other system caches, a conventional LRU replacement policy is employed.

Cache performance metrics. In order to focus on the actual strength of the proposed FTA scheme, we opt to illustrate our evaluation results using as metric the relative reduction of cache misses achieved by our proposal normalized to the misses reported by the baseline LRU. An interesting alternative would be to presents the performance improvements, in terms of IPC, however such approach can be disruptive in our case because the reported results will be heavily biased by the cache-miss-penalty-hiding capabilities of the processor and the overall memory system configuration.

information (2-bit long) from the SRAM array and feeds this

Block disabling schemes and fault model. Our technique assumes that the cache is equipped with the ability to disable cache portions (blocks or subblocks) that contain faulty bits upon permanent error detection [12]. In general, block or subblock disabling is an attractive option because of its low overhead e.g., 1-bit per cache block or subblock. This bit indicator will be called as faulty bit hereafter. In this work, we assume that the cache disabling scheme is applied in a subblock granularity (a faulty bit is assigned in every 16 bytes) in order to keep the area overhead in a reasonable size. Exploring finer granularities in left for future work. The detection of the defective subblocks can be done during manufacturing testing or by BIST-based periodic testing in the field. Any conventional BIST can be employed to perform these tests. A detailed description of BIST operation is out of the scope of this paper.

Another design concern is the choice of the appropriate faulty model used to study the effect of faults. There are two main parameters that must be taken into account: i) the number of faults and (ii) the location of the faults in the memory array. A detailed study about those issues can be found in [15]. According to [15], 100-1000 random fault maps are sufficient to obtain accurate average values for the miss rate in faulty caches. As for the predicted probability of failure (p_{fail}) for SRAM cells, we set the p_{fail} equal to 1e-03 [15]. Thus, we randomly produce 100 fault maps according to the above-mentioned p_{fail}.

IV. Coarse Grain Spatial Footprint Predictor

Spatial locality prediction provides the foundation of our methodology. Several studies have looked at improving the cache performance by using spatial footprint predictors (SFP) to detect spatial locality [6,14]. Although successful, the already proposed schemes are too fine grain and too storage demanding for our purposes. In contrast to those approaches, the range of the required predictions in our case is very narrow. Since in this work, we opt to divide each cache frame of faulty caches in two equal divisions (each division is marked by a separate faulty flag), coarse grain spatial footprint predictions can be issued. In other words, the proposed coarse grain spatial footprint predictor (CG-SFP) is designed to predict if the whole cache block or only the left/right portion of a block will be requested by the core (three decisions in total). The latter characteristic allows us to design frugal and scalable prediction structures which are described below.

A. CG-SFP Basic Functionality

The key idea of CG-SFP is to relate a cache block to the load/store instruction (PC) that accesses the relevant block and issues a prediction according to the instruction previous behavior. Fig. 3 depicts the structure of CG-SFP. The predictor is composed by a context addressable memory (CAM), called PC History Table (PCHT), that is responsible for storing active PCs. A second array (SRAM-based), indexed by the CAM, is the Spatial Footprint History Table (SFHT) which holds the predicted spatial footprint information for the corresponding PC. The operation of CG-SFP is as follows:

Lookup. Lookup is the operation that predicts the spatial footprints of the blocks touched by a PC. It is a straightforward procedure: the instruction (PC) that brings a new block into the cache (just missed block) is used as input to PCHT. A PCHT hit means that this PC has already appeared in the program execution while a miss indicates that this PC is encountered for the first time so no prediction can be made. If a match occur, the PCHT selects the corresponding predicted spatial footprint information to the FTA module (described in Section V).

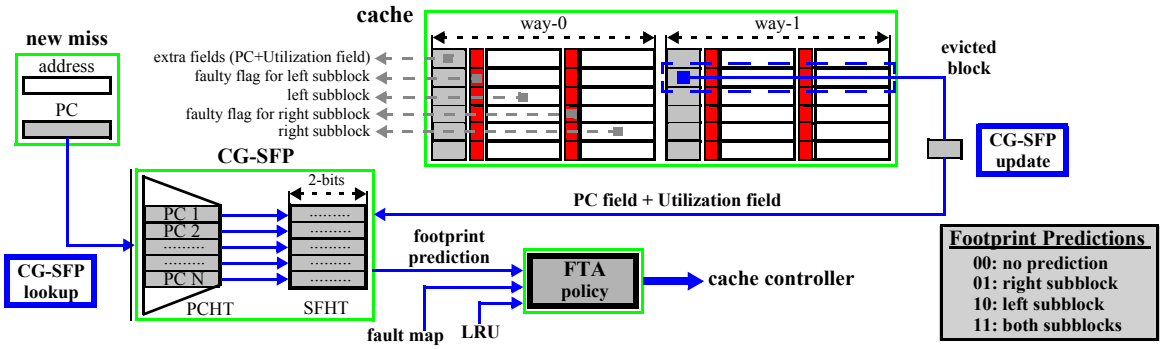

Fig. 3. The anatomy of CG-SFP predictor.

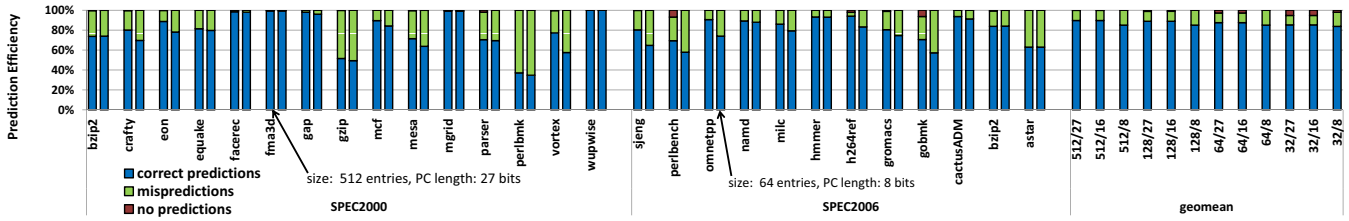

Fig. 4. Prediction accuracy for two predictor configurations (left part) and as averaged values for various sizes and limited PC bits (right part).

Update. Update is the operation to refresh the predictor with new information. The inputs to this mode of operation are the PC and the usage history of evicted blocks. As shown in Fig. 3, each cache frame is extended with two fields. The first field (PC field) is responsible to hold the PC that brought the specific block into the cache and the second field (utilization field) is designed to keep which subblocks (left, right, or both subblock) are touched by the core during the lifetime of the block into the cache. As a result, during block eviction, the two fields are forwarded to predictor. If no entry exists in the predictor for the given PC, a new entry is added and the gathered utilization history is stored in SFHT. If the predictor already contains information about the particular PC, then the corresponding SFHT entry is accordingly updated.

B. Design Choices and Practical Implementation

The purpose of CG-SFP is to provide the necessary information to mask out the memory hard faults in the target cache. In this section we discuss practical CG-SFP implementations and we show how those design choices influence the effectiveness of our predictor.

PC and Utilization fields update. As mentioned, each cache frame should be extended to accommodate two extra fields: the PC and the utilization history fields. Both fields are required for the CG-SFP update operation. Consider the PC field, one obvious design choice is if this information should be updated every time the specific block residing in the cache is accessed (experience a hit) by a different PC. In this work, we decided to update the PC field only during the service of a miss. First, in this way, the cache critical path (hit path) is not modified. Second, updating the PC field every time a hit occurs would increase the cache power consumption. Third and most importantly, by storing only the "first touch" PC, from the full sequence of PCs that touch a specific block, the CG-SFP can be implemented with significantly smaller number of entries. In addition, in this way the predictor update operation is out of the critical path (happens only during the service of a miss). Refreshing the utilization field (in case of a hit) does not also affect the cache cycle time, because the update of this field can be performed during driving the sought data to the core (in parallel to the cache output driver). Moreover, since the utilization field is 2-bits long, this update task can be performed by a simple combinational circuit, consisting by a two-level gate tree, which is negligible.

Predictor size and PC length. The CG-SFP storage requirements are mainly defined by the number of predictor entries and the size of each PC. Obviously, the SFHT table has a small effect in the storage overheads because each SFHT entry is 2-bits long. We performed simulations to identify a reasonable PCHT size, while we reduce the length of the captured PCs at the same time. The hope is to end up with a predictor of limited size and a small number of required PC bits without sacrificing the accuracy of the predictions.

Fig. 4 depicts the results of our analysis for a 32KB/2-way cache. The horizontal axis is divided into two main parts: the left part shows the gathered statistics for two predictor configurations and for the two studied benchmark suites, whereas the right part illustrates the geometric means over all the benchmarks for 12 configurations. To distinct each predictor configuration, we use the following notation: X/Y meaning X predictor entries and each entry is tagged by Y PCs bits (we always keep the lower order bits). Each bar in Fig. 4 depicts the percentage of correct, mis, and no predictions. Considering the per-benchmark statistics, we selected to show two configurations: 64/8 and 512/27 (no changes observed with more than 512 entries; 32-bits minus the byte offset).

As Fig. 4 indicates, our CG_SFP predictor reports great prediction accuracies in the majority of the benchmarks. The 512/27 CG-SFP offers a prediction accuracy of more than 80% in 18 benchmarks. However, there are cases like *perlbmk* and *gzip*/SPEC2000 in which our predictor fails to offer sufficient prediction accuracies. One possible solution for this would be to add in the predictor control logic an appropriate confidence mechanism, but we leave this analysis for future work. Interestingly, the 64/8 predictor is also proved very effective capturing almost the whole strength of the 512/27 predictor. On average, the 64/8 predictor is able to report 85.1% correct predictions (out of the total predictions). In the rest of this paper, we experiment with the 64/8 CG-SFP.

PC sampling. Our prediction scheme requires also modifications in the cache structure to accommodate the PC and utilization fields. Although, the selected 64/8 predictor

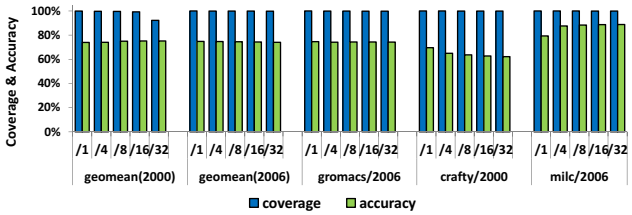

Fig. 5. Accuracy and coverage for various update rates.

relies its predictions in 8-bit long PCs, the total overhead of enhancing each cache frame with the required extra fields is still significant. To overcome this problem, we resort to a PC sampling approach. Not all cache frames should be augmented with the PC and utilization fields in order to provide safe update operations. In our design we select to insert the PC and utilization fields only to a limited number of cache frames (observation frames). As a first step, we limit the number of observation cache frames only to the first cache way. Our experiments reveal that no reduction in the accuracy of the predictor is observed. Moreover, we further reduce the number of observation frames by assigning one observation frame every N cache frames.

Fig. 5 shows our simulation results when N varies. The numbers juxtaposed the x-axis corresponds to the value of N. Due to space restrictions, we give the geometric mean of the two benchmark suites and three representative cases. The blue bars in Fig. 5 depict the coverage of the predictions, while the green ones show the prediction accuracies[1]. As we can see, while the coverage remain fairly stable when N varies, the accuracy results may i) drop as in *crafty* (six benchmarks belong in this category), ii) increase as in *milk* (four benchmarks), or iii) remain stable as in *gromacs* (17 benchmarks). Due to those fluctuations[2], the average accuracy stays almost constant. For the rest of this paper, we set the value of N equal to 16.

Time considerations. Since our target is to increase the reliability in the latency-sensitive DL1s, we have to be careful so as to not put any additional delays in the cache access path. Since the update and lookup of the prediction structures happen only during the service of a miss, they do not impact the critical (hit) path of the processor.

Overall Storage Requirements. The memory overheads introduced by our predictor are: 64 (entries) x 11 bits (for each entry —8 bits for each PC + 2 bits for the predicted footprint data + 1 valid bit) = 88 bytes. In addition, per cache frame modifications are also required. Assuming a 32KB DL1, the storage overheads are: 1024 frames x 10 bits (for each entry — 8 bits for the PC field + 2 bits for the utilization field) divided by (2 x 16) (2 because we choose to include those fields only in the first way, 16 because we consider a sampling rate equal to 16) = 40 bytes. Thus, the total storage requirements is equal to 128 bytes (less than 0.3% in a 32KB cache)[3].

V. Managing Faulty Caches

In this work, we propose the exploitation of low cache block utilization to explicitly manage defective caches. The target is to mask out the memory faults by carefully guiding the cache (re)placement decisions (i.e., to not let the memory faults to manifest themselves at the microarchitectural level). Our management methodology is performed in two levels which are explained below.

```
01: procedure FTA_POLICY // for a 2-way cache
02: inputs: prediction, LRU, set_fault_map
03: outputs: replacement
04: // prediction: 00 (no), 01 (right_subblk),
05: // 10 (left_subblk), 11 (two_subblks)
06: // replacement: way_0, way_1
07: // set_fault_map: one_subblk, three_subblks
08: // two_subblks_same_way, two_subblks_diff_way
09:      switch (set_fault_map)
10:       case 'one_subblk' :
11:       if prediction == 00 then
12:       replacement := LRU ;
13:       else if prediction == 01 or 10 then
14:       replacement := fault_way ;
15:      // way with fault subblk
16:       else // prediction == 11
17:       replacement := healthy_way ;
18:      // way with healthy subblks
19:       end if
20:      case 'two_subblks_same_way' :
21:       replacement := healthy_way ;
22:      case 'two_subblks_diff_way' :
23:       replacement := LRU ;
24:      case 'three_subblks' :
25:       replacement := healthy_subblk ;
26:      return replacement ;
27: // flip-bit is appropriately set in all cases
28: end procedure
```

Fig. 6. The proposed FTA policy.

At the lower level, each faulty cache frame is augmented by 1-bit (*flip-bit*). Flip-bit is applicable only when the corresponding physical frame contains one faulty part (defective subframe). If flip-bit is set, the sound cache physical part (subframe) hosts subblocks that normally would be located at the defective subframe. In contrast, if flip-bit is clear, the subblock resides in the correct physical location[4]. flip-bit allows us to make better utilization of the CG-SFP predictions, since our underlying FTA policy must not be aware of the exact physical positions of the requested blocks.

At the higher level, the proposed FTA policy is employed. Our FTA policy tries to orchestrate the (re)placement decisions between the fully and partially faulty cache frames by appropriately skewing the decisions of the baseline LRU scheme. More specifically, the output of CG-SFP, the decision made by the baseline LRU, and, of course, the fault map of the target cache set are exposed to the FTA policy module (Fig. 3). The FTA module is responsible to select the evicted way and make this decision available to the cache controller. The cache control logic is then responsible to handle the rest of the eviction/insertion operation.

The pseudocode of the proposed FTA policy is depicted in Fig. 6. On a miss, we search the corresponding cache set. Based on the faulty status of the target set (*line 9*), a different path is followed (switch cases in *lines 10*, *20*, *22*, and *24*). The switch case in *line 20* is invoked when the corresponding cache set contains two faulty subframes and both subframes coexist in the same frame. *Line 22* is responsible for the sets in which the two faulty subframes are spread across the two ways and *line 24* shows the pseudocode for the sets in which three faulty subframes exist. The switch cases in *lines 22* and *24* are inherently managed by the underlying flip-bit.

Obviously, the CG-SFP predictions are utilized when the accessed cache set contains one faulty subframe (*line 10*). In this case, the FTA module "sees" two frames which can accommodate blocks with different spatial footprints. Among

[1] Coverage is defined as the percentage of (correct predictions + mispredictions)/(total possible predictions). Accordingly, accuracy is defined as (correct predictions)/coverage.

[2] Further analyzing those fluctuations is left for future work.

[3] We compare our proposal against a faulty cache managed already by the baseline LRU policy, thus we do not account for the faulty flags.

[4] This can be achieved by XORing the flip-bit with the MSB of the offset of the requested block. Only one XOR gate is required in the whole cache. Locating one flip-bit in each frame requires an extra 64 bytes storage (for a 16KB cache).

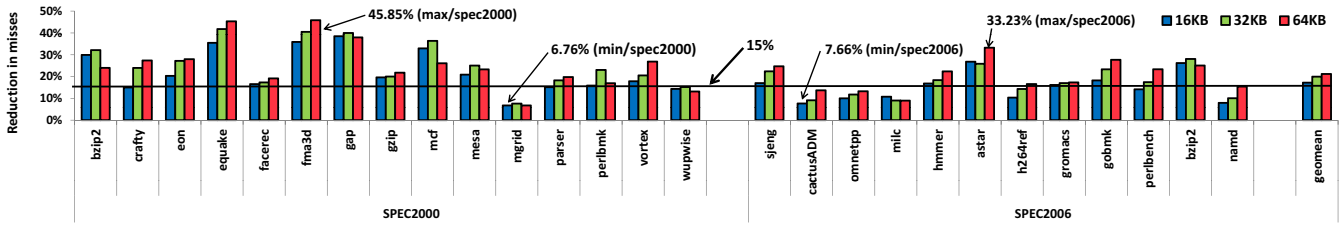

Fig. 7. Reduction of misses achieved in a FTA managed cache normalized to the misses of a conventional cache (averaged results among 100 random fault maps).

those frames, the sound one must host the blocks that will be fully utilized by the core (*lines 16-17*), whereas the partially faulty frame should be selected for the blocks with 50% utilization (*lines 13-15*). Finally, when our CG-SFP is not able to issue a prediction (*lines 11-12*), our FTA policy chooses to follow the decision made by the baseline LRU.

VI. Evaluation

This section presents our evaluation results. As mentioned, to study the effectiveness of our FTA technique, we randomly produce 100 fault maps assuming a predicted probability of failure (p_{fail}) for SRAM cells equal to 1e-03 [15]. According to the specific p_{fail} the number of faulty bit-cells is 131 in a 16KB, 262 in 32KB, and 524 in a 64KB 2-way cache. Applying the cache block disabling scheme in a subblock granularity (a faulty flag is assigned in each cache subframe) ends up with 17% reduction (on average) of the cache healthy storage area.

Fig. 7 shows our overall range of results across all fault maps. The y-axis in Fig. 7 shows the relative reduction in the number of misses achieved by the FTA scheme normalized to the misses occurred in a cache managed by the baseline LRU scheme. There are three bars attached to each benchmark (one for each cache size; 16KB, 32KB, and 64KB). Note that each bar represents the averaged result over all the generated fault maps. Finally, the x-axis is divided into three groups. The first two groups illustrate the per-benchmark results of the two studied benchmark suites, while the rightmost group of bars depicts the average statistics for each cache size across all benchmarks.

As Fig. 7 indicates our approach shows noticeable improvements over the baseline LRU. The FTA policy manages to ameliorate the reported misses by more than 15% in 21 benchmarks (out of 27) in the 32KB organizations, while similar improvements can be observed in the other cache sizes. In addition, there are no cases in which our proposal fails to reduce the number of misses. The largest benefits are achieved in *equake* and *fma3d.* Those two benchmarks combine both low spatial footprint memory blocks together with high CG-SFP prediction accuracies. At the opposite side, *mgrid* exhibits the lowest improvements. In *mgrid,* the majority of the cached blocks is fully utilized (100% usage), thus there is a limited potential for improvement. However, even in this case, the reduction is above 6% in all cache sizes. On average and as can be seen from the rightmost group of bars in Fig. 7, our proposal reduces the number of misses by 17.22% in the 16KB cache, 19.65% in the 32KB, and 21.19% in the 64KB cache.

VII. Conclusions

In this work, we provide a new methodology to mitigate the impact of permanent faults in first-level faulty data caches. In contrast to previous techniques, our approach to address the cache reliability challenge is to enhance the faulty caches with a spatial footprint prediction mechanism that will provide the necessary information (unused portions of the requested blocks) to the underlying cache insertion/eviction logic. To this end, we propose a dedicated fault-tolerant aware (re)placement policy that is carefully designed to better utilize the available fault-free area of the target cache. To facilitate storage-efficient spatial footprint predictions, we introduce a new class of coarse-grain, instruction-based spatial locality predictors that requires minimal hardware overheads. Our experimental findings for 100 fault maps, a wide rage of scientific applications from SPEC2000 and SPEC2006 suites, and various cache sizes reveal significant reductions in the resulting number of misses (19.38% on average).

References

[1] A. Ansari, S. Gupta et al., "Maximizing Spare Utilization by Virtually Reorganizing Faulty Cache Lines," in *Transactions on Computers*, 2011.

[2] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *Proc. of Design Automation Conference*, 2009.

[3] S. Borkar, T. Karnik et al., "Parameter variations and impact on circuits and microarchitecture," in *Proc. of Design Automation. Conference*, 2003.

[4] K. Bowman, J. Tschanz et al., "Circuit techniques for dynamic variation tolerance," in *Proc. of Design Automation Conference*, 2009.

[5] J.H. Edmondson, P.I. Rubinfeld et al., "Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor," in *Digital Technical Journal*, 1995.

[6] T. Johnson and W. Hwu, "Run-time adaptive cache hierarchy management via reference analysis," in *Proc. of Intl. Symposium on Computer Architecture*, 1997.

[7] C.K. Koh, W.F. Wong et al., "The salvage cache: a fault-tolerant cache architecture for next-generation memory technologies," in *Proc. of Intl. Conference on Computer Design*, 2009.

[8] C.K. Koh, W.F. Wong et al., "Tolerating process variations in large set associative caches: The buddy cache," in *Transactions on Architecture. and Code Optimization*, 2009.

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

[10] H. Lee, A. Cho et al., "DEFCAM: A design and evaluation framework for defect-tolerant cache memories," in *Transactions on Architecture and Code Optimization*, 2011.

[11] S. R. Nassif, N. Mehta, and Y. Cao, "A resilience roadmap," in *Proc. of Design, Automation, and Test in Europe*, 2010.

[12] A.F. Pour and M.D. Hill, "Performance Implications of Tolerating Cache Faults," in *Transactions on Computers*, 1993.

[13] S. Ozdemir, D. Sinha et al., "Yield-Aware Cache Architectures," in *Proc. of Intl. Symposium on Microarchitecture*, 2006.

[14] P. Pujara and A. Aggarwal, "Increasing cache capacity through word filtering," in *Proc. of Intl. Conference on Supercomputing*, 2007.

[15] D. Sanchez, Y. Sazeides et al., "An analytical model for the calculation of the expected miss ratio in faulty caches," in *Proc. of On-Line Testing Symposium*, 2011.

[16] P. Shirvani and E. McCluskey, "PADded cache: a new fault tolerance technique for cache memories," in *Proc. of VLSI Test Symposium*, 1999.

[17] P.R. Turgeon, A.R. Stell, and M.R. Charlebois, "Two Approaches to Array Fault Tolerance in the IBM Enterprise System/9000 Type 9121 Processor," in *IBM Journal*, 1991.

[18] H.T. Vergos and D. Nikolos, "Performance recovery in direct-mapped faulty caches via the use of a very small fully associative spare cache," in *Proc. of Computer Performance and Dependability Symposium*, 1995.

[19] C. Wilkerson, H. Gao et al., "Trading off cache capacity for reliability to enable low voltage operation," in *Proc. of Intl. Symposium on Computer Architecture*, 2008.

[20] S. Zafar, B. Lee et al., "A model for negative bias temperature instability (nbti) in oxide and kappa," in *Proc. of Symposium on VLSI Technology*, 2004.