A Novel Fault Tolerant Cache Architecture Based on Orthogonal Latin Squares Theory

Filippos Filippou[*], Georgios Keramidas[+┘], Michail Mavropoulos[*], Dimitris Nikolos[*]

[*]Department of Computer Engineering & Informatics, University of Patras, Patras, GR26500, Greece

[+]Think Silicon S.A., GR26505, Greece/[┘]Computer & Informatics Engineering Department, Technological Educational Institute of Western Greece, GR30020, Greece

***Abstract*—Aggressive dynamic voltage and frequency scaling is widely used to reduce the power consumption of microprocessors. Unfortunately, voltage scaling increases the impact of process variations on memory cells resulting in an exponential increase in the number of malfunctioning memory cells. As a result, various cache fault-tolerant (CFT) techniques have been proposed. In this work, we propose a new CFT technique which applies a systematic redistribution (permutation) of the cache blocks (assuming various block granularity levels) within the cache structure using the orthogonal Latin Square concept and taking as input the location of the malfunctioning cells in the cache array. The aim of the redistribution is twofold. First, to uniformly distribute the faulty blocks to sets and second, to gather the faulty subblocks to a minimum number of blocks, so as the fault free blocks are maximized. Our evaluation results using the benchmarks of SPEC2006 suite, 100 memory fault maps, and four percentages of malfunctioning cells show that our proposal exhibits strong capability to reduce cache performance degradation especially in situations with high percentages of faulty cells and compares favorably to already known techniques.**

I. Introduction

Reducing the supply voltage (Vdd) is an effective technique to lower the power consumption of ICs. Unfortunately, Vdd cannot be reduced below a certain level (called Vddmin), because some of the on-chip devices will become unreliable [4,16]. Low-voltage operation affects memory structures more than the logic elements resulting in an exponential increase in the number of malfunctioning SRAM cells [4,16] during aggressive voltage scaling. Hence, the high failure rate of SRAM structures at low voltage, limits the extent of voltage scaling in the processor. Therefore, it becomes critical to devise new fault-tolerant techniques to protect caches against process variation caused failures.

CFT techniques can be classified into four main categories. The first category includes circuit-level techniques [10], however these approaches incur large area overheads. The second class of CFT schemes relies on the usage of error correcting codes (ECC) to detect/correct the errors caused by the malfunctioning cells. Unfortunately, the large storage overheads combined with long error correction times render the ECC techniques impractical for first level caches. The third category relies on redundancy and tries to substitute the defective cache parts e.g., blocks, with functional spare elements [2]. However, due to the limited redundancy that can be afforded, these approaches offer a limited capacity in the number of defects that can be tolerated.

The fourth class of CFT techniques is based on the concept of disabling cache portions, such as blocks, sublocks or words that include malfunctioning cells, and apply several schemes to reduce the consequences of the disabled cache portions. In [12] the cache is equipped with the ability of dynamically varying its associativity according to the volume of defective cells. Some other techniques restructure the cache to combine partially faulty blocks to get a cache with a larger number of non-faulty blocks e.g. [16]. The later techniques require significant circuit modifications which increase the hardware overhead and affect the timing characteristics of the time-sensitive L1 caches. Some techniques remap faulty cache lines to operational ones. For example, PADed cache uses configurable address decoders that are programmed (via hard-wired fuses) to select non-faulty blocks as replacements of faulty blocks [14]. In [15], the authors proposed a rather complex scheme according to which a separate array is responsible to hold remapping information for every cache block. The latter technique requires large memory tables and complex lookup/redirection operations that reside in the cache critical path. In [1], a subblock disabling based scheme using dynamic address remapping by XORing the cache set index and a counter was proposed to reduce performance variation across different cores. However, subblock disabling i) adds a significant burden in order to maintain the inclusion/exclusion properties of the caches, ii) further complicates the underlying cache coherency protocol, and iii) requires sophisticated cache (re)placement policies in order to be effective [9].

The goal of "Block Remap with Turnoff" scheme [8] (called BRwT hereafter) is to achieve a uniform distribution of the faulty blocks per cache set. The permutation relies on simple XOR operations driven by a limited set of programmable registers (called permutation registers, PRs, hereafter). More specifically, by XORing the cache set index with a proper, pre-calculated permutation value, it is possible to re-distribute the cache blocks (healthy and faulty) in the memory array with meager hardware overheads. In [8] an exhaustive search of all possible block remaps that can be formulated by one-hot permutation vectors was proposed. By extensive experiments (100 random fault maps), we reveal that the one-hot policy of BRwT offers a limited capacity in recovering the extra misses caused by the malfunctioning SRAM cells. Moreover, an inherit characteristic of BRwT is that the size of the fault free area of the cache remains constant.

The main contributions of this work are:

- We examine the cache behavior under various percentages of malfunctioning cells (pfails) and cache configurations. Our analysis reveals that the non-uniform distribution of the faulty blocks into the cache sets is one of the main sources of poor cache performance.
- We propose a systematic method based on the Orthogonal Latin Square theory [3,7] for i) distributing uniformly the faulty blocks into the cache sets and ii) gathering the faulty subblocks (at several granularity levels) into a minimum number of faulty blocks, so as the number of the healthy

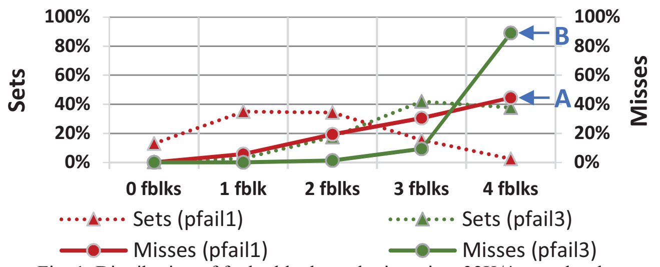

Fig. 1. Distribution of faulty blocks and misses in a 32K/4-way dcache

blocks is maximized. In the subblock disabling technique [1], a malfunctioning cell in the tag renders the whole corresponding block as faulty. Our method corresponds faulty tags to faulty data blocks (to the extent is possible) in order to increase the number of fault free blocks. Note that in our case, disabling takes place at block level eliminating the drawbacks of the subblock-level disabling mechanism.

- The evaluation of the proposed recovery mechanism is based on both trace and cycle-accurate simulations (based on gem5) using the benchmarks of SPEC2006 suite, 100 random memory fault maps, and four pfails. The power results are extracted from McPAT [11].

The theory of Orthogonal Latin Squares [3,7] was applied also in [5] targeting to increase the performance of faulty Branch Target Buffers (BTB) in OoO processors. The main differences between this work and [5] are:

- In this work, we apply two techniques for the reduction of the number of the faulty blocks. In [5], no attempt to reduce the faulty blocks has been taken.
- The aim of [5] is to distribute the faulty blocks so as no set to be fully faulty. In this work, our goal is to uniformly distribute the faulty blocks into the cache sets. Thus, our goal is more challenging and demanding.
- In this work, we use more than one PRs per way of the cache array, while in [5] one PR per way was employed.

Structure of the paper. Section II motivates this study. Section III outlines our evaluation framework. Section IV presents the proposed CFT mechanism and describes the hardware overheads of our approach. Section V provides our experimental findings and Section VI concludes this work.

II. Motivation

Omitting at the point the simulation details, in this section we quantify the impact of the distribution of faulty blocks (within the memory area) on the number of the resulting cache misses. More specifically, we categorize the number of extra (due to faults) misses according to the number of faulty blocks residing in the associated cache sets that generated those extra misses. Fig. 1 depicts the percentage of sets with i faulty blocks (left vertical axis/dashed lines) and the percentage of the extra misses in each group of sets with i faulty blocks (right axis/solid lines) assuming a 32KB/4-way cache.

By concentrating on the right points on the x-axis of Fig. 1 (fully faulty sets), it is clear that for pfail1, even the number of the fully faulty sets is below 2.5% of the total sets, those sets are responsible for the 44.5% of the extra (due to faults) misses (marked by "A"). In the higher pfai3, this number ramps-up exponentially (89.1%) (marked by "B"). In general, the number of misses generated by each set category (sets with i faulty blocks) increases significantly as the number of i steps up. Even in the case that the percentage of sets with i faulty blocks is significantly smaller than those with j faulty blocks, with i > j, the number of the misses caused by the sets with i faulty blocks is significantly larger.

Table 1. CPU/memory system configuration and pfails

Parameters	Settings
Pipeline Depth	10
Combined Branch Predictor (Tournament Predictor)	History Table: 2K-entries, Local/Global Pred.: 2K/8K-entries, Choice Pred.: 8K-entries, BTB: 4K-entries
L1 Instruction & L1 Data Caches	32kB, 4-way, 64B-block, 2-cycles, LRU, 16 MSHRS, 8 Write Buffers
L2 Cache	8MB, 8-way, 64B-block, 20-cycles, LRU, 32 MSHRS, 32 Write Buffers
DRAM	2GB, 50ns, 12.8GB/s
Process Technology	22nm SOI
Nominal Operation	2.5 GHz, 1Volt
Studied SRAM percentages of malfunctioning cells (pfails) [6]	*pfail1:* 1e-03 (900 MHz, 525 mV)
	pfail2: 2e-03 (800 MHz, 500 mV)
	pfail3: 3e-03 (700 MHz, 475 mV)
	pfail4: 4e-03 (600 MHz, 450 mV)

Uniformly (re)distributing the faulty blocks across the cache sets, the number of fully faulty sets will be reduced even if not nullified. Moreover, as Fig. 1 implies, even if we fail to achieve a uniform distribution of the faulty blocks across the cache sets, it is important (wrt. the number of misses) to reduce the number of sets with the largest number of faulty blocks. Motivated by this, in this work we propose a new technique which applies a systematic redistribution of the cache blocks to sets (assuming various block granularity levels) taking as input the location of the malfunctioning cells within the memory array. The aim of the redistribution is twofold. First, to uniformly distribute the faulty blocks to sets (or at least to reduce the sets with the largest number of faulty blocks) and second, to gather the faulty subblocks to a minimum number of blocks so as the number of the fault free blocks, and hence the effective cache size to be maximized.

III. Evaluation Framework

Simulation infrastructure. Our evaluations are based on trace-based simulations for cache misses studies and cycle-accurate, timing simulations for performance experiments. Due to simulation time restrictions, the first set of simulations is used for exploring the various parameters of our mechanism as well as to compare our proposal against two related CFT techniques. The second set of simulations is based on the gem5/x86 simulator assuming that both L1 data and instruction caches are faulty. The simulated processor is a dynamic 4-issue core with 128-entries instruction window. The remaining core and memory system configurations are illustrated in Table 1. The whole SPEC2006 suite (29 benchmarks in total) with the reference inputs is used. In all cases, we simulate the most frequent Simpoint sample for 200M instructions after a warmup period of 50M instructions.

Failures. In our method, the cache blocks containing malfunctioning cells are disabled, so it is not necessary to consider a specific fault model. Although disabling takes place at block level, since our proposal applies redistribution at subblock level, we consider that each subblock is guarded by a separate fault bit; a value "1" denotes that the subblock contains malfunctioning cells. 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 employing more reliable SRAM cells [1,2,10]. For each pfail (see Table 1) and benchmark, we performed simulation for 100 random fault maps [13]. In a real system, the fault map for each state of operation (V/f) can be produced

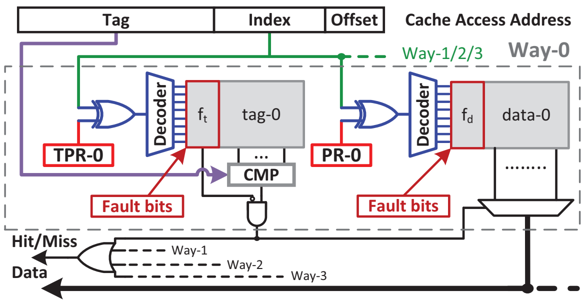

Fig. 2. A high level diagram of the basic permutation scheme

during production testing or under periodic BIST based testing in the field of the application in order to take also into account failures due to aging. A detailed description of the testing operation is out of the scope of this work.

It is important to note that for each pfail and each CFT technique, the number of misses of each pair (benchmark – fault map) is normalized with respect to the number of misses of the initial faulty cache[1] for that benchmark and fault map. This is done for all benchmarks and fault maps i.e., for each technique for a given pfail, we run 2900 simulations (29 benchmarks x 100 fault maps).

Reconfiguration Strategy. Reconfiguring the proposed mechanism at run-time requires to writeback the dirty blocks to the next cache level and flash the cache contents. In this work, we assume that a monolithic Voltage/frequency (V/f) level is enforced for the whole execution and the permutation vectors (one for each V/f level) are calculated offline.

IV. Multi-Dimensional Permutation Mechanism

The proposed method is a combination of four techniques: i) inter-way permutation, ii) block-level intra-way permutation, iii) subblock-level intra-way permutation, and iv) horizontal partitioning into logical groups. More specifically, *inter-way permutation* is applied at block level in the ways of the data array. Its purpose is the uniform distribution of the faulty blocks into the cache sets. *Block-level intra-way permutation* is applied between the tag and data array of each way. Its aim is to align the faulty tags to faulty data blocks so as to increase the number of the fault free blocks (tag - data). *Subblock-level intra-way permutation* is applied in the data array of each way. The target is to gather the faulty subblocks of the data array to a minimum number of blocks so as the fault free blocks are maximized. The *horizontal partitioning into logical groups* of sets offers the possibility to enforce independent inter-way/intra-way permutations in the sets/blocks belonging to the same group. Note that all the above techniques are orthogonal to each other.

Our proposal is based on the theory of Orthogonal Latin Squares [3,7]. A direct application of the approaches [3,7] in our case would aim to find an assignment so as each cache set would contain at most one faulty block. Obviously such a solution could not be found in large pfails, thus it is not applicable in today's process technologies. To achieve the design objectives of our approach, we extended/generalized the approach of [3]. As shown in Fig. 2, the basic block-level permutation scheme can be implemented using XOR gates and permutation registers (PR). The problem is formulated as: given the locations of the malfunctioning cells in the cache memory array, we have to derive the required values of the

[1] A cache structure in which no CFT technique is employed.

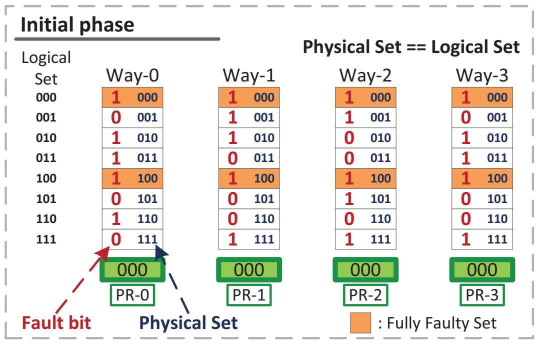

Fig. 3. Initial state for the inter-way working example

PRs i.e., *to formulate suitable logical sets by calculating suitable permutation vectors.*

Inter-way permutation. The inter-way permutation scheme is described by a working example. Fig. 3 depicts the initial cache state assuming a 32-frames/4-way cache, while the steps of our approach are illustrated in Fig. 4. Each cache block in Fig. 3 contains two numbers. The red number is the fault bit and the number in the right corresponds to the physical set of the specific block. Initially, the logical and physical sets are identical; the PRs of each way (shown in the bottom of Fig. 3) are set to "0". Finally, the fully faulty sets are marked in brown background. As noted, the goal is to calculate a suitable permutation value for each register (PR-Z) of each associative way-Z. We define the following terms:

- FBwZ: vector containing the locations of the faulty blocks within each way-Z
- FSbN: vector containing the locations of the faulty sets with exactly N faulty blocks
- CPR: candidate PR values
- XFB: vector derived by XORing the elements (in an all-to-all fashion) of two FBwZ vectors or between an FBwZ vector and a FSbN vector. XFB includes also the frequency of appearance of each result

As indicated in Fig. 4, the process begins (***step-1***) by considering the first two ways (way-0 and way-1). The target is to devise a value for PR-1 that will result in the minimum number of conflicting sets i.e., cache sets with multiple faulty blocks. By XORing the FBw0 and FBw1 vectors, the XFB vector is derived (Fig. 4). The least repeated values in the XFB vector are placed into the CPR vector and one of them (randomly selected) is assigned to PR-1. The logical sets formulated after the new value of PR-1 can be seen in the left box of Fig. 4. The strength of our approach relies on the fact that we are able to extract the conflicting or the partially conflicting cache sets after the usage of PR-1. In particular, during step-1, two new vectors are derived: FSb1 and FSb2. The first vector contains the sets which have one faulty block (partially conflicting), while the second vector contains the locations of the conflicting (fully faulty) cache sets.

The next phase (***step-2***) follows exactly the philosophy of step-1. The only difference is how the XFB vector is derived. In particular, two XFB vectors are calculated by XORing the FBw2 with the FSb1 and FSb2 vectors (from step-1) respectively. At this point, we assign a higher priority to XFB2 entries in order to minimize the number of fully faulty sets. In other words, the least repeated value from XFB2 are the first level candidates and one of them (randomly selected) is forwarded to PR-2 register. Finally, a new FSb3 vector is created and the FSb1 and FSb2 vectors are updated. The FSb3 vector contains the new cache sets that have one faulty block in every cache way after the application of the new value in PR-2.

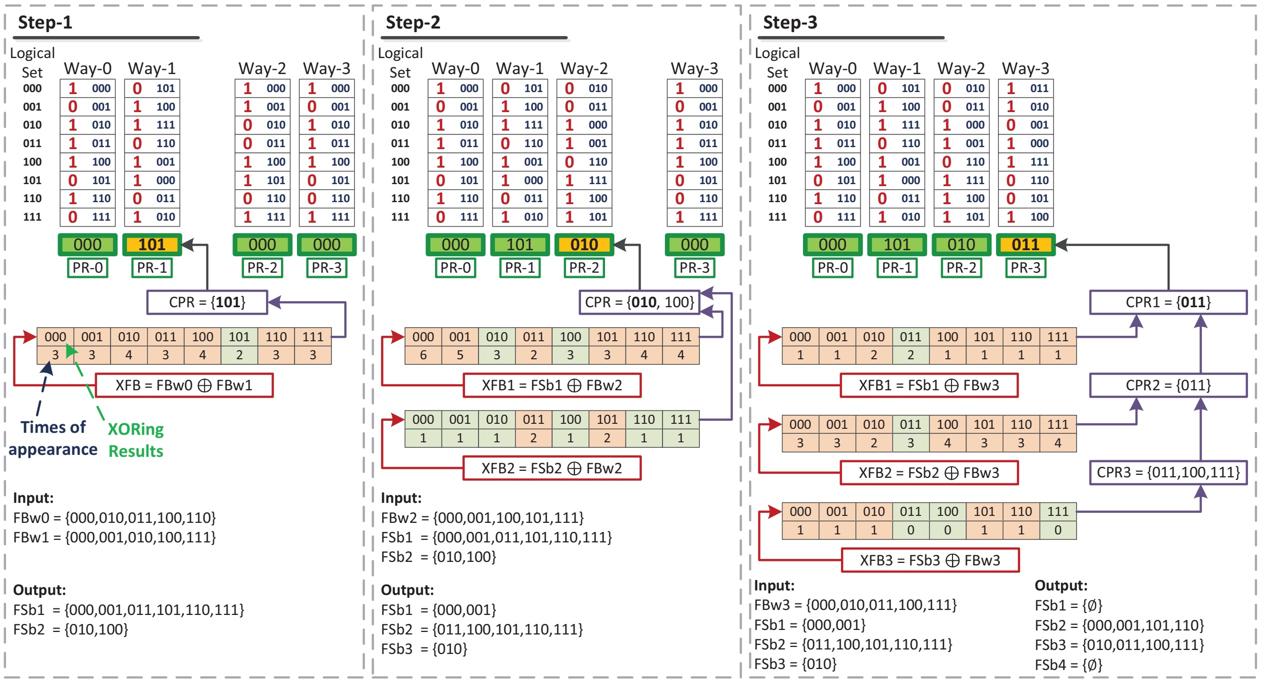

Fig. 4. Step-by-step process of the inter-way permutation algorithm

The remaining process (***step-3***) is fairly similar to step-2 and for space reason it is omitted. However, Fig. 4 presents the required calculations and the derived vectors. Note that as indicated by the FSb4 vector, *our proposal manages to eliminate the fully faulty sets by appropriately distributing the faulty blocks within the memory array*.

The **block-level intra-way permutation** can be considered as a special case of the subblock-level intra-way permutation whose description follows.

Subblock-level intra-way permutation. The objective of this scheme is to virtually merge the faulty subblocks in the least number of faulty blocks, so as the fault free blocks are maximized. In practice, the methodology is similar to the one described in the inter-way permutation scheme and the only difference is that the PR values are selected by exactly the opposite criterion. A working example is illustrated in Fig. 5 assuming a subblock_size equal to block_size/2.

Since permutation is applied in subblock level, each subblock is accompanied by a fault bit. The scheme is applied independently for each way of the data array. Due to the small size of the tags we do not apply this technique in the tag array. The left box of Fig. 5 shows the initial state of the way, while the right box shows the output of the intra-block permutation scheme. We consider that the way-Z consists of two parts called subway-0 and subway-1 (sway-i) respectively and each sway-Y has a unique permutation register PR-ZY. sway-0/sway-1 are the left/right subblocks of the blocks of each way.

As in the previous case, we define the following terms:

- FBwZsY: vector containing the locations of the faulty subblocks within the sway-Y of each way-Z
- FBwZ: vector containing the fully faulty blocks in way-Z
- CPR: candidate PR values
- XFSB: vector derived by XORing the elements (in an all-to-all fashion) of two FBwZsY vectors or between an FBwZsY vector and a FBwZ vector. XFSB includes also the frequency of appearance of each result.

The process begins (***step-1***) by XORing the FBwZsY vectors and as a result the XFSB vector is derived. On the

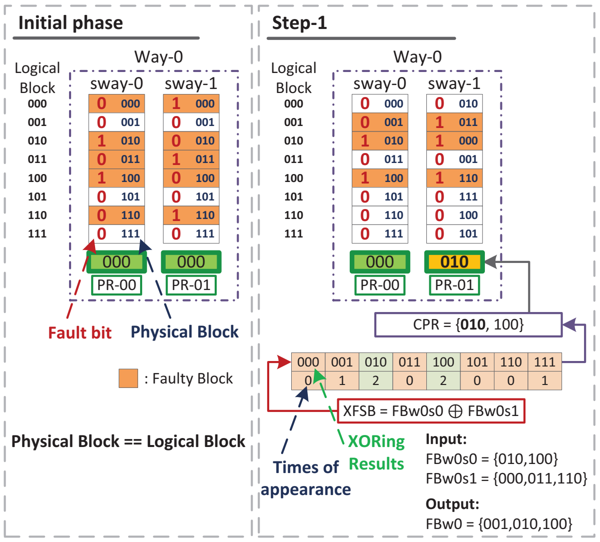

Fig. 5. Step-by-step process of the intra-block permutation algorithm

contrary to the inter-way scheme, the most repeated values of XFSB are placed to the CPR vector (these numbers maximize the number of fully healthy blocks). Finally, in the case that a block is divided to more than two subblocks, the process will continue for the next sub-way taking as input the results of step-1 (similar to step-2 and step-3 of the inter-way scheme).

Horizontal partitioning into logical groups. According to this scheme, the cache sets are divided into groups and each group will contain the same number (power of two) of cache sets. The key point is that each group can have its own PRs offering the possibility of applying independent permutation vectors in the groups of each way. The size of PRs in each group is equal to the set index reduced by $\log_2(i)$, where i is the number of the formulated groups.

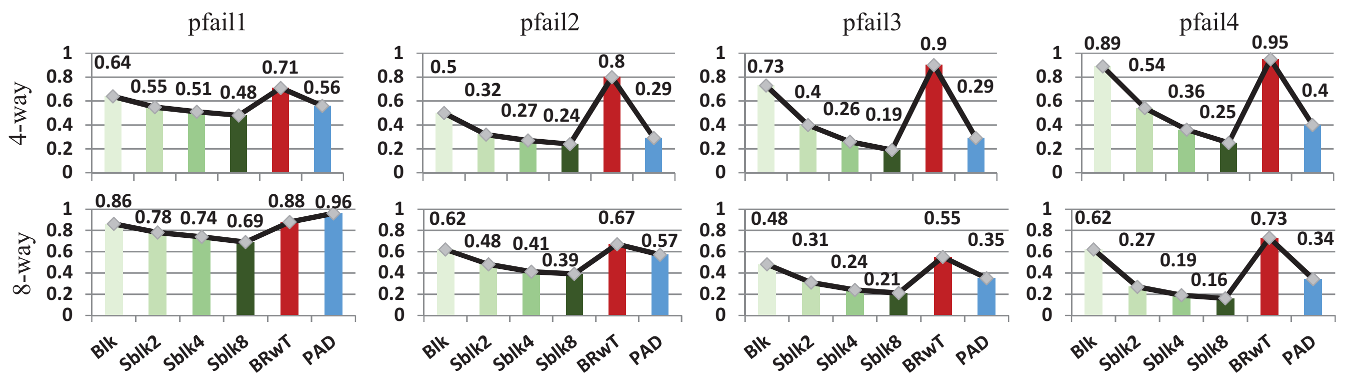

Fig. 6. Misses in a 32KB L1 data cache when different CFT methods are used. Results are normalized to the initial faulty cache.

Efficient combination of permutation schemes. The most efficient order of applying the above schemes (as we did in this work) is: i) horizontal partitioning into logical groups scheme, ii) subblock-level intra-way scheme, iii) inter-way scheme, and iv) block-level intra-way scheme. The reason that the subblock-level intra-way scheme must be applied before the inter-way counterpart stems from the fact that the target of the former is to maximize the number of the fault free blocks (partially faulty blocks are disabled), so as the inter-way algorithm can be employed in a more efficient manner.

Cost in Hardware and Time. Let w be the number of cache ways, s the number of sets, j and k the number of groups in the tag and the data array respectively, and z the number of subways in the data array i.e., the number of subblocks per block. Although the disabling takes place at block level, the intra-way permutation technique in subblock level should be aware of the faulty subblocks, thus each subblock of the data array must be protected by a fault bit. Due to the small size of the tag array, block level disabling is employed (thus one fault bit is required per tag). Therefore, the size of the fault map in bits is equal to $F = s\times w+s\times w\times z$.

In addition, the number of registers in bits is equal to:

$$R = j\times w\times log_2(s/j)+k\times w\times z\times log_2(s/k)$$

The above relations for a 32KB/4-way cache with 64-bytes blocks (s=128) and j=2 (due to the small size of the tag array a greater value is not required), k=4 and z=4 we get:

$$F = 2560 \text{ bits and } R = 368 \text{ bits}$$

The above overheads are less than 6% of the total cache memory area.

The use of a separate decoder for every way of the data array does not add any cost, because this design paradigm is already followed in today's process technologies [11]. Depending on the timing characteristics of the cache, often more than one decoders are used per way (due to horizontal partition of the cache into subarrays of sets or a partition of the ways of the data array into subways) [14]. The designer can choose the values of the parameters j, k and z in such a way to achieve the best performance and/or power consumption characteristics. Note that as the number of decoders increases, the timing characteristics of cache are improved.

Finally, the delay of the additional XOR gates (the PRs do not add any delay), although it resides on the critical path, it does not *affect the cache cycle (pipeline) time*. This is because under today's wire-limited technologies, cache latency is dominated by the bit/wordlines and h-tree traversals times (delay inside MAT plus the wordline/bitline reset/restore delays) [11]. Thereby, our proposal does not influence the operational frequency of the target faulty cache.

V. Experimental Results

The first part of this section presents our trace-based simulation results using the cache misses as a metric. The goal of this part is to explore various parameters of our approach assuming different cache organizations as well to compare the proposed mechanism against related CFT schemes. The second part presents the EDP (Energy-Delay Product) reduction achieved by our proposal using the gem5 and McPAT tools [11]. Table 1 contains the McPAT configuration parameters. Note that EDP refers to the whole processor (core, L1s plus L2 caches). In the latter part, due to simulation time restrictions, only one configuration is assumed in which faults are inserted in both instruction and data, first level caches.

Competitive Mechanisms. To the best of our knowledge, BRwT [8] and PADed cache [14] are the only CFT schemes that follow the philosophy of cache block "remapping," thus they can be directly compared to our technique. In practice, BRwT is the synergy of two techniques. The first one targets to distribute uniformly the faulty blocks to the cache sets. The second technique employs slower access times in the fully faulty sets (extra cycles are introduced), while in the remaining sets, the faulty blocks are disabled and these sets are accessed in the normal (nominal) access time. Since the second technique can be used in synergy to our technique, we will compare our method against only the first technique of [8].

The PADed cache is configured to operate in two programmable levels (using the MSBs) [14]. Note that for each programmable level of the decoder i) a bit of the index should be stored in the tag and ii) extra circuitry is required for generating the extra control signals of the programmable decoders. Another drawback of the PADded cache is that the LRU replacement policy cannot be applied because more than one blocks with different indexes might correspond to the same cache frame. Hence, random replacement policy is assumed.

Trace-based Evaluation. Fig. 6 shows the misses of our proposal (for various versions), the BRwT, and PADed cache techniques. All results are normalized to the misses of the initial faulty cache. In all versions, the tag array is partitioned into two groups while the data array into 4 groups. Blk denotes that permutation is performed only at block-level, while Sblk2, Sblk4, and Sblk8 denote that the intra-way permutation at subblock level is implemented with two, four, and eight subblocks per block respectively. Lower bars in Fig. 6 indicate a more efficient recovery of the extra (due to faults) misses.

From Fig. 6 we can see that after the application of our method the misses due to faulty blocks are reduced from 52% up to 84% depending on the cache associativity and the pfail value. Moreover, we can see that i) for constant associativity the efficiency of our method is improved as pfail increases up to a point and ii) for small pfails e.g., pfail1, as the associativity

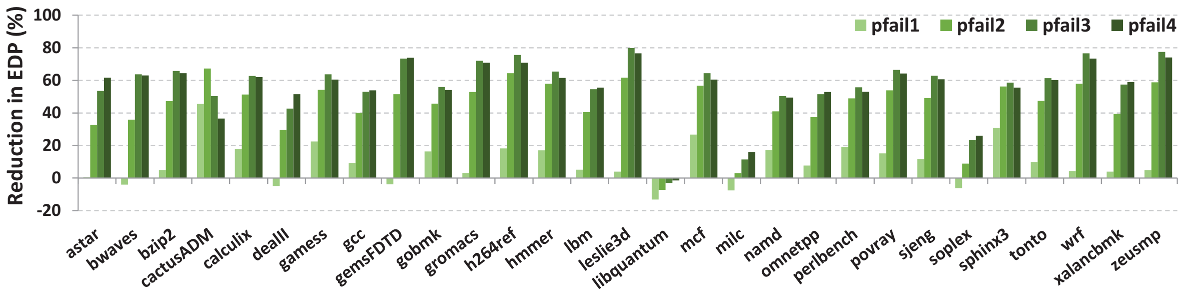

Fig. 7. EDP reduction when the proposed permutation mechanism is applied compared to the faulty case (32KB first level instruction and data 4-way associative caches)

increases the normalized misses increase too. On the contrary for large pfails as the cache associativity increases, the number of the normalized misses is reduced. The reason is twofold. For caches with constant capacity and block size, the percentage of the (almost) fully faulty sets increases as the value of pfail increases (for constant associativity), while decreases with the associativity (for constant value of pfail). In addition, when the percentage of the (almost) fully faulty sets is large e.g., in pfail4, the efficiency of our method, wrt. the uniform distribution of faulty blocks among sets, increases with the cache associativity.

Cycle-Accurate Evaluation. Fig. 7 shows the EDP reduction achieved by our permutation mechanism (assuming four subblocks per block and a horizontal partition of four groups in the data array) compared to the faulty case. The x-axis contains all the SPEC2006 benchmarks. There are four bars attached to each benchmark corresponding to different pfails. As Fig. 7 indicates, our proposal manages to significantly reduce the EDP in almost all benchmarks and pfails. For example, 21 and 26 out of 29 applications report an improvement well above 40% in pfail2 and pfail3 respectively. On the contrary, across all pfails, there are nine cases (pfail/benchmark) out of 116 (4x29) that our proposal reports an increase in EDP. This behavior is more pronounced in *libquantum*. In fact, *libquantum* is the only benchmark with negative EDP in pfail2, pfail3, and pfail4. *libquantum* is the most memory-demanding SPEC2006 application characterized by large working sets and a rather L1 cache insensitive behavior, thus cannot benefit by a more effective CFT mechanism. As such, the extra cost introduced in the cache structure to support our permutation mechanism results actually in an increase in the EDP.

On average, pfail3 exhibits the largest improvements over the faulty cache (56.76% on average), while remarkable EDP reductions are reported in the other pfails: 9.52%, 44.26%, and 55.84% in pfail1, pfail2, and pfail4 respectively. The average EDP in pfail4 is faintly less than the EDP in pfail3. This is due to the fact that in pfail4 the remaining fault-free area (resulted after our permutation technique is applied) is still not capable to fit the majority of the working sets of the executed applications, thus the improvements are slightly less compared to pfail3, but still over 55% in both cases.

VI. Conclusions

In this paper, we address the problem of process variation in first level caches operating at low supply voltages. First, we investigate the distribution of the faulty blocks to the cache sets and its impact on cache misses. By taking the latter information into account, we propose a novel, systematic permutation method for i) gathering the faulty subblocks of each way into a minimal number of faulty blocks and ii) distributing uniformly the remaining faulty blocks to the cache sets. Using cycle accurate simulations and the benchmarks of SPEC2006 suite, our experimental results prove that our proposal is able to offer a remarkable reduction in processor-wide EDP; well-above 40% over the faulty case for three out of four percentages of malfunctioning cells.

References

[1] J. Abella, J. Carretero et al. Low Vccmin Fault-Tolerant Cache with Highly Predictable Performance, in *Proc. of Intl. Symposium on Microarchitecture,* 2009.

[2] A. Ansari, S. Gupta et al. Zerehcache: Armoring Cache Architectures in High Defect Density Technologies, in *Proc. of Intl. Symposium on Microarchitecture,* 2009.

[3] D.C. Bossen, C.L. Chen, and M.Y. Hsiao. Fault Alignment Exclusion for Memory Using Address Permutation, in *IBM Journal of Research and Development*, 1984.

[4] Z. Chishti, A.R. Alameldeen et al. Improving Cache Lifetime Reliability at Ultra-Low Voltages, in *Proc. of Intl. Symposium on Microarchitecture,* 2009.

[5] F. Filippou, G. Keramidas et al. Recovery of Performance Degradation in Defective Branch Target Buffers, in *Proc. of On-line Testing Symposium,* 2016.

[6] M. Gottscho, A.B. Mofrad et al. Power/Capacity Scaling: Energy Savings with Simple Fault-Tolerant Caches, in *Proc. of Design Automation Conference,* 2014.

[7] M.Y. Hsiao and D.C. Bossen. Orthogonal Latin Square Configuration for LSI Memory Yield and Reliability Enhancement, in *Transactions on Computers*, 1975.

[8] M.A. Hussain and M. Mutyam. Block Remap with Turnoff: A Variation-Tolerant Cache Design Technique, in *Proc. of Asia and South Pacific Design Automation Conference, 2008.*

[9] G. Keramidas, M. Mavropoulos et al. Spatial Pattern Prediction based Management of Faulty Data Caches, in *Proc. of Intl. Conference in Design, Automation, and Test in Europe, 2014.*

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

[11] S. Li, J.H. Ahn et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures, in *Proc. of Intl. Symposium on Microarchitecture,* 2009.

[12] M. Mavropoulos, G. Keramidas et al. A Defect-Aware Reconfigurable Cache Architecture for Low-Vccmin DVFS-Enabled Systems, in *Proc. of Intl. Conference in Design, Automation, and Test in Europe,* 2015.

[13] 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.

[14] P.P. Shirvani and E.J. McCluskey. PADded Cache: a New Fault Tolerance Technique for Cache Memories, in *Proc. of VLSI Test Sympocium,* 1999.

[15] J. Wang, Y. Liu et al. Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing, in *Proc. of Intl. Conference on Parallel Processing, 2016.*

[16] 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.