Instruction Based Management of Faulty Data Caches

Georgios Keramidas, Michail Mavropoulos, Anna Karvouniari, Dimitris Nikolos

Department of Computer Engineering & Informatics, University of Patras, Greece

ABSTRACT

We propose a new approach to mitigate the impact of faulty bits in data caches. Our technique assumes that faulty caches are enhanced with the ability of disabling their defective parts at cache subblock granularity. Our experiments 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). We propose frugal PC-indexed spatial predictors to orchestrate the (re)placement decisions among the fully and partially faulty blocks. Using cycle-accurate simulations, we show that our approach is able to offer significant benefits.

KEYWORDS: Faulty caches, Spatial characteristics, Fault tolerance, Prediction.

1 Introduction

Over the last two decades, scaling of CMOS devises has provided remarkable improvements in performance of ICs. However, as silicon industry is moving toward the "end of Moore's Law," increases in static and dynamic variations, wear-out failures, and manufacturing defects are affecting the yield and reliability of ICs. 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. According to [4], the predicted probability of failure (pfail) for SRAM cells is equal to 2.6e-04 in 12nm technology.

As a result, many researchers turn their attention into the area providing various fault-tolerance cache schemes that operate either at the circuit or at the architectural level. A class of architectural level methods for masking out the memory faults relies on inserting spare elements (blocks or columns) in the cache array. Clearly, due to the limited redundancy that can be afforded, those solutions cannot scale to high failure rate situations [2]. The usage of error correcting codes (ECC) has also been proposed. However, the use of ECC is not practical for hard errors due to their large storage overheads and ECC repair time penalty rendering them not suitable for the latency-sensitive L1 caches. Finally, several fault tolerant schemes are based on the concept of disabling faulty cache portions, such as block or words, and reconfiguring operational ones (e.g., physical or logical neighborhood blocks) to act as a substitute of the disabled cache parts [5]. Obviously, those techniques require significant circuit modifications which may also pose a great burden to the designers of time-sensitive L1 caches.

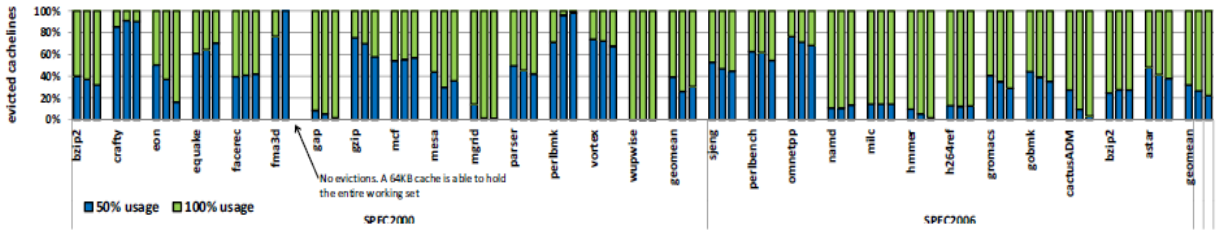

Fig. 1. Cache utilization breakdowns for various 2-way cache sizes (from left to right: 16KB, 32KB, and 64KB).

Motivation. In contrast to these approaches, our proposal targets to better utilize the cache fault-free area by exploiting the low cache block utilization. In particular, we show that striving to substitute every faulty subblock or word with a sound one is not always necessary, if the executing applications exhibit specific access patterns. Partially faulty cache frames can host cache blocks with limited spatial footprints with minimal or no impact in performance. Thus, if someone is able to predict the to-be-referenced words of a cache block and accordingly 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. *To the best of our knowledge, this is the first approach that proposes dedicated fault tolerant aware cache (re)placement policies.*

To quantify the potential of our approach, we analyzed the spatial footprints of several applications. Fig. 1 plots our results as stacked bars. We assume that the cache blocks are divided into two equally sized parts. 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. 1 indicates, different benchmarks exhibit different characteristics. 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.

2 Coarse Grain Spatial Footprint Predictor

Our target is to enhance faulty caches with a spatial pattern prediction mechanism that will provide the necessary information (unused portions of the blocks) to the underlying cache (re)placement logic. While many sophisticated spatial footprint predictors (SFP) are available [3], the proposed mechanisms are too fine grain and storage demanding for our purposes. The range of the required predictions in our case is very narrow (each cache frame is split into two equal divisions). As a result, we propose practical, instruction (PC) based, coarse grain SFP (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/right portion of a block will be requested by the core (3 decisions in total). The latter characteristic leads to *frugal and scalable prediction structures requiring 128 bytes total storage overheads (less than 0.3% in a 32KB cache).* A detailed analysis of the storage overheads of our proposal can be found in [1].

CG-SFP Structures. The key idea is to relate a cache block to the instruction (PC) that accesses the relevant block and issues a prediction according to the instruction previous behavior (Fig. 2). The predictor is composed by a PC History Table (PCHT) array that is

responsible for storing active PCs. A second array, indexed by the PCHT, is the Spatial Footprint History Table (SFHT) which holds the predicted spatial footprint information for the corresponding PC. The operation of CG-SFP is described in the rest of this section.

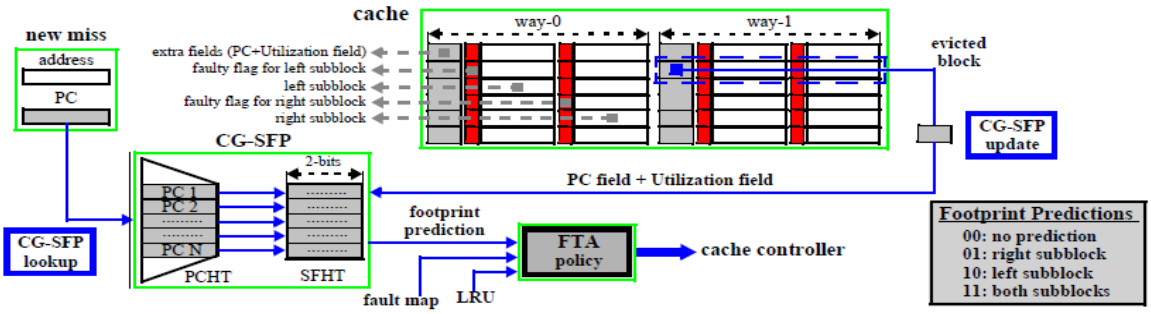

Fig. 2. The anatomy of CG-SFP predictor.

Lookup. Lookup is the operation that predicts the spatial footprints of the blocks touched by an instruction (PC): the PC of a 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 hit occurs, the PCHT selects the corresponding predicted spatial footprint information and feeds this information to the FTA module.

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. 2, 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.

Practical Implementation. Due to spare considerations, the detailed methodology to increase the effectiveness of CG-SFP (in terms of accuracy and coverage of the issued predictions), while keeping the storage requirements in a reasonable level (below 0.3% in a 32KB cache) can be found in [1].

```
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. 3. The proposed FTA policy.

3 Managing Faulty Caches

The proposed management methodology is performed in two levels. At the lower level, each faulty cache frame is augmented by 1-bit (*flip-bit*). If flip-bit is set, the sound cache physical part

(subframe) hosts subblocks that normally would be located at the defective subframe. If flip-bit is clear, the subblock resides in the correct physical location. flip-bit allows us to make better utilization of CG-SFP predictions, since our management 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 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. 2). 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 FTA policy is depicted in Fig. 3.

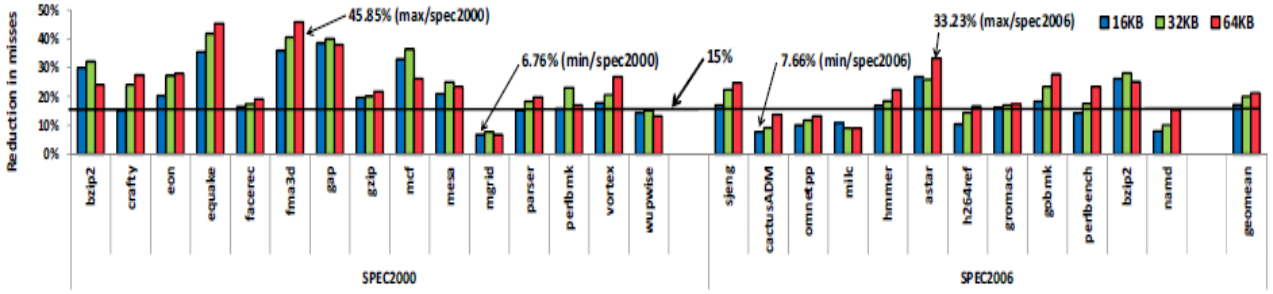

Fig. 4. Reduction of misses achieved in a FTA managed cache normalized to the misses of a conventional cache.

4 Results

To evaluate our FTA policy, we perform cycle accurate simulations using applications from SPEC2000 and SPEC2006 suites. More details about the simulation infrastructure can be found in [1]. In addition, we randomly produce 100 fault maps assuming a SRAM cell pfail equal to 1e-03 [4] and we assume that the cache disabling scheme is applied in a subblock granularity (a faulty bit is assigned in every 16 bytes). Fig. 4 shows the relative reduction in the number of misses achieved by our FTA normalized to the misses occurred in an unmanaged cache (in the latter case partially defective cache frames can also serve as (re)placement candidates). As indicated by the rightmost group of bars in Fig. 4 (average statistics for each studied cache size across all benchmarks), our proposal reduces the number of misses by 17.2%, 19.7%, and 21.2% in the 16KB, 32KB, and 64KB respectively.

5 References

[1] G. Keramidas, M. Mavropoulos et al. Spatial pattern prediction based management of faulty data caches. *DATE, 2014.*

[2] H. Lee, A. Cho et al. DEFCAM: A design and evaluation framework for defect-tolerant cache memories. *TACO*, 2011.

[3] P. Pujara and A. Aggarwal. Increasing cache capacity through word filtering. *ICS*, 2007.

[4] D. Sanchez, Y. Sazeides et al. An analytical model for the calculation of the expected miss ratio in faulty caches. *IOLT*, 2011.

[5] C. Wilkerson, H. Gao et al. Trading off cache capacity for reliability to enable low voltage operation. *ISCA*, 2008.