Journal of Parallel and Distributed Computing xxx (xxxx) xxx

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A fault-tolerant last level cache for CMPs operating at ultra-low voltage

Alexandra Ferrerón [a,c,1], Jesús Alastruey-Benedé [a,c,*], Darío Suárez Gracia [a,c], Teresa Monreal Arnal [b,c], Pablo Ibáñez Marín [a,c], Víctor Viñals Yúfera [a,c]

[a] Departamento de Informática e Ingeniería de Sistemas, Instituto de Investigación en Ingeniería de Aragón, Universidad de Zaragoza, Spain
[b] Universitat Politècnica de Catalunya, BarcelonaTech, Spain
[c] HiPEAC European Network of Excellence, Belgium

HIGHLIGHTS

- Fault-Tolerant Last Level Cache for CMPs Operating at Ultra-Low Voltage.
- Mechanism that exploits redundancy and reuse to enhance block disabling performance.
- Fault-aware LLC management that maps critical blocks to operative cache entries.
- Detailed evaluation of block disabling techniques in a shared-memory coherent CMP.

ARTICLE INFO

Article history:
Received 19 December 2017
Received in revised form 23 July 2018
Accepted 22 October 2018
Available online xxxx

Keywords:
Near-threshold voltage
SRAM reliability
Fault-tolerance
On-chip caches
Cache management

ABSTRACT

Voltage scaling to values near the threshold voltage is a promising technique to hold off the many-core power wall. However, as voltage decreases, some SRAM cells are unable to operate reliably and show a behavior consistent with a hard fault. Block disabling is a micro-architectural technique that allows low-voltage operation by deactivating faulty cache entries, at the expense of reducing the effective cache capacity. In the case of the last-level cache, this capacity reduction leads to an increase in off-chip memory accesses, diminishing the overall energy benefit of reducing the voltage supply. In this work, we exploit the reuse locality and the intrinsic redundancy of multi-level inclusive hierarchies to enhance the performance of block disabling with negligible cost. The proposed fault-aware last-level cache management policy maps critical blocks, those not present in private caches and with a higher probability of being reused, to active cache entries. Our evaluation shows that this fault-aware management results in up to 37.3 and 54.2% fewer misses per kilo instruction (MPKI) than block disabling for multiprogrammed and parallel workloads, respectively. This translates to performance enhancements of up to 13% and 34.6% for multiprogrammed and parallel workloads, respectively.

1. Introduction

For recent CMOS technologies, power density is the main performance limiting factor across most computing segments. Moore's law continues to hold, with a doubling of the number of transistors and integration density in each new process generation, but Dennard scaling no longer applies, and we are not able to keep a constant power density across technology generations. Power budgets prevent us from utilizing all the available transistors, leading to dark silicon [44].

For years, industry has relied on scaling the supply voltage (V_{dd}) to reduce power consumption, but this trend has dramatically slowed since the 90 nm generation because of leakage. Reducing operating voltages to values near the threshold voltage (V_{th}) would minimize leakage and switching power consumption. The resulting power reduction could be used to activate more chip resources and potentially achieve performance improvements [14].

Unfortunately, V_{dd} scaling is limited by the tight margins of the on-chip cache SRAM transistors. Excessive parameter variations in SRAM cells limit the voltage scaling of memory structures to a minimum voltage, $V_{dd_{min}}$, below which SRAM cells may not operate reliably. $V_{dd_{min}}$ usually determines the minimum voltage of the whole processor, and in current technologies is typically of the order of 0.7–1.0 V, when regular 6T SRAM cells are employed.

In the literature, various solutions have been proposed to enable reliable cache operation at low voltages. At the circuit level, the use

* Corresponding author at: Departamento de Informática e Ingeniería de Sistemas, Instituto de Investigación en Ingeniería de Aragón, Universidad de Zaragoza, Spain.
E-mail address: jalastru@unizar.es (J. Alastruey-Benedé).
[1] Now at Google.

https://doi.org/10.1016/j.jpdc.2018.10.010
0743-7315/

Please cite this article as: A. Ferrerón, J. Alastruey-Benedé, D. Suárez Gracia et al., A fault-tolerant last level cache for CMPs operating at ultra-low voltage, Journal of Parallel and Distributed Computing (2018), https://doi.org/10.1016/j.jpdc.2018.10.010.

of larger transistors or more transistors (assist circuitry) improves SRAM cell resilience [29,47]. The main drawbacks of this approach are the associated increases in area and power consumption. First-level caches in chip multiprocessors (CMPs) occupy little area, and their access time often determines the processor cycle time. Commercial processors, such as the Intel Nehalem family, use robust 8T SRAM cells to build reliable first-level caches, since this represents an affordable overhead [30]. In contrast, last-level caches (LLCs) are usually shared and have larger sizes and associativity, accounting for much of the die area [9]. Hence, for LLCs, minimum-geometry 6T cells are preferred to achieve higher densities.

At the architectural level, fault-tolerant cache designs rely on disabling faulty resources at different granularities [32], or correcting defective bits through either error correction codes (ECCs) [13] or a distributed duplication of blocks [5,45]. Block Disabling (BD) is a simple technique that disables a cache entry when a defective bit is found [42]. It is already implemented in modern processors to protect against hard faults [9]. However, due to the random distribution of defective cells, the capacity of the cache is rapidly compromised. Complex techniques based on ECCs or the combination of faulty resources are able to rescue more cache capacity, but incur large storage overheads and sometimes require complex remapping that penalizes the cache access latency.

In our work, we have developed a new approach to mitigate the impact of SRAM failures in LLCs due to parameter variations, based on BD but also relying on the underlying structures already present in CMPs. We identify a natural source of on-chip data redundancy that arises because of the replication of blocks in inclusive multi-level cache hierarchies and exploit this redundancy through a smart fault-aware cache management policy.

In this paper, we make the following contributions. First, we provide an evaluation of BD techniques in a shared-memory coherent CMP running parallel and multiprogrammed workloads with a complete and detailed memory model, exploring SRAM cells with different probabilities of failure. Second, we introduce a technique that keeps the tags of the LLC and, therefore, the tracking capabilities of the coherence directory operational. This way, a block not physically stored in the LLC can reside in the private level and be made available to other cores. As an alternative to main memory supply, we set up a cache-to-cache copy service to support code or data sharing (thread migration, operating system, or parallel workloads). Finally, we propose a fault-aware cache management policy that predicts the usefulness of a block based on its use pattern, and guides the allocation of blocks to faulty and non-faulty cache entries, adding no overhead to the original replacement policy.

Our fault-aware cache management policy is able to decrease the LLC misses per kilo instruction (MPKI) by up to 37.3%, with respect to BD, which translates to speedup improvements of 2 to 13% for multiprogrammed workloads. For parallel workloads, the MPKI values decrease by 5 to 54.2%, with respect to BD, for the different SRAM cells considered, improving performance up to 34.6%.

This paper extends our previous work [16] in several significant ways: (i) a new fault-aware cache management policy aiming at caches operating at low voltages, (ii) a detailed implementation of block disabling with operational tags (BDOT) technique, (iii) a more realistic SRAM fault model, improving the accuracy of the results, and (iv) a more detailed evaluation including multi-programmed workloads and cache capacity/energy analysis.

The rest of the paper is organized as follows. Section 2 introduces the problem of process variations and its effect on SRAM cell reliability. Section 3 comments on BD and its impact on large cache structures. In Section 4, we describe how to take advantage of the coherence infrastructure to operate at low V_{dd}. Section 5 introduces a fault-aware cache management policy for LLC operating at low voltages. Section 6 describes the methodology. Section 7 presents our evaluation. Section 8 discusses the system impact. In Section 9, we comment on related work, and in Section 10, we outline our conclusions.

Table 1
Area relative to cell C1 and percentage of non-faulty 64-byte entries in a cache operating at 0.5 V, for the 6 bit cells introduced in [47].

Cell type	C1	C2	C3	C4	C5	C6
Relative area	1.00	1.12	1.23	1.35	1.46	1.58
% non-faulty	0.0	9.9	27.8	35.8	50.6	59.9

2. Process variations in SRAM cells

SRAM structures are especially vulnerable to failures due to process variations, as they are aggressively sized to meet high density requirements, and because of the vast number of cells that comprise on-chip SRAM structures [7]. In particular, intra-die random dopant fluctuations (RDFs) are the main cause of threshold voltage variation [43]. The stochastic nature of the ion implantation process leads to a distribution of V_{th} values across a chip, which reduces the already tight transistor margins. Hence, SRAM structures have a minimum voltage, $V_{dd_{min}}$, to guarantee reliable operation, which is typically of the order of 0.7–1.0 V in current process generations, when 6T cells are used.

The robustness of SRAM cells under the $V_{dd_{min}}$ range has been extensively analyzed in the literature [5,13,29,31,45]. Zhou et al. studied six different sizes of 6T SRAM cells in 32 nm technology, and their probabilities of failure as V_{dd} decreases [47]. According to that study, at 0.5 V, the probability of failure of an SRAM cell (P_{fail}) is between 10^{-3} and 10^{-2}. The use of larger cells reduces the probability of failure, as non-uniformities average out, increasing read and write margins and resulting in more robust devices. However, large cells reduce the density and increase power and energy consumption.

Table 1 describes the six SRAM cells of Zhou's study (C1, C2, C3, C4, C5, and C6) in terms of their area relative to the smallest cell (C1), and lists the percentage of non-faulty entries in caches built from these cells operating at 0.5 V, assuming 64-byte cache entries. An entry is considered faulty if it contains at least one defective bit.

As Table 1 shows, less than 10% of the cache entries are non-faulty for the small cells C1 and C2 at 0.5 V. If the cache is implemented with the more robust C6 cells, however, the percentage of non-faulty cache entries rises to 60%, but at the cost of a 58% increase in area (relative to C1), and the consequent increase in leakage, which is not a suitable option for a large structure such as an on-chip LLC.

In this work, we take Zhou's reliability study as a reference to test our proposals on a wide range of failure probabilities. We will only consider C2 to C6 operating at 0.5 V (our target near-threshold V_{dd}), as at this voltage, a cache built with C1 cells would have all its capacity compromised.

3. Impact of block disabling on large shared caches at ultra-low voltages

A simple approach to handling hard faults is the disabling of faulty elements. BD deactivates resources at block (cache entry[2]) granularity: when a fault is detected at a given cache entry, that entry is marked as defective and it can no longer store a cache

[2] In this work, we differentiate between cache block and cache entry: block refers to the transfer unit, the content *per se*, while entry refers to the physical group of cells that store a block.

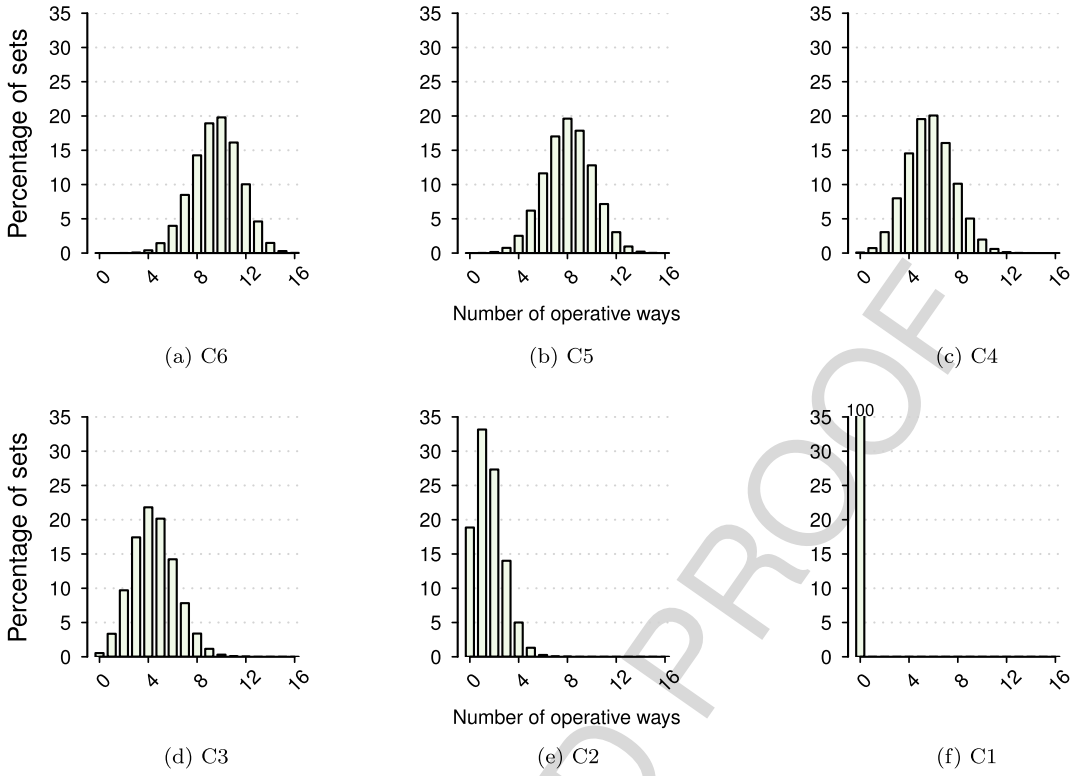


Fig. 1. Available associativity of a 16-way set associative block disabling cache (64-byte block) made up of cells C6–C1 operating at 0.5 V.

block [42]. This technique is implemented in modern processors to enable them to tolerate hard faults [9].

BD has also been studied for operation at low voltages because of its easy implementation and low overhead [31]. From the implementation perspective, only one bit per entry suffices to mark the entry as faulty. The main drawback of this approach is that the amount of capacity dramatically falls when the probability of failure increases, as shown in Table 1. Even if the total count of faulty cells in the cache is less than 1%, the effective cache capacity is strongly affected because of the random distribution of faulty cells. BD results in caches with variable associativity per set, determined by the number and distribution of faults in the cache.

The interaction between BD and a system's cache organization also plays an important role. Modern commercial processors, such as the Intel Core i7, implement inclusive hierarchies to facilitate coherence management. Inclusive hierarchies require that all the blocks cached in a private cache are also stored in the shared LLC. The coherence information is embedded in the LLC; i.e., the sharing state and a bit vector to represent the current sharers are added to each block. To force inclusion, when a block is evicted from the LLC, explicit back invalidations are required to remove the copies of the private cache blocks, if present (inclusion victims) [6].

Inclusive hierarchies perform poorly when the aggregated size of the private caches is similar to the size of the LLC [23], and BD exacerbates the problem because of the substantial associativity and capacity degradation in the LLC. Fig. 1 shows the available associativity in a 16-way set associative cache bank with 64-byte blocks, when built with cells C1–C6 (Table 1) operating at 0.5 V. The number of faulty ways per set follows a binomial distribution $B(n, p)$, where n is the associativity, and p denotes the probability of failure of a cache entry. Fig. 1 shows how the associativity degrades as more faulty cells appear on the cache structure. On average, 50% of the ways are faulty if the cache is built with C5 cells, and this percentage rises to 90% when using C2 cells. The associativity loss directly translates to a significant increase in the number of inclusion victims. For instance, the number of invalidations in a cache built with C3 cells is 10 times larger than in a cache implemented with fault-free SRAM cells.

This finding suggests that inclusive hierarchies are not particularly suitable for systems that implement BD in the presence of a significant number of faults. From the coherence management perspective, however, only directory inclusion is required: blocks present in the private levels have to be tracked only in the shared level tag array, without the need for a replica in the data array [6]. This observation is the basis for the techniques we propose in this paper.

Our proposal has been designed for inclusive memory hierarchies, but most of the proposed ideas could benefit non-inclusive and exclusive hierarchies as well. The objectives of our replacement and promotion algorithms are to assign the non-faulty entries to blocks with reuse and to blocks that are not present in the private caches. On the one hand, these objectives are still valid in a non-inclusive hierarchy; however, their relative importance is different, and our algorithms should consider different priorities for allocation and promotion decisions. On the other hand, our proposal alleviates a specific problem of inclusive hierarchies, such as the need to invalidate a block in a private cache when it is evicted from the shared cache. This problem does not exist in non-inclusive hierarchies, and therefore our proposal is not applicable in this specific aspect.

Note on Fig. 1 that, when using cell types C3 and C2, 0.6% and 18.9% of the sets have no operative ways, respectively. To be able to offer a complete comparison with BD, we assume that at least one of the ways in each set is non-faulty, although this is not a requirement for the techniques we present in this paper, and the LLC is able to operate even when all the ways of a set are faulty.

4. Exploiting inclusive hierarchies to enable ultra-low voltage operation: Block Disabling with Operational Tags

The BD scheme simply assumes one extra bit per entry to identify faulty cache entries in the data array (one or more faulty

Table 2
Main characteristics of the CMP system.

Cores	8, Ultrasparc III Plus, in-order, 1 instr./cycle, single-threaded, 1 GHz at V_{dd} 0.5 V
Coherence protocol	MESI, directory-based (full-map) distributed among LLC cache banks
Consistency model	Sequential
L1 cache	Private, 64 KB data and instr. caches, 4-way, 64 B block size, LRU, 2-cycle hit access time
LLC cache	Shared, inclusive, interleaved by line address, 1 bank/tile, 1 MB/bank, 16-way, 64 B block size, NRU replacement 8-cycle hit access time (4-cycle tag access + 4-cycle data access)
Memory	2 memory controllers, located at the edges of the chip; 1333 MHz DDR3 2 channels, 8 Gb/channel, 8 banks, 8 KB page size, open page policy; raw access time 50 cycles
NoC	Mesh, 2 virtual networks (VNs): requests and replies; 2 virtual channels per VN; 16-byte flit size 1-cycle latency hop, 2-stage routing

cells). Faulty data entries are excluded from tag search and replacement, involving a net reduction in associativity, and a consequent increase in inclusion victims. From the coherence management perspective, however, tracking blocks in the shared level tag array suffices to ensure directory inclusion. This is the basis of our previous work, [16], and the starting point of the first technique we propose: *Block Disabling with Operational Tags* (BDOT).

Assuming a two-level inclusive hierarchy, to force directory inclusion, we turn on the tags of faulty entries in the LLC, including them in the conventional operations of search and replacement. The tag of a faulty entry, if valid, tracks a cache block that might be present in the private caches, but that cannot be stored in the shared cache. Enabling the tags of the faulty entries restores the associativity of the shared cache as seen by the first-level private caches, eliminating the problem of the increase in the number of inclusion victims caused by the loss in associativity.

In this situation, two kinds of LLC entries have to be distinguished: tag-only (T), where the associated data entry is faulty and only the tag is stored, and tag-data (D), where the associated data entry is non-faulty and both tag and data are stored. From the implementation perspective, one resilient bit still suffices to indicate whether the entry is faulty or not. The coherence protocol needs to be adapted to this new situation, where a T entry only stores the block tag and directory state. Whenever a request to a block stored in a T entry arrives to the LLC bank, the request needs to be sent to the next level (in this case, the off-chip memory) to recover the block, and the same occurs with dirty blocks, which need to be written back to memory after being evicted from a private cache.

To fully exploit this scheme, no failures should occur in the cells of the tag array. This can be accomplished, for example, by using robust cells (e.g., increasing the number of transistors per cell) or increasing the strength of the ECC. Tags occupy very little area in comparison to the data array (around 6% for our configuration, see Table 2 in Section 6), and increasing the cell size by 33% (assuming 8T SRAM cells [10]) will only increase the total area of a cache bank by 2%. Since using sophisticated ECCs could increase the access latency of the tag array, while using resilient tag cells involves little overhead, we opt for the latter. This approach is also consistent with prior work [5,45]. Moreover, many of today's CPUs use different cell types for tag and data arrays [26]. Contrary to other proposals, our mechanism works even when all entries of a set are faulty. Contrary to other proposals, our mechanism works even when all entries of a set are faulty. The LLC saves the tags for both faulty and non-faulty entries, maintaining the coherence status of all the blocks, and allowing blocks to be stored in the private levels without the need of a data replica in the shared level. Hence, it is possible to store a block in the private caches even if all the data ways of the corresponding LLC set are faulty.

4.1. BDOT limitations

BDOT, as described above, has two potential limitations, both related to the allocation of blocks to faulty entries.

First, BDOT always forwards requests to blocks allocated to faulty entries to the off-chip memory. However, a block allocated to a faulty entry might be present on-chip, if it is being used by a private cache (L1). This situation is common in parallel workloads, which share data and instructions. In this case, the directory information can be used to orchestrate cooperation among L1 caches. When the directory protocol receives an L1 request to a shared block mapped to a T entry, it forwards the request to one of the sharers of the block, namely, the L1 cache closest to the requester in terms of Manhattan distance. That L1 will serve the block through a cache-to-cache transfer.

Cache-to-cache transfers are already implemented in the baseline coherence protocol for exclusively owned blocks. Hence, no additional hardware is required and a slight modification of the directory protocol suffices to trigger a shared block transfer. So from now on, we assume that BDOT includes this feature.

The second limitation comes from allocating blocks to LLC entries without taking into account their T or D nature. Unfortunately, this blind allocation can result in heavily reused blocks being attached to faulty entries. Indeed, if a particular block of the LLC is required repeatedly from an L1 cache (i.e., the block shows reuse), any replacement algorithm will tend to protect it, reducing its eviction chances. Thus, if a block with reuse is initially allocated to a T entry, unless replicated in other cores, all L1 cache misses will be forwarded off-chip by the LLC.

In the next section, we introduce a specific allocation and reallocation policy for BDOT caches that differentiates between T and D entries.

5. Fault-aware cache management policy for BDOT caches

Conventional cache management policies assume that every cache entry can store a block, while BDOT breaks this assumption: each set in an N-way set associative cache contains T entries that store only tags, and D entries that store tags and data. Keeping in mind the main goal of improving the overall LLC performance under BDOT, this section introduces a fault-aware cache management policy that takes into account the distinct nature of T and D entries, and the reuse pattern of the reference stream. In particular, we seek to achieve the following two goals:

1. To allocate blocks that are most likely to be used in the future to D entries.
2. To maximize the amount of on-chip data by giving greater priority (higher chances of being allocated to D entries) to blocks that are not present in private cache levels.

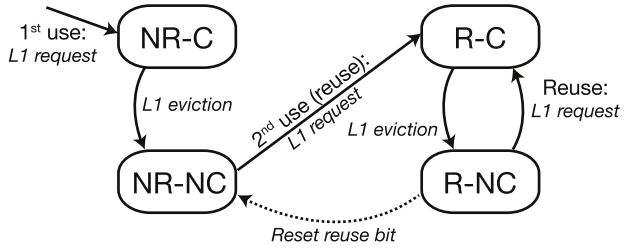


Fig. 2. Reuse and inclusion states for a block in LLC. NR, R, C, and NC represent: Non-Reused, Reused, Cached (in L1), and Non-Cached (in L1), respectively. Replacement and coherence transitions are not shown.

Prior work has shown that *reuse* is a very effective predictor of the usefulness of a given block in the LLC [4,11]. Reuse locality can be described as follows: lines accessed at least twice tend to be reused many times in the near future, and recently reused lines are more useful than those reused earlier [4]. Therefore, seeking to achieve our first goal, we exploit reuse locality to predict which blocks should be allocated to *D* entries. With respect to our second goal, a request to a block allocated to a *T* entry and present in L1 can be serviced through a cache-to-cache transaction, whilst if the block is not present in L1, the request will always be forwarded to the off-chip memory, incurring a penalty in access time and energy. Therefore, it is preferable to dedicate *D* entries to blocks not available on the L1 caches.

These goals may be added to any management policy. In this work, we will build on top of a state-of-the-art reuse-based replacement algorithm: Not-Recently Reused (NRR) [4]. Next, we describe the baseline replacement in some depth and then we add awareness of the existence of faulty entries.

5.1. Baseline NRR replacement algorithm

The NRR algorithm requires four states per LLC block, as depicted in Fig. 2. When a block not present in the LLC is requested by the processor (*1st use: L1 request*), it is stored in the L1 and the LLC (to force inclusion), its state being in the LLC `NR-C` (Non-Reused, Cached). When the block is evicted from the private cache (*L1 eviction*), its LLC state changes to `NR-NC` (Non-Reused, Non-Cached). On a new request (*2nd use: L1 request*), a copy of the block is stored again in L1, and its LLC state is `R-C` (Reused, Cached). At this point, the block has shown reuse in the LLC and, very likely, it will be reused many times in the near future. Finally, when the block is evicted again from the L1, the state becomes `R-NC` (Reused, Non-Cached). Subsequent requests and evictions switch between the `R-NC` and `R-C` states.

Having LLC blocks classified this way, the replacement policy can exploit L1 temporal locality and LLC reuse. In an inclusive hierarchy, the replacement of a block in the LLC forces the invalidation of its copies in the private caches, if any, and this usually implies performance degradation, assuming that blocks in L1 are being actively used [23]. Therefore, the highest priority (protection) is given to blocks stored in private caches. As a secondary objective, the highest priority is given to blocks that have shown reuse in the LLC. Hence, NRR selects victims in the following order: `NR-NC`, `R-NC`, `NR-C`, `R-C`. Reuse recency is taken into account by resetting the reuse bit when all the non-cached blocks are marked as reused (transition from `R-NC` to `NR-NC`). This way, more recently reused blocks become more protected.

The implementation of NRR only requires one *reuse bit* per block. The protection of private copies can be implemented in various different ways [23], but one simple solution is to use the presence bit-vector of the coherence directory, assuming non-silent tag evictions of clean blocks.

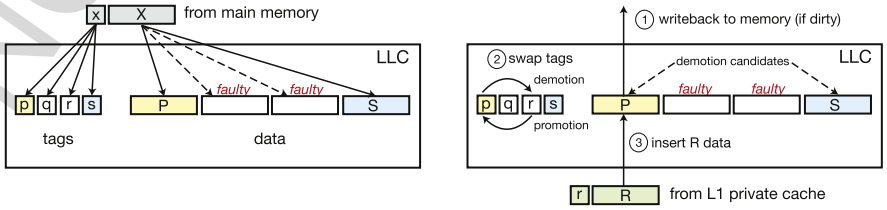
5.2. Reused-based and fault-aware management for BDOT caches

Seeking to guarantee that valuable blocks remain in the LLC, we devise a fault-aware management policy by distinguishing between *T* and *D* entries. One option is to *promote* blocks by reallocating them from *T* to *D* entries, if needed, to improve the overall cache performance. The design choices include *where* the promoted data comes from and *which* victim is chosen as a target of the consequent *demotion*. At the same time, we want to continue exploiting reuse in the simple and efficient way offered by an NRR-like replacement algorithm, which is unaware of faulty entries. Thus, our goal is to design a comprehensive cache management policy, merging reuse exploitation and faulty entry management. Below, we elaborate on the two mechanisms that are key to achieving this, namely block insertion/replacement and block promotion/demotion.

5.2.1. Insertion and replacement of blocks

On a first insertion (LLC miss), an incoming block has not shown reuse, and hence allocating it to a *T* entry seems a reasonable idea. Fig. 3(a) shows an example of a cache block to be inserted in a 4-way cache set with two *T* entries (those storing `q` and `r` tags) and two *D* entries (those storing `p` and `s` tags and the corresponding `P` and `S` data). A victim is selected among the blocks allocated to *T* entries. The baseline replacement policy dictates which of those blocks (`Q` and `R`) is selected for replacement. This is equivalent to predicting that the incoming block `X` is not going to be reused. If the reuse pattern of the block is mispredicted, block `X` should be reallocated to a *D* entry, to reduce its access time and transfer energy in future L1 misses. This reallocation will be performed using the promotion mechanism we detail in the next subsection.

Dealing with first insertions this way is very simple but has a clear disadvantage, related to the distribution of *T* and *D* entries, with respect to the percentage of reused and non-reused blocks. For example, if the number of *T* entries is small, the insertion policy would place considerable pressure on these scarce entries.



(a) Insertion from main memory after LLC miss.

(b) Insertion from L1 after L1 eviction (promotion and demotion).

Fig. 3. Insertion and promotion actions for a fault-aware cache management policy example in a cache set with two faulty cache entries. Lowercase and capital letters indicate tag and data, respectively.

Blocks would be unavoidably forced to leave the LLC before having had enough time to show a reuse pattern, even though there are many available *D* entries. In an extreme case, when all the entries in a set are *D* type, this cache management policy could not be implemented. Solving this problem is not easy. We explored various adaptive mechanisms in which some *D* entries are used as *T*. However, it is difficult to determine the optimal number of *T* entries, this being highly dependent on the workload. After carrying out several experiments (data not shown in Section 7, for the sake of brevity), the performance returns were disappointing given the required complexity.

Given that our promotion mechanism reallocates reused blocks to *D* entries and non-reused ones to *T* entries (as we will see in the following subsection), we realized that the baseline NRR replacement policy itself suffices to achieve our initial goals because it protects reused blocks. Since NRR gives lower priority to non-reused blocks, blocks allocated to *T* entries will have more chances to be evicted. This implies that, with a balanced distribution between *T* and *D* entries, an incoming block will have a higher probability of being inserted in a *T* entry than in a *D* entry. If the number of T entries in a set is very low, and even if there are no *T* entries in a set, the mechanism still works correctly. NRR periodically resets the reuse bit of those blocks not present in private caches, so some *D* entries become replacement candidates with the same priority as T entries. Hence, the initial insertion does not necessarily have to consider the nature of the entry, and our implementation relies only on the baseline replacement policy to select the victim block.

5.2.2. Promotion and demotion of blocks

A blind allocation of blocks to cache entries may result in valuable blocks (i.e., those with reuse) being initially allocated to *T* entries, and vice versa. However, this undesirable situation can be tracked on the fly through the reuse footprint, and reversed by swapping a *T* entry with a *D* entry: when a block allocated to a *T* entry shows reuse, we will *promote* it to a *D* entry. Promotion involves a complementary *demotion* of the block stored in the selected *D* entry.

To select which block is demoted, we also rely on reuse and L1 presence information. Reused blocks should be kept in the LLC, but unlike in the baseline replacement policy, *block demotion does not involve an LLC tag eviction*. Furthermore, if the block is present in L1, losing the contents of the LLC is not critical, because there is at least one on-chip copy of the block, which can be supplied by a cache-to-cache transaction. Thus, to maximize the amount of on-chip data, the demotion algorithm will select the victim block among those present in L1. Among the blocks in L1, non-reused ones should have more chances of being demoted.

Note that the promotion of a block can be performed at two different times: at reuse detection (i.e., on a second L1 request to a block stored in a *T* entry) or after the second eviction from L1 (i.e., on eviction after reuse). Performing the promotion after the second request from L1 duplicates the content, as a copy of the block is also stored in a private cache, whilst performing the promotion after the L1 eviction meets the goal of maximizing the amount of on-chip data. Therefore, we opt for the latter and trigger promotions only after L1 evictions, non-silent block data evictions being necessary.

The promotion/demotion process is illustrated in Fig. 3(b). When block R, which is stored in a *T* entry, is evicted from the L1 cache and selected for promotion (i.e., its reuse bit is set), we select a victim among the demotion candidates (P and S in Fig. 3(b)). Once the victim is selected (P in our example), we swap the cache contents in three steps: ① discard the data entry P, writing back the data to memory, if dirty; ② swap p and r tags; and ③ copy the data (R) to the available *D* entry, which was occupied by the demoted block (P).

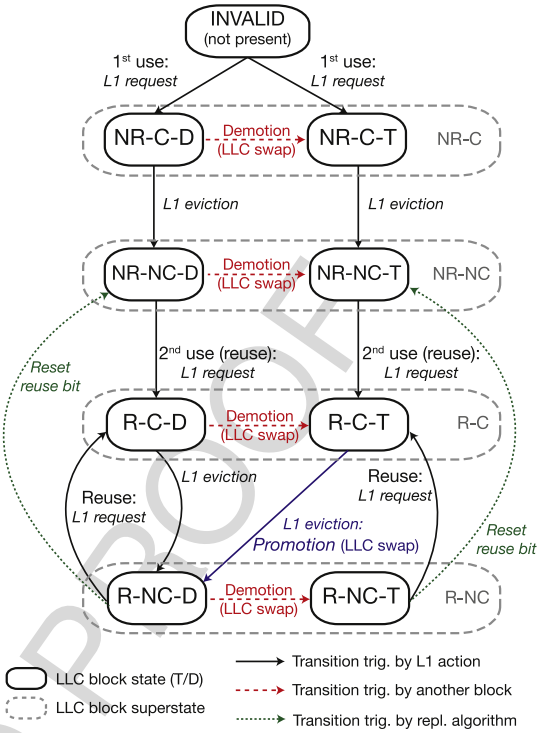


Fig. 4. Reuse and inclusion states for a block in LLC with BDOT.

5.2.3. Summary and implementation

Fig. 4 illustrates the implementation of the aforementioned ideas. The states of the baseline replacement algorithm shown in Fig. 2 are now *superstates* split into *T* and *D* states. The initial allocation of blocks (*1st use: L1 request* in Fig. 4) does not take into account the nature of the entry, and it solely depends on the victim selection arising from reuse and L1 presence; i.e., it only depends on the baseline replacement algorithm. After insertion, blocks will move along NR−C, NR−NC, R−C, and R−NC superstates as they would do in a cache without considering faulty entries.

To guarantee that high value blocks – those showing reuse – remain in the LLC, the policy reallocates them from *T* to *D* entries when they are evicted from the L1 and reside in a faulty LLC entry: R−C−T state. After *L1 eviction*, blocks in R−C−T trigger a promotion, which results in the transition to an R−NC−D state and reallocation to a *D* entry, with the consequent demotion of another block within the set to a *T* entry. A block being demoted can be in any of the superstates, and according to the victim selection algorithm, we first demote blocks that are present in the private levels, in order to maximize the amount of data available in the on-chip hierarchy. As a secondary objective, the policy attempts to first demote low priority blocks, that is, those without reuse. In particular, it selects blocks in the following order: NR−C−D, R−C−D, NR−NC−D, and R−NC−D.

This reuse-based, fault-aware policy adds no extra storage overhead to the baseline reuse-based replacement policy, as only the bit indicating reuse and the presence bit vector are needed to orchestrate the replacement and promotion decisions. Moreover, swapping blocks only requires some extra control logic to perform the following actions: first, the logic reads the demoted victim and inserts the promoted block, as for conventional block insertion, and, then, it writes back the tag of the demoted block. Promoting blocks after L1 eviction implies non-silent eviction of data blocks. This overhead does not affect latency, as L1 replacements are not in the critical path, and has a negligible impact on energy consumption.

Please cite this article as: A. Ferrerón, J. Alastruey-Benedé, D. Suárez Gracia et al., A fault-tolerant last level cache for CMPs operating at ultra-low voltage, Journal of Parallel and Distributed Computing (2018), https://doi.org/10.1016/j.jpdc.2018.10.010.

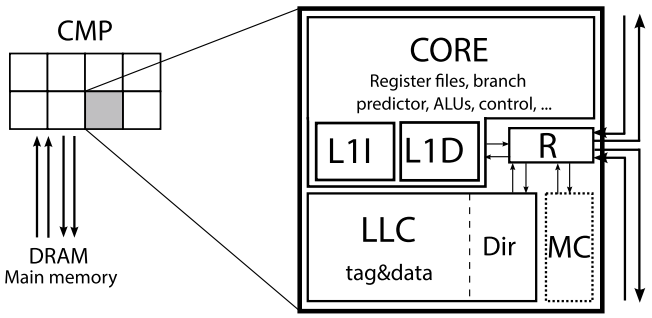


Fig. 5. Modeled 8-core CMP.

The fault-aware cache management technique here presented could be implemented on top of other replacement algorithms (such as LRU or NRU). We decided to rely on NRR because of its simple, yet efficient implementation, and because it fits the general principles behind our ideas. Finally, and regarding the reallocation from T to D entries and vice versa, other policies are also possible. For example, instead of relying on the reuse information of the blocks, a future use predictor [46] could be utilized to decide which blocks should be allocated to D entries, or a dead block predictor [27] could be used to indicate which blocks may be demoted to T entries, but these solutions add complexity to the cache logic as well as requiring more storage overhead.

6. Methodology

6.1. Overview of the system

Our baseline system consists of a tiled CMP, with an inclusive two-level cache hierarchy, where the second level cache or LLC is shared and distributed among the processor cores. Tiles are interconnected by means of a mesh. Each tile has a processor core with a private first level cache (L1) split into instructions and data, and a bank of the shared LLC, both connected to the router (Fig. 5). Similarly to most CMP, the write-policy for L1 data caches is write-back because other policies, such as write-through, may collapse the interconnection network [19]. The mesh will have to convoy every single store from the cores to the LLC banks to guarantee content inclusion. The CMP includes two memory controllers located at the edges of the chip. Table 2 shows the parameters of the baseline processor, memory hierarchy, and interconnection network.

We assume it runs at a frequency of 1 GHz with an operating voltage of 0.5 V. Note that the DRAM module voltage is not scaled like the rest of the system, and hence, the relative speed of main memory with respect to the chip increases as the voltage decreases. This model is consistent with prior work [5,45].

Our baseline coherence protocol relies on a full-map directory with Modified, Exclusive, Shared, Invalid (MESI) states. We use explicit eviction notification of both shared and exclusively owned blocks. L1 caches are built with robust SRAM cells that can run reliably at low or near-threshold voltages, while LLC data banks are built with conventional 6T SRAM cells and, therefore, they are sensitive to failures [30].

As in previous studies [5,45], we assume that the LLC tag arrays are hardened by using upsized cells such as 8T [10]. The baseline LLC replacement policy is Not-Recently Used (NRU) [37] extended with private copy protection [23]. We implement this protection by using coherence directory information updated by non-silent L1 block evictions.

6.2. Experimental set-up

Regarding our experimental set-up, we model the CMP system described in Table 2. We use Simics [34] in combination with GEMS [36] to simulate the on-chip memory hierarchy and interconnection network, and DRAMSim2 [40] to simulate the DDR3 DRAM in detail. To obtain timing, area, and energy consumption, we use the McPAT framework [33] for the on-chip components, and DRAMSim2 for the DRAM module. We extend the Ruby module (GEMS) to simulate the cache swaps in detail in order to take into account their dynamic energy overhead.

We use a set of 20 multiprogrammed workloads built as random combinations of the 29 SPEC CPU 2006 applications [21], with no special distinction between integer and floating point programs. Each application appears on average 5.5 times with a standard deviation of 2.5. Programs were run on a real machine until completion with the reference inputs. Hardware counters were used to locate the end of the initialization phase. Every multiprogrammed mix was run for as many instructions as the longest initialization phase, and a checkpoint was created at this point. We then run cycle-accurate simulations including 300 million cycles to warm up the memory hierarchy and 700 million cycles for data collection.

We also include a selection of shared-memory parallel applications from PARSEC [8] with a significant memory footprint ($MPKI_{LLC} \geq 1.0$) when running the *sim-large* input in the baseline system: `canneal` ($MPKI_{LLC} = 4.3$), `ferret` (1.6), `streamcluster` (1.0), and `vips` (1.2). We proceed in a similar way to that used for multiprogrammed workloads[3] and run 300 million cycles to warm up the memory structures once the parallel phase has started, and then collect statistics for 700 million cycles.

One challenge for analyzing fault mitigation techniques is the large set of required simulations. Running all workloads and simulated models combinations for a single fault map can lead to wrong results, as other authors have described [20,41]. For example, if all the faults affect to the most/least frequently accessed cache sets, the observed speed-up would be much lower/higher than in reality.

To address this issue, we rely on statistical sampling to generate random fault maps and run Monte Carlo experiments to guarantee a 5% margin of error with a confidence level of 95% [22]. In other words, the number of samples is increased as necessary to reach the target margin of error within the desired level of confidence. For our workloads, simulated models, metrics, margin of error and confidence level, each point of the design space has to be simulated between 20 and 30 times, each one with a different fault map. We pick the 5% margin of error and the 95% confidence level as a good trade-off between simulation time and accuracy, increasing both has a large impact in the required number of simulations. To ensure all simulations have similar numbers of faults but at different locations, we compute the faultiness of each memory cell randomly and independently of other cells [2,12]. Finally, we consider that the number and location of faulty cells do not change during workload execution.

7. Evaluation

This section evaluates the effectiveness of the proposed BDOT management technique for LLC caches in terms of MPKI, adding up the misses in all LLC banks and dividing by the aggregated instruction count of all cores. Later, in Section 8, we analyze the impact on system performance, area, and energy.

[3] We observed that no OS activity appeared when our parallel applications were run and the ratio of CPU utilization between the different threads was practically constant across the simulations.

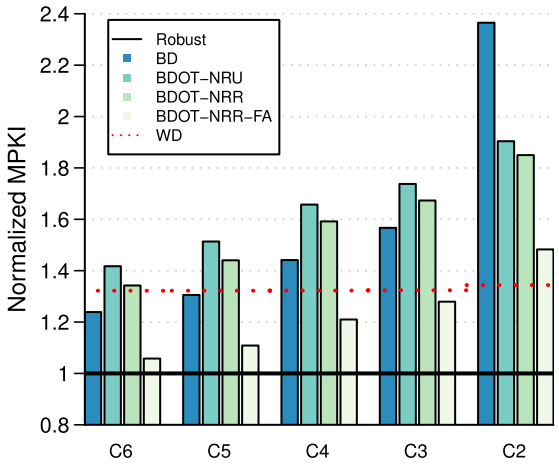


Fig. 6. Normalized MPKI (average for SPEC mixes) with respect to Robust for the different proposals and cell types. Average MPKI for Robust: 5.09.

To assess the effectiveness of our proposals, we include several additional configurations. First, as an upper bound in performance, a robust cache built with *unrealistically robust cells* (Robust); i.e., cells that operate at ultra-low voltages with neither failures nor power or area overheads, which corresponds to a perfect unattainable solution. Then, we also include *block disabling* (BD), as our proposal emerges from it. Finally, we add results for *word disabling* (WD) [45]. Word disabling is a more complex technique that combines consecutive faulty cache entries to recreate fully functional ones, at the cost of reducing the cache capacity. Section 9 presents a comprehensive discussion of this and other techniques versus our proposals.

In summary, we consider the following configurations:

- **Robust**: reference system; the LLC is built with unrealistically robust cells. All data are presented with respect to this system.
- **BD**: system implementing block disabling, as presented in Section 3, with NRU replacement.
- **BDOT-NRU**: system implementing block disabling with operational tags, as presented in Section 4, with NRU replacement.
- **BDOT-NRR**: system implementing BDOT with NRR replacement, as presented in Section 5.1.
- **BDOT-NRR-FA**: system implementing BDOT with fault-aware NRR replacement, as presented in Section 5.2.
- **WD**: system implementing word disabling with NRU replacement [45].

As in the case of NRR, the NRU implementation also includes private copy protection. Our detailed results include multiprogrammed workloads (the 20 SPEC CPU 2006 mixes) and parallel workloads (the 4 selected PARSEC applications), for the five cell types considered (C6, C5, C4, C3, and C2).

7.1. Multiprogrammed workloads

Fig. 6 shows the LLC MPKI results for the multiprogrammed workloads.

BD is a valid solution for a cache with few defective entries, like one built with C6 cells, where the average MPKI penalization is 23.9%. However, this penalization increases rapidly with the number of faulty entries, reaching 136% for C2. Using the tags of the defective LLC entries to keep the coherence state of blocks stored in L1 allows BDOT-NRU to incur fewer MPKI than BD for C2, but it does not offer any advantage (the MPKI value increases) for the rest of the cells.

To differentiate and quantify the benefit of a reuse-based replacement and our fault-aware cache management policy, we first implement NRR on top of BDOT (BDOT-NRR), without taking into account the nature of cache entries (faulty or non-faulty). This naive implementation offers a slight improvement with respect to BDOT-NRU for all cell types, but it is still worse than BD, except for C2, as in the case of BDOT-NRU. The explanation for this behavior is the blind allocation of blocks to entries, without taking into account whether the entry can store only the tag (T) or both the tag and the data (D). Allocating a block that shows reuse to a T entry implies that all the requests to that block are forwarded to the next level (in this case, off-chip). Besides, due to the reused-based policy, this block will remain in the defective entry of the LLC, protected by the replacement algorithm. However, blocks with reuse allocated to D entries are also protected from replacement, and that explains why the relative differences between BDOT-NRR and BDOT-NRU are larger when using larger cells (i.e., with less faults, like C6 and C5).

BDOT-NRR-FA addresses this issue, adding the information of defective entries to the cache management policy. The penalization in terms of MPKI is 14.6%, 15.1%, 16%, 18.3%, and 37.3% lower than with BD for C6, C5, C4, C3, and C2, respectively. If we compare BDOT-NRR-FA with BDOT-NRR, there are 20% fewer MPKI, irrespective of the cell type, demonstrating the goodness of the design.

Regarding WD, although there are significant differences in terms of the number of defective entries among the cell types considered (Table 1), the MPKI for the different configurations is almost constant. Two reasons explain this behavior: (i) a single defective cell forces the entry to be classified as faulty, and (ii) the number of defective cells per entry is usually small (three on average for the smallest cell: C2 [15]) and, therefore, very often blocks are successfully stored by combining two consecutive entries. Thus, the average number of ways per set in our system when implementing WD is eight across the different cell configurations. Compared to BD, WD obtains better results when the average number of defective entries is greater than half, which is the case of cells C4, C3, and C2, as shown in Table 1. BDOT-NRR-FA lowers the MPKI with respect to WD by 20%, 16.1%, 8.5%, and 3.4%, for C6, C5, C4, and C3, respectively. WD only beats BDOT-NRR-FA in caches with a high number of defective cells (C2, where on average 90% of the entries are faulty). However, BDOT-NRR-FA requires no additional overhead, whilst WD requires additional storage and logic to reconstruct blocks.

7.2. Parallel workloads

Fig. 7 shows the relative LLC MPKI for the parallel workloads, with respect to the baseline. As with multiprogrammed workloads, BDOT-NRR-FA has a lower average MPKI than BD and non fault-aware implementations of BDOT. In particular, BDOT-NRR-FA improves MPKI with respect to BD by 5%, 5%, 9.6%, 19.2%, and 54.2% on average for C6, C5, C4, C3, and C2, respectively. Comparing with the multiprogrammed workloads, the relative MPKI numbers shown in Fig. 7 are larger, moving away from the Robust system to a greater extent for all cell types, even for the winning alternatives (WD and BDOT-NRR-FA). But it is worth noting that the absolute MPKI values for the parallel applications considered are low (Section 6), which makes the relative increases appear more substantial.

Upon closer examination of the results, we can make some interesting observations. Fig. 8 shows the LLC MPKI analysis per application for the different cell types. BD is better than plain BDOTs (BDOT-NRU, BDOT-NRR) in C6–C3 cells (C3 in `canneal` is an exception), while in cell C2 the trend clearly reverses. On the contrary, BDOT-NRR-FA is better than BD in most cases, being `vips` the only exception (cells C6–C3), and giving very noticeable reductions in the smallest cell C2. For `vips`, BDOT-NRR-FA only

Please cite this article as: A. Ferrerón, J. Alastruey-Benedé, D. Suárez Gracia et al., A fault-tolerant last level cache for CMPs operating at ultra-low voltage, Journal of Parallel and Distributed Computing (2018), https://doi.org/10.1016/j.jpdc.2018.10.010.

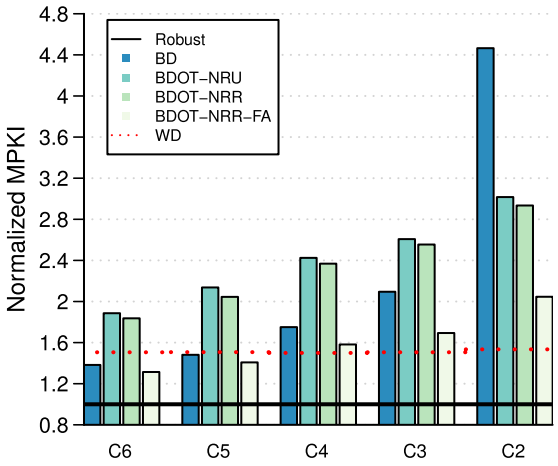


Fig. 7. Normalized MPKI (average for PARSEC) with respect to Robust for the different proposals and cell types. Average MPKI for Robust: 2.01.

beats BD in C2 because its image processing algorithm shows very little reuse with a small working set. In such non-demanding environment, BD can store the `vips` working set.

Finally, the costly WD shows a similar tendency to that observed with multiprogrammed workloads, with a relatively constant performance independently of the cell type. In this case, BDOT-NRR-FA beats WD when using C6 or C5 (12.7% and 6.6% lower MPKI values, respectively), but it cannot reach WD performance for C4, C3, or C2 (5.5%, 12.4%, and 33.3% higher MPKI values, respectively).

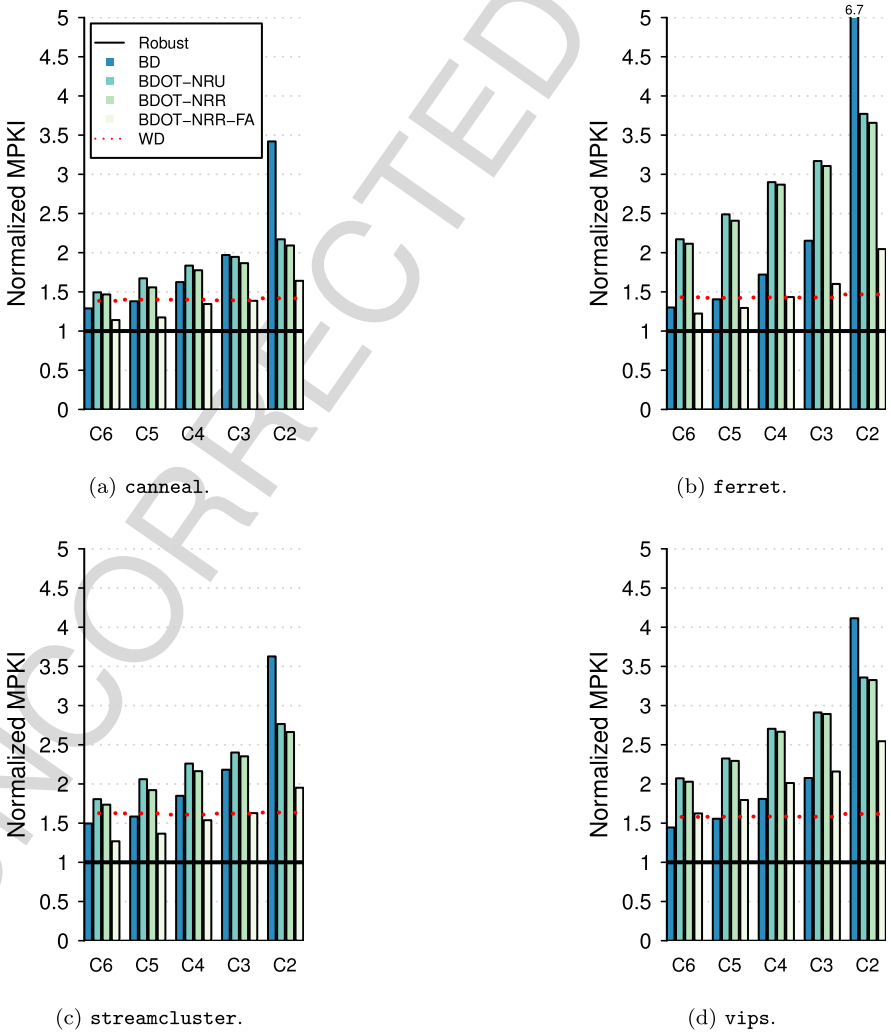
8. System impact

This section analyzes the impact of our proposals on the system in terms of performance, area, and energy consumption. As in the previous section, we present results relative to the Robust system and compare with the BD and WD mechanisms.

8.1. Performance

Fig. 9 shows the performance relative to the robust cell for both multiprogrammed and parallel workloads.

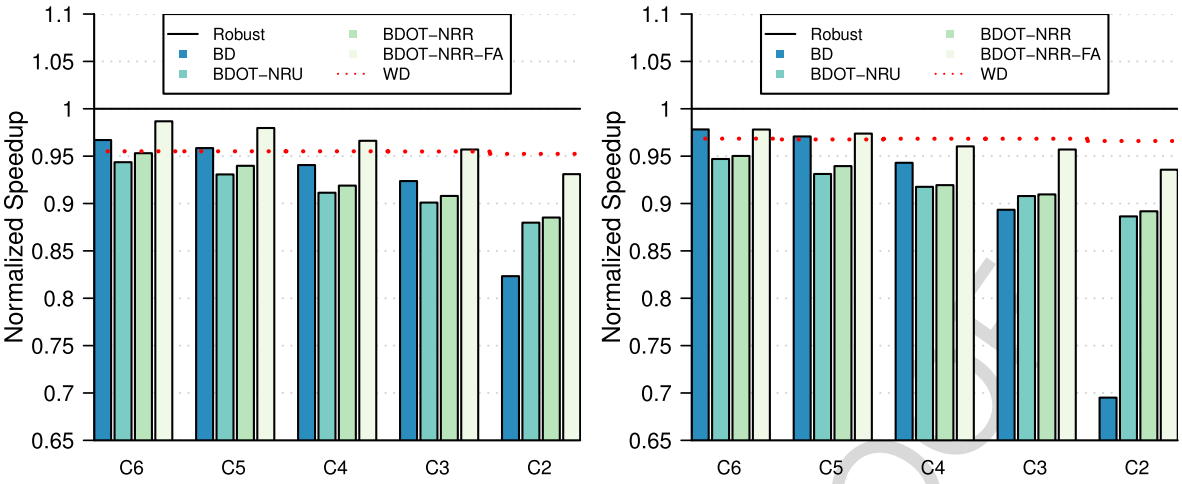
For multiprogrammed workloads (Fig. 9(a)), performance follows the same trend as MPKI, BDOT-NRR-FA being the best design option except in the case of C2 cells, for which WD outperforms BDOT-NRR-FA by 2.2%. In particular, BDOT-NRR-FA shows a performance degradation with respect to the Robust reference system of 1.3%, 2%, 3.4%, 4.3%, and 6.9% for C6, C5, C4, C3, and C2, respectively, or, in other words, a performance improvement with respect to BD of 2%, 2.2%, 2.7%, 3.6%, and 13.1%.



(a) `canneal`. (b) `ferret`.

(c) `streamcluster`. (d) `vips`.

Fig. 8. Per-application normalized MPKI (PARSEC) with respect to Robust for the different proposals and cell types. Average MPKIs for Robust: 4.26, 1.59, 1.0 and 1.19, for `canneal`, `ferret`, `streamcluster`, and `vips`, respectively.



(a) Multiprogrammed workloads. (b) Parallel workloads.

Fig. 9. Normalized speedup (average) with respect to Robust for the different proposals and cell types.

As in the case of multiprogrammed workloads, speedup in parallel application performance (Fig. 9(b)) also follows the same trend as in the MPKI results, with a notable exception. For C3, BDOT-NRU and BDOT-NRR perform slightly better than BD on average, while in Fig. 7, the average MPKI value with these techniques was higher than with BD. As we already mentioned, the LLC MPKI for the parallel applications in the baseline system is small (Section 6), and small MPKI increases with respect to this system appear relatively large in Fig. 7. Nevertheless, for C3, `streamcluster` has a dramatic speedup degradation with BD. This is due to the large number of back invalidations to L1 blocks to force directory inclusion (inclusion victims). Specifically, in this application, the number of invalidations to L1 blocks decreases 20 times when implementing BDOT. The MPKI numbers are similar, but the number of instructions executed differ considerably. For this application, we observe a performance improvement of 6.1% when using BDOT-NRU (6.2% for BDOT-NRR), with respect to BD.

On average, BDOT-NRR-FA shows a similar performance to BD for C6 and C5, where the performance degradation with respect to the reference system is 2.2% and 2.9%, respectively, and for C4, C3, and C2, the performance is better, by 1.8%, 7.1%, and 34.6%, respectively. BDOT-NRR-FA and WD have similar performance (within 1%), except for in the case of C2, for which WD achieves a 3.1% better performance.

In summary, BDOT-NRR-FA is an excellent choice for caches with different numbers of defective entries, as it achieves as good performance as more complex fault-tolerant techniques without adding any extra storage overhead to the cache.

8.2. Area and energy

Larger SRAM cells are less likely to fail, but at the cost of larger areas and higher power consumption. Even the largest cell considered by Zhou's study (C6), which requires a 41.1% larger area than C2, is far from reaching fully functional performance: 40.1% of the cache entries are faulty at 0.5 V (Table 1).

Our fault-aware mechanism has a minimal impact on area. Only two extra bits suffice to implement BDOT-NRR-FA: one bit marks entries as defective (as in BD), and the other one stores the replacement policy (i.e., NRR) information. Thus, no extra storage overhead is added compared to the BD system.

Minimizing area helps to reduce energy in the LLC. Signals traveling smaller distances require less dynamic power for switching, and, most importantly, small cells consume less static power. To estimate the sub-threshold current, I_{sub}, causing the static consumption, we assume that I_{sub} is directly proportional to the transistor width of the cells considered, and estimate it with respect to C2 [47]. For the unrealistically robust cell, we assume that it is the same size as C2, but with a null probability of failure. Energy consumption also includes the dynamic overhead of LLC block swaps and L1 clean data eviction required by the fault-aware BDOT policy. Finally, we account for both the on-chip power and the off-chip DRAM power.

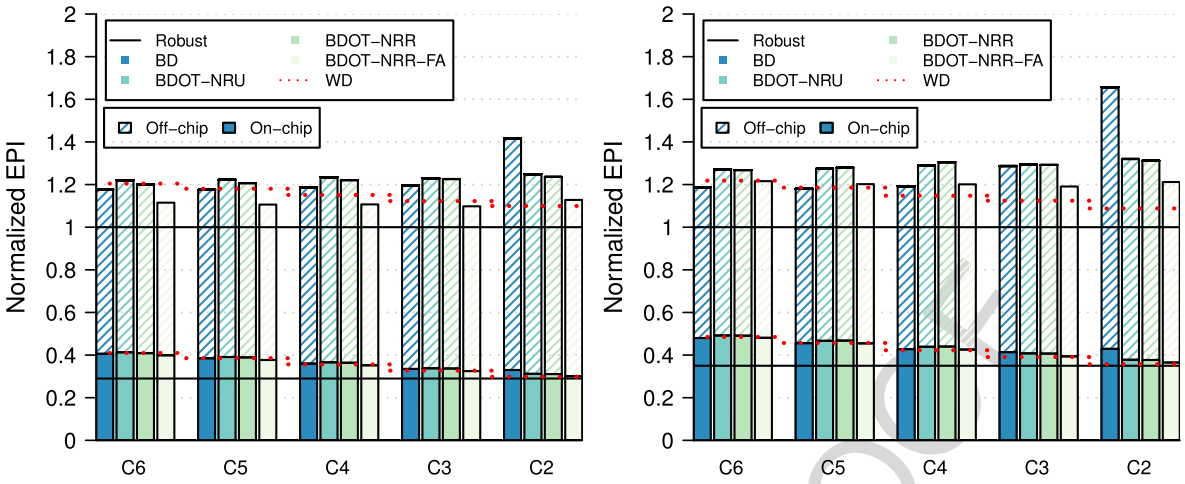
Fig. 10 shows the energy per instruction (EPI) for all the systems and cell types considered, both for multiprogrammed (Fig. 10(a)) and parallel (Fig. 10(b)) workloads, with respect to a system implemented with robust cells at 0.5 V, distinguishing between on-chip and off-chip consumption.

For BD, the 2.4-fold higher MPKI for C2 escalates the off-chip DRAM traffic, and in turn, significantly increases off-chip DRAM EPI for both multiprogrammed and parallel workloads. On average, BDOT-NRR-FA results in a 5.4%, 5.8%, 6.8%, 8.2%, and 20.4% lower overall EPI than BD for C6, C5, C4, C3, and C2, respectively, for the multiprogrammed workloads. In the case of parallel workloads, the EPI of BDOT-NRR-FA is within 2% of BD for C6, C5, and C4, and 7.4% and 26.8% lower for C3 and C2, respectively.

Regarding WD, the results show the same trend as performance, namely, BDOT-NRR-FA EPI results are 7.5%, 9.8%, 7%, and 4% lower for C6, C5, C4, and C3, respectively, when considering multiprogrammed workloads, while for parallel workloads, the EPI values of the two techniques are very similar for C6 and C5, but BDOT-NRR-FA cannot achieve the efficiency of WD for the other cell configurations.

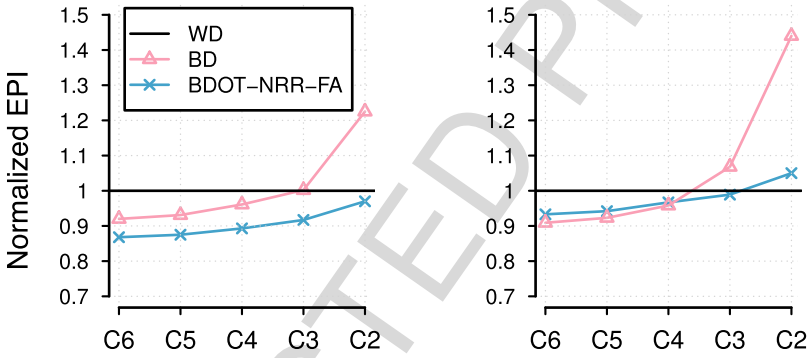
The energy results shown above do not consider any block power gating technique [39]. Assuming a more aggressive approach, where fine-grained block power gating is affordable [18], the benefits of BD-based techniques in terms of power and energy will be enhanced, as faulty entries do not consume static power during operation. Applying this technique, the EPI with BDOT-NRR-FA would be 6.2%, 6.7%, 7.2%, 6.3%, and 5.5% lower for the multiprogrammed workloads than the EPI values in Fig. 10 with C6, C5, C4, C3, and C2 cells, respectively. The same tendency is observed in the parallel workload results.

Fig. 11 compares the EPI values with BD and BDOT-NRR-FA when implementing block power gating with those obtained with WD. We observe that for multiprogrammed workloads all the cell configurations achieve significant improvements in terms of EPI with respect to WD, and in the case of parallel workloads, only the C2 configuration is not able to reach the WD efficiency.



(a) Multiprogrammed workloads. (b) Parallel workloads.

Fig. 10. Normalized EPI (average) with respect to Robust for the different proposals and cell types.



(a) Multiprogrammed workloads. (b) Parallel workloads.

Fig. 11. Normalized EPI (average) with respect to word disabling, when implementing fine-grained block power gating.

9. Related work

Conventional 6T SRAM cells fail to operate reliably in the near-threshold regime, as the ratio constraints for read stability and writability of transistors cannot be guaranteed, especially in view of V_{th} variations. Prior proposals to mitigate the impact of SRAM cell failures due to parameter variation at ultra-low voltages can be categorized into circuit and architectural solutions.

Circuit solutions include methods that improve the bit cell resilience by increasing its size [47], or by adding assist/spare circuitry [10,29]. Increasing the cell size or the number of transistors per cell comes at the cost of significant increases in the SRAM area (lower density) and power consumption. Since the LLC accounts for much of the die size, increasing its area (e.g., ST SRAM cells [29] double the area of the SRAM structure) is not a design option.

Architectural solutions include redundancy through ECCs, disabling techniques, and duplication mechanisms. Our proposal fits in this category.

ECCs are extensively employed to protect designs against soft errors. Some studies have extended the use of ECCs to protect against hard errors when running at ultra-low voltages [3,13]. ECCs are usually optimized to minimize their storage requirements, at the cost of complex logic to detect and correct errors. Thus, the ability to detect and correct more errors comes at the cost of increasing the complexity of the decoding stage, or the storage requirements of the check bits [13]. Our proposal is orthogonal to the use of ECCs to provide more functional entries (or any other technique that increases the number of functional entries), as it adapts seamlessly to the amount of functional and non-functional data entries in the cache.

Regarding BD [42], Lee et al. examine performance degradation of disabling cache lines, sets, ways, ports, or the complete cache in a single processor environment [32]. Ladas et al. implement a victim cache to compensate for the loss in associativity [31]. Our approach also relies on BD, but does not require any additional structures.

Ghasemi et al. propose the use of heterogeneous cell sizes, in order that when operating at low-power, ways or sets made of smaller SRAM cells are deactivated if they start to fail [17]. Khan et al. propose a mixed-cell memory design, where a small portion of the cache is implemented with robust cells, which store dirty cache blocks, and the remainder with non-robust cells [26]. They modify the replacement policy to guide the allocation of blocks based on the type of request (load or store). Zhou et al. combine spare cells, heterogeneous cell sizes, and ECCs into a hybrid design to improve on the effectiveness obtained by any single technique alone [47]. In contrast to these techniques, we do not rely on the existence of robust ways and we guide the allocation of blocks to faulty or operational LLC entries based on their reuse.

The granularity at which capacity is disabled could be finer, though this would add complexity to cache accesses. Word disabling tracks defects at word-level granularity, combining two consecutive cache entries into a single fault-free entry, halving both associativity and capacity [45]. Abella et al. propose to bypass faulty subentries rather than disable full cache lines, but this

technique is suitable only for the first-level cache, where accesses are word wide [1]. Palframan et al. follow a similar approach, patching faulty words from other microprocessor structures, such as the store queue or the miss status holding register [38]. Ferrerón et al. compress cache blocks to fit them in faulty entries, allowing the utilization of 100% of the cache entries [15]. More complex schemes couple faulty cache entries using a remapping mechanism [5,28,35]. They group collision-free cache entries (from the same or different cache banks) relying on the execution of a complex algorithm and structures to store all the mapping strategy. Re-mapping mechanisms add a level of indirection to the cache access (increasing its latency), and the combination of cache entries to recreate a cache block adds complexity. Besides, several cache accesses are needed to obtain a fault-free cache block, increasing the energy consumption and/or the block access latency. Unlike the aforementioned proposals, we do not add any additional structures or re-mapping mechanisms, only minor changes to the coherence protocol and replacement policy.

In the context of ultra-low voltages, Keramidas et al. use a PC-indexed spatial predictor to orchestrate the replacement decisions among fully and partially usable entries in first-level caches [25]. We based our allocation predictions on reuse patterns, which simplifies the hardware, and we do not consider the use of partially faulty entries.

Regarding the implementation of our techniques, it is worth referring to the work of Jaleel et al. [23]. In inclusive hierarchies, the private caches filter the temporal locality and hot blocks (i.e., blocks being actively used by the processor) are degraded in the replacement algorithm of the LLC, eventually being evicted. They address this problem by protecting blocks present in the private caches and preventing their replacement in the LLC through several techniques, including: sending hints to the LLC, identifying temporal locality via early invalidation, and querying the private caches about the presence of blocks. We also protect private copies in all the replacement policies considered (including the baseline one), in our case by using the coherence information and assuming non-silent evictions of clean blocks.

Albericio et al. base replacement decisions on block reuse locality [4]. They propose the *Not-Recently Reused* (NRR) algorithm, which protects blocks present in the private caches and blocks that have shown reuse in the LLC. Their simple yet efficient implementation achieved better performance than more complex techniques such as RRIP [24]. Our proposal uses NRR as the base replacement policy.

10. Summary and conclusions

Voltage reduction has been the primary driver to reduce power during recent decades, but ultra-deep-submicron technologies have suddenly stopped this trend because of problems with leakage and stability. Manufacturing-induced parameter variations make SRAM cells unstable at lower voltages, meaning that they require a minimum voltage to operate reliably. SRAM cell failures can be tolerated by deactivating faulty cache entries. This technique is called Block Disabling (BD) and requires only one bit per tag. Unfortunately, as the number of defective entries increases, so does performance degradation, and the energy saved from decreasing V_{dd} does not compensate for the extra energy required for the additional main memory accesses.

The reduction in associativity and capacity experienced by inclusive LLCs extended with BD has two specific drawbacks in multicore systems. First, the number of inclusion victims in private L1 caches increases. Second, the MPKI values also grow, increasing LLC miss latency and main memory energy consumption.

To cope with the first problem, we propose Block Disabling with Operational Tags (BDOT), which uses robust cells to implement the LLC tag array. BDOT enables some cache blocks to be only in private levels by simply tracking their tags (T entries), and extends the existing cache-to-cache coherence service to clean blocks. Thus, with regard to inclusion victims, the LLC associativity is fully restored. BDOT requires a small amount of extra control, and it adds no storage overhead to BD (the bit that marks operative entries sufficing to distinguish between LLC T and D entries). Any replacement algorithm may work with BDOT, and we have tested NRU and NRR, two low-cost state-of-the-art proposals for LLCs.

After the last copy L1 eviction of a block tracked by a T entry, a future reference to this block will involve an off-chip access, even though we know that reuse chances are high. Hence, we tackle the second problem from the key observation that we can preserve the data cached on-chip by exchanging the valuable, just evicted T entry block (promotion), for an L1-present D entry block (demotion). Furthermore, if all blocks allocated to D entries lack L1 copies, we can still resort to demotion, losing effective on-chip capacity, assuming that an incoming L1 block showing reuse (second L1 replacement) is more valuable than any older block allocated to a D entry. We have implemented these ideas in BDOT-NRR-FA, the fault-aware version of BDOT that selects for demotion a D entry victim block that has a backup copy in L1 (first criterion), and has not shown reuse in the LLC (second criterion). Compared to a BDOT LLC using NRR replacement, BDOT-NRR-FA improves performance and energy efficiency with no area overhead, because the bits per block required, namely for the presence vector, operative entry, and reuse are required, respectively, by the coherence mechanism, BD, and conventional replacement.

We tested our proposals against a wide range of multiprogrammed and parallel workloads under different P_{fail} situations. Our best proposal, BDOT-NRR-FA, beats BD, results in up to 37.3% and 54.2% lower MPKI values for multiprogrammed and parallel workloads, respectively. These decreases translate to performance improvements of 13% and 34.6%, respectively. Regarding energy use, our proposal decreases EPI by between 5.4% and 20.4% for multiprogrammed, and between 2% and 26.8% for parallel workloads. The largest savings come from LLCs with the most faulty cells, and gains are consistent across programs, making our proposal very suitable for the operation of multicore LLCs at low voltages for current and future technology nodes.

References

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, A. González, Low Vccmin fault-tolerant cache with highly predictable performance, in: 42nd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2009, pp. 111–121, http://dx.doi.org/10.1145/1669112.1669128.
[2] A. Agarwal, B. Paul, H. Mahmoodi, A. Datta, K. Roy, A process-tolerant cache architecture for improved yield in nanoscale technologies, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 13 (1) (2005) 27–38, http://dx.doi.org/10.1109/TVLSI.2004.840407.
[3] A. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, S.L. Lu, Energy-efficient cache design using variable-strength error-correcting codes, in: 38th Annual Int. Symp. on Computer Architecture, 2011, pp. 461–471, http://dx.doi.org/10.1145/2000064.2000118.
[4] J. Albericio, P. Ibáñez, V. Viñals, J.M. Llabería, Exploiting reuse locality on inclusive shared last-level caches, ACM Trans. Archit. Code Optim. 9 (4) (2013) 38:1–38:19.
[5] A. Ansari, S. Feng, S. Gupta, S. Mahlke, Archipelago: A polymorphic cache design for enabling robust near-threshold operation, in: IEEE 17th Int. Symp. on High Performance Computer Architecture, 2011, pp. 539–550, http://dx.doi.org/10.1109/HPCA.2011.5749758.
[6] J.L. Baer, W. Wang, On the inclusion properties for multi-level cache hierarchies, in: 15th Annual Int. Symp. on Computer Architecture, 1998, pp. 73–80, http://dx.doi.org/10.1145/633625.52409.
[7] A.J. Bhavnagarwala, X. Tang, J.D. Meindl, The impact of intrinsic device fluctuations on CMOS SRAM cell stability, IEEE J. Solid-State Circuits 36 (4) (2001) 658–665, http://dx.doi.org/10.1109/4.913744.
[8] C. Bienia, S. Kumar, J.P. Singh, K. Li, The PARSEC benchmark suite: Characterization and architectural implications, in: 17th Int. Conf. on Parallel Architectures and Compilation Techniques, 2008, pp. 72–81, http://doi.acm.org/10.1145/1454115.1454128.

[9] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, D. Srivastava, The 65-nm 16-MB shared on-die L3 cache for the dual-core intel xeon processor 7100 series, IEEE J. Solid-State Circuits 42 (4) (2007) 846–852, http://dx.doi.org/10.1109/JSSC.2007.892185.
[10] L. Chang, R. Montoye, Y. Nakamura, K. Batson, R. Eickemeyer, R. Dennard, W. Haensch, D. Jamsek, An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches, IEEE J. Solid-State Circuits 43 (4) (2008) 956–963, http://dx.doi.org/10.1109/JSSC.2007.917509.
[11] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney, J. Nuzman, Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches, in: 21st Int. Conf. on Parallel Architectures and Compilation Techniques, 2012, pp. 293–304, http://dx.doi.org/10.1145/2370816.2370860.
[12] L. Cheng, P. Gupta, C.J. Spanos, K. Qian, L. He, Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (3) (2011) 388–401, http://dx.doi.org/10.1109/TCAD.2010.2089568.
[13] Z. Chishti, A.R. Alameldeen, C. Wilkerson, W. Wu, S.-L. Lu, Improving cache lifetime reliability at ultra-low voltages, in:42nd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2009, pp. 89–99, http://dx.doi.org/10.1145/1669112.1669126.
[14] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, T. Mudge, Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits, Proc. IEEE 98 (2) (2010) 253–266, http://dx.doi.org/10.1109/JPROC.2009.2034764.
[15] A. Ferreron, D. Suarez, J. Alastruey, T. Monreal, P. Ibañez, Concertina: Squeezing in cache content to operate at near-threshold voltage, IEEE Trans. Comput. 65 (3) (2016) 755–769, http://dx.doi.org/10.1109/TC.2015.2479585.
[16] A. Ferrerón, D. Suárez-Gracia, J. Alastruey-Benedé, T. Monreal, V. Viñals, Block disabling characterization and improvements in CMPs operating at ultra-low voltages, in: 2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing, 2014, pp. 238–245, http://dx.doi.org/10.1109/SBAC-PAD.2014.12.
[17] H. Ghasemi, S. Draper, N.S. Kim, Low-voltage on-chip cache architecture using heterogeneous cell sizes for high-performance processors, in: IEEE 17th Int. Symp. on High Performance Computer Architecture, 2011, pp. 38–49, http://dx.doi.org/10.1109/HPCA.2011.5749715.
[18] M. Gottscho, A. BanaiyanMofrad, N. Dutt, A. Nicolau, P. Gupta, DPCS: Dynamic power/capacity scaling for SRAM caches in the nanoscale era, ACM Trans. Archit. Code Optim. 12 (3) (2015) 27:1–27:26, http://dx.doi.org/10.1145/2792982.
[19] J. Handy, The Cache Memory Book, Morgan Kaufmann, 1998.
[20] D. Hardy, I. Sideris, N. Ladas, Y. Sazeides, The performance vulnerability of architectural and non-architectural arrays to permanent faults, in: 45th Annual IEEE/ACM Int. Symp. on Microarchitecture, 2012, pp. 48–59, http://dx.doi.org/10.1109/MICRO.2012.14.
[21] J.L. Henning, SPEC CPU2006 benchmark descriptions, SIGARCH Comput. Archit. News 34 (4) (2006) 1–17, http://dx.doi.org/10.1145/1186736.1186737.
[22] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, 1991.
[23] A. Jaleel, E. Borch, M. Bhandaru, S.C. Steely Jr., J. Emer, Achieving non-inclusive cache performance with inclusive caches: Temporal Locality Aware (TLA) cache management policies, in: 43rd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2010, pp. 151–162, http://dx.doi.org/10.1109/MICRO.2010.52.
[24] A. Jaleel, K.B. Theobald, S.C. Steely Jr., J. Emer, High performance cache replacement using re-reference interval prediction (RRIP), in: 37th Annual Int. Symp. on Computer Architecture, 2010, pp. 60–71, http://dx.doi.org/10.1145/1816038.1815971.
[25] G. Keramidas, M. Mavropoulos, A. Karvouniari, D. Nikolos, Spatial pattern prediction based management of faulty data caches, in: Conference on Design, Automation & Test in Europe, 2014, pp. 60:1–60:6, http://dl.acm.org/citation.cfm?id=2616606.2616680.
[26] S.M. Khan, A.R. Alameldeen, C. Wilkerson, J. Kulkarni, D.A. Jimenez, Improving multi-core performance using mixed-cell cache architecture, in: IEEE 19th Int. Symp. on High Performance Computer Architecture, 2013, pp. 119–130, http://dx.doi.org/10.1109/HPCA.2013.6522312.
[27] S.M. Khan, Y. Tian, D.A. Jimenez, Sampling dead block prediction for last-level caches, in: 43rd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2010, pp. 175–186, http://dx.doi.org/10.1109/MICRO.2010.24.
[28] C.-K. Koh, W.-F. Wong, Y. Chen, H. Li, Tolerating process variations in large, set-associative caches: The buddy cache, ACM Trans. Archit. Code Optim. 6 (2) (2009) 8:1–8:34, http://dx.doi.org/10.1145/1543753.1543757.
[29] J. Kulkarni, K. Kim, K. Roy, A 160 mV robust schmitt trigger based subthreshold SRAM, IEEE J. Solid-State Circuits 42 (10) (2007) 2303–2313, http://dx.doi.org/10.1109/JSSC.2007.897148.
[30] R. Kumar, G. Hinton, A family of 45 nm IA processors, in: IEEE Int. Solid-State Circuits Conf. Digest of Technical Papers, 2009, pp. 58–59, http://dx.doi.org/10.1109/ISSCC.2009.4977306.
[31] N. Ladas, Y. Sazeides, V. Desmet, Performance-effective operation below Vcc-min, in: IEEE Int. Symp. on Performance Analysis of Systems Software, 2010, pp. 223–234, http://dx.doi.org/10.1109/ISPASS.2010.5452017.
[32] H. Lee, S. Cho, B. Childers, Performance of graceful degradation for cache faults, in: IEEE Computer Society Annual Symp. on VLSI, 2007, pp. 409–415, http://dx.doi.org/10.1109/ISVLSI.2007.81.
[33] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures, in: 42nd Annual IEEE/ACM Int. Symp. on Microarchitecture, 2009, pp. 469–480, http://dx.doi.org/10.1145/1669112.1669172.
[34] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: A full system simulation platform, Computer 35 (2) (2002) 50–58, http://dx.doi.org/10.1109/2.982916.
[35] T. Mahmood, S. Kim, S. Hong, Macho: A failure model-oriented adaptive cache architecture to enable near-threshold voltage scaling, in: IEEE 19th Int. Symp. on High Performance Computer Architecture, 2013, pp. 532–541, http://dx.doi.org/10.1109/HPCA.2013.6522347.
[36] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset, SIGARCH Comput. Archit. News 33 (4) (2005) 92–99, http://dx.doi.org/10.1145/1105734.1105747.
[37] S. Microsystems, UltraSPARC T2 supplement to the UltraSPARC architecture, Draft d1.4.3, Sun Microsystems Inc., 2007.
[38] D.J. Palframan, N.S. Kim, M.H. Lipasti, iPatch: Intelligent fault patching to improve energy efficiency, in: IEEE 21st Int. Symp. on High Performance Computer Architecture, 2015, pp. 428–438, http://dx.doi.org/10.1109/HPCA.2015.7056052.
[39] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, T.N. Vijaykumar, Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories, in: Int. Symp. on Low Power Electronics and Design, 2000, pp. 90–95, http://dx.doi.org/10.1109/LPE.2000.155259.
[40] P. Rosenfeld, E. Cooper-Balis, B. Jacob, DRAMSim2: A cycle accurate memory system simulator, Comput. Archit. Lett. 10 (1) (2011) 16–19, http://dx.doi.org/10.1109/L-CA.2011.4.
[41] D. Sánchez, Y. Sazeides, J.M. Cebrián, J.M. García, J.L. Aragón, Modeling the impact of permanent faults in caches, ACM Trans. Archit. Code Optim. 10 (4) (2013) 29:1–29:23, http://dx.doi.org/10.1145/2541228.2541236.
[42] G. Sohi, Cache memory organization to enhance the yield of high performance VLSI processors, IEEE Trans. Comput. 38 (4) (1989) 484–492, http://dx.doi.org/10.1109/12.21141.
[43] X. Tang, V.K. De, J.D. Meindl, Intrinsic MOSFET parameter fluctuations due to random dopant placement, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 5 (4) (1997) 369–376, http://dx.doi.org/10.1109/92.645063.
[44] M. Taylor, A landscape of the new dark silicon design regime, IEEE Micro 33 (5) (2013) 8–19, http://dx.doi.org/10.1109/MM.2013.90.
[45] C. Wilkerson, H. Gao, A.R. Alameldeen, Z. Chishti, M. Khellah, S.-L. Lu, Trading off cache capacity for reliability to enable low voltage operation, in: 35th Annual Int. Symp. on Computer Architecture, 2008, pp. 203–214, http://dx.doi.org/10.1109/ISCA.2008.22.
[46] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S.C. Steely Jr., J. Emer, SHiP: Signature-based hit predictor for high performance caching, in: 44th Annual IEEE/ACM Int. Symp. on Microarchitecture, 2011, pp. 430–441, http://dx.doi.org/10.1145/2155620.2155671.
[47] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, N.S. Kim, Minimizing total area of low-voltage SRAM arrays through joint optimization of cell size, redundancy, and ECC, in: IEEE Int. Conf. on Computer Design, 2010, pp. 112–117, http://dx.doi.org/10.1109/ICCD.2010.5647605.

Alexandra Ferrerón received the M.S. and Ph.D. degrees in computer science engineering from the Universidad de Zaragoza, Spain, in 2013 and 2016, respectively. Her research interests include high-performance low-power on-chip memory hierarchies, ultra-low and near-threshold voltage computing, and High Performance Computing. She currently works as Site Reliability Engineer for BigQuery (Google Cloud Platform) at Google Switzerland.

Jesús Alastruey-Benedé received the Telecommunications Engineering degree and the Ph.D. degree in Computer Science from the Universidad de Zaragoza, Spain, in 1997 and 2009, respectively. He is a Lecturer in the Departmento de Informática e Ingeniería de Sistemas (DIIS), Universidad de Zaragoza, Spain. His research interests include processor microarchitecture, memory hierarchy, and High Performance Computing (HPC) applications. He is a member of the Instituto de Investigación en Ingeniería de Aragón (I3A) and the European HiPEAC NoE.

Please cite this article as: A. Ferrerón, J. Alastruey-Benedé, D. Suárez Gracia et al., A fault-tolerant last level cache for CMPs operating at ultra-low voltage, Journal of Parallel and Distributed Computing (2018), https://doi.org/10.1016/j.jpdc.2018.10.010.

Darío Suárez Gracia (S'08, M'12) received the Ph.D. degree in computer engineering from the Universidad de Zaragoza, Spain, in 2011. From 2012 to 2015, he was been working at Qualcomm Research Silicon Valley on power aware parallel and heterogeneous computing for mobile devices. Currently, he is an assistant professor at the Universidad de Zaragoza in Spain. His research interests include parallel programming, heterogeneous computing, memory hierarchy design, networks-on-chip, and accelerators for computer vision applications. He is also a member of the IEEE, the IEEE Computer Society, and the Association for Computing Machinery.

Teresa Monreal Arnal received the M.S. degree in Mathematics and the Ph.D. degree in Computer Science from the University of Zaragoza, Spain, in 1991 and 2003, respectively. Until 2007, she was with the Informática e Ingeniería de Sistemas Department (DIIS) at the University of Zaragoza, Spain. Currently, she is an Associate Professor with the Computer Architecture Department (DAC) at the Universitat Politécnica de Catalunya (UPC), Spain. Her research interests include processor microarchitecture, memory hierarchy, and parallel computer architecture. She collaborates actively with the Grupo de Arquitectura de Computadores from the University of Zaragoza (gaZ).

Pablo Ibáñez Marín received the M.S. degree in computer science from the Universitat Politécnica de Catalunya in 1989, and the Ph.D. degree in computer science from the Universidad de Zaragoza in 1998. He is an Associate Professor in the Departamento de Informática e Ingeniería de Sistemas (DIIS) at the Universidad de Zaragoza, Spain. His research interests include processor microarchitecture, memory hierarchy, parallel computer architecture, and High Performance Computing (HPC) applications. He is a member of the Instituto de Investigación en Ingeniería de Aragón (I3A) and the European HiPEAC NoE.

Víctor Viñals Yúfera received the M.S. degree in Telecommunications and the Ph.D. degree in Computer Science from the Universitat Politècnica de Catalunya (UPC) in 1982 and 1987, respectively. He was associate professor in the Facultat d'Informática de Barcelona from 1983 to 1988. Currently, he is full professor in the Informática e Ingeniería de Sistemas Department at the University of Zaragoza (Spain). His research interests include processor microarchitecture, memory hierarchy, and parallel computer architecture. He is member of the ACM, the IEEE Computer Society, and HiPEAC. He also belongs to the Computer Architecture Group and the I3A Institute of the University of Zaragoza.

Please cite this article as: A. Ferrerón, J. Alastruey-Benedé, D. Suárez Gracia et al., A fault-tolerant last level cache for CMPs operating at ultra-low voltage, Journal of Parallel and Distributed Computing (2018), https://doi.org/10.1016/j.jpdc.2018.10.010.