Concertina: Squeezing in Cache Content to Operate at Near-Threshold Voltage

Alexandra Ferrerón, *Student Member, IEEE,* Darío Suárez-Gracia, *Member, IEEE,*
Jesús Alastruey-Benedé, Teresa Monreal-Arnal, and Pablo Ibáñez, *Member, IEEE*

Abstract—Scaling supply voltage to values near the threshold voltage allows a dramatic decrease in the power consumption of processors; however, the lower the voltage, the higher the sensitivity to process variation, and, hence, the lower the reliability. Large SRAM structures, like the last-level cache (LLC), are extremely vulnerable to process variation because they are aggressively sized to satisfy high density requirements. In this paper, we propose Concertina, an LLC designed to enable reliable operation at low voltages with conventional SRAM cells. Based on the observation that for many applications the LLC contains large amounts of null data, Concertina compresses cache blocks in order that they can be allocated to cache entries with faulty cells, enabling use of 100% of the LLC capacity. To distribute blocks among cache entries, Concertina implements a compression- and fault-aware insertion/replacement policy that reduces the LLC miss rate. Concertina reaches the performance of an ideal system implementing an LLC that does not suffer from parameter variation with a modest storage overhead. Specifically, performance degrades by less than 2%, even when using small SRAM cells, which implies over 90% of cache entries having defective cells, and this represents a notable improvement on previously proposed techniques.

Index Terms—Near-threshold voltage, SRAM variability, fault-tolerance, on-chip caches

1 Introduction

From tiny wearable devices to massive data centers, power density has become the *de facto* limiter for improving performance. Unfortunately, neither increasing transistor count nor reducing integration scale can fuel advances in performance anymore [37]; the number of active transistors has topped off, and voltage does not scale with technology.

The most straightforward way to reduce power density is to scale supply voltage (V_{dd}), but scaling is limited by the tight functionality margins of SRAM cells in last-level cache (LLC) transistors. Below a minimum operating voltage ($V_{dd_{min}}$), typically of the order of 0.7–1.0V for regular six-transistor (6T) SRAM cells, process parameter variation in nanoscale technology has become so severe that SRAM cells no longer operate reliably.

Existing approaches to improving SRAM reliability can be classified into two groups: circuit- and architecture-based. Circuit-level techniques improve the reliability of the SRAM cell in low-voltage operation by increasing its size or by adding assist circuitry [24], [42]. These come at the cost of an increase in power consumption and array area, which is not practical for large structures such as the on-chip LLC. At the architectural level, cache-tolerant designs rely on: (a) correcting defective bits through error correction codes (ECCs), increasing their complexity to enable them to detect and repair more defective bits per block [14]; (b) disabling faulty resources (words, lines, ways), marking a resource as defective whenever one faulty cell is found [27]; or (c) combining faulty resources to create functional ones, relying on the fact that several defective entries can be combined to store a block in a distributed manner [5], [39]. In all cases, a significant fraction of the memory structure is either disabled or sacrificed to store redundant information, degrading the overall effective cache capacity.

In this paper, we present Concertina, an efficient and fault-tolerant LLC that operates with unreliable SRAM cells at ultra-low voltages. Unlike previous architectural schemes, Concertina enables all the cache entries, even the faulty ones. Our key idea is to compress cache blocks, in order that they fit within the functional elements of faulty entries. Since not all cache blocks can be evenly compressed and not all faulty entries can store the same amount of information, it is not possible to use existing cache management policies based on the premise that a block can be stored in any entry of a set. To address this issue, we study different insertion/replacement policies that are aware of the nature of both the incoming block (degree of compression) and the corresponding cache set entries (number of defective cells).

Compression has been proposed in related literature as a promising technique to increase the effective on-chip cache capacity [2], [13], [34], [40], or to reduce the energy consumption of the cache [12], [22], but to the best of our knowledge, it has not been explored as a way to enhance tolerance to cache faults in the context of ultra-low voltage operation. This application sets different requirements for the compression scheme. On the one hand, the probability of failure is high for conventional SRAM cells, and hence a large fraction of cache blocks have to be compressed. On the other hand, low compression ratios are good enough to recover

A. Ferrerón, J. Alastruey-Benedé, and P. Ibáñez are with the Departmento de Informática e Ingeniería de Sistemas and the Instituto de Investigación en Ingeniería de Aragón (I3A), Universidad de Zaragoza, Spain, and HiPEAC. E-mail: {ferreron, jalastru, imarin}@unizar.es
D. Suárez-Gracia is with Qualcomm Research Silicon Valley, Santa Clara, CA, USA. E-mail: dgracia@qti.qualcomm.com
T. Monreal-Arnal is with the Departamento de Arquitectura de Computadores, Universitat Politècnica de Catalunya, Spain, and HiPEAC. E-mail: teresa@ac.upc.edu
Manuscript received January XX, 2015; revised XXX XX, 2015.

a faulty cache entry, as the probability of having a high number of faulty cells is very low. This latter observation, which is in line with the analysis of other authors [4], is also crucial to our proposal. Concertina implements null subblock compression, a fast, simple, and efficient scheme that takes advantage of the large amount of zero chains present in the LLC blocks. Null subblock compression in combination with a smart compression- and fault-aware replacement policy results in performance within 2% of that of a system that implements the LLC with ideal defect-free cells.

This work makes the following contributions. First, we provide the first analysis of requirements for compression techniques at ultra-low voltages. Second, we propose the use of compression to improve cache utilization at ultra-low voltages, with a low-complexity combination- and remapping-free mechanism, and maintaining the regular nature of SRAM cells. Finally, we present and evaluate cache management policies that match compressed blocks with faulty cache entries.

The rest of this paper is organized as follows. Section 2 introduces the problem of process variation in SRAM cells, and Section 3 presents related work. Section 4 explores compression requirements at ultra-low voltages and introduces our proposed compression scheme. Section 5 presents the Concertina architecture. Section 6 describes our experimental methodology, and Section 7 presents the evaluation of our proposal. Section 8 discusses the system impact of Concertina, and Section 9 outlines our conclusions.

2 Process Variation in SRAM cells

Scaling supply voltage to the near-threshold region allows a dramatic decrease in power consumption, as dynamic power decreases quadratically with supply voltage. However, at lower voltages, devices are more sensitive to process variation, which impacts circuit functionality, and eventually limits supply voltage reduction.

Intra-die random dopant fluctuation (RDF) is a main cause of threshold voltage (V_{th}) variation. Specifically, V_{th} is determined by the number and location of dopant atoms implanted in the transistor channel region. The stochastic nature of ion implantation leads to different V_{th} values across the chip. The resultant threshold mismatch is a particular problem in nanometer processes because of the vast number of cells on a chip. SRAM structures are especially vulnerable, since they are aggressively sized to meet high density requirements, reducing read, write, and hold margins [8]. As the static noise margins depend on V_{dd}, SRAMs have a minimum voltage below which they cannot operate reliably. This voltage ($V_{dd_{min}}$) is typically of the order of 0.7–1.0V when regular 6T cells are employed.

Many authors have extensively analyzed the robustness of SRAM cells under the $V_{dd_{min}}$ range. For instance, Zhou *et al.* describe six different sizes of 6T SRAM cells in 32 nm technology, and how their probabilities of failure change with the voltage supply and the size of the constituent transistors [42]. We will take this study as a reference for the rest of the paper. According to Zhou *et al.*, at 0.5 V (our target near-threshold voltage) the probability of failure (P_{fail}) of a standard SRAM cell ranges between 10^{-3} and 10^{-2}. Larger cells have a lower probability of failure because non-uniformities in channel doping average out with larger transistors, resulting in more robust devices, but at the price of larger area, and higher energy consumption. Table 1 describes the six SRAM cells of Zhou's study (C1, C2, C3, C4, C5, and C6) with their areas relative to the smallest cell (C1), as well as the percentages of non-faulty entries of a cache implemented with the cells at 0.5 V, assuming 64-byte cache entries[1]. An entry is considered faulty if it contains at least one defective bit. As Table 1 shows, only 10% of the cache entries are non-faulty for the small C2 cell at our target voltage of 0.5 V. If the cache is implemented with the more robust C6 cells, the percentage of non-faulty cache entries rises to 60%, but at the cost of a 41.1% increase in area (relative to C2), which is not a viable option for a large structure such as the on-chip LLC.

TABLE 1
Percentage of fault-free entries in a cache at 0.5 V (64-byte cache entry), for the different cell sizes, and their areas relative to the C1 size.

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

3 Related Work

As noted above, solutions proposed to date to address the variability in SRAM cells at ultra-low voltages can be arranged into two groups: circuit and microarchitectural techniques.

Circuit solutions include proactive methods that improve the bit cell characteristics by increasing the SRAM cell size or adding assist/spare circuitry.

Larger transistors reduce V_{th} variability, since nonuniformities in channel doping average out, and result in more robust devices with a lower probability of failure [42]. Another approach to reducing variability is to add assist read/write circuitry by increasing the number of transistors per SRAM cell. Some examples are 8T [10], 10T [7], or Schmitt trigger-based (ST) SRAM cells [24]. Increasing the SRAM cell size or the number of transistors per cell comes at the cost of significant increases in the SRAM area (lower density) and in energy consumption. For example, the use of ST SRAM cells doubles the area of the SRAM structure, which is not practical for large structures such as on-chip LLCs.

A different approach is to provide separate voltages for logic and memory [41], but separate voltage domains complicate chip design [31]. Besides, scaling will likely increase the differences between voltage domains for logic and memory and eventually increase the number of voltage domains itself.

From the **microarchitecture perspective**, excessive parametric variability can cause circuit behavior consistent with a hard fault. Several runtime methods have been proposed to mitigate its impact, including redundancy through ECCs, disabling techniques, and the combination of faulty resources to recreate functional ones.

1. In this paper, we assume 64-byte cache entries without any loss of generality; all techniques described here could be easily applied to other cache entry sizes.

ECCs are extensively employed to protect designs against soft errors, and they have also been studied in ultra-low voltage contexts to protect against hard errors [3], [14]. To store an ECC, the capacity of the cache must be extended or part of its effective capacity has to be sacrificed. To reduce storage overhead, Chen *et al.* propose to store ECCs in the unused fragments that appear as a result of compression [12]. ECCs are usually optimized to minimize their storage requirements, at the cost of more complex logic to detect and correct errors. Thus, correcting more than two errors requires a high latency overhead or encodings with more check bits as described in [14], where half of the cache capacity is dedicated to storing the ECC. Finally, it is worth noting that the use of ECCs to protect against hard errors will jeopardize resilience to soft errors.

A simple approach to mitigating hard faults is to disable faulty entries (cache entries with faulty bits, which cannot store a complete cache block). This technique, called block disabling, is already implemented in modern processors to protect against hard faults [9], [15]. Block disabling has been studied at ultra-low voltages because of its easy implementation and low overhead (1 bit per entry sufficing). However, the large number of faults at low operating voltages implies a large percentage reduction in cache capacity and associativity. For instance, for C2, although defective bits represent around 0.5% of the total, only 9.9% of the capacity is available (Table 1). Lee *et al.* examine performance degradation by disabling cache lines, sets, ways, ports, or the complete cache in a single processor environment [27]. To compensate for the associative loss, Ladas *et al.* propose the implementation of a victim cache in combination with block disabling [26].

Ghasemi *et al.* propose the use of heterogeneous cell sizes, in order that when operating at low voltages, ways or sets of small SRAM cells are deactivated if they start to fail [17]. Khan *et al.* propose a mixed-cell memory design, where some of the cells are more robust and failure-resistant and, therefore, more reliable candidates to store dirty cache blocks, while the rest are designed using traditional cells [21]. Zhou *et al.* combine spare cells, heterogeneous cell sizes, and ECCs into a hybrid design to improve on the effectiveness obtained by any of the techniques applied alone [42].

The granularity of the disabled storage might be finer, but at the cost of a larger overhead. Cache entries can be divided into subentries of a given size. A defective cell implies disabling just the subentry which it belongs to, rather than the whole entry. Figure 1 shows for the six SRAM cells from Zhou's study, the potential cache capacity that can be used by tracking faults at finer granularities (different subentry sizes). For example, cells C3-C6 can potentially use more than 80% of the cache capacity by disabling 8-byte subentries.

Previous proposals took advantage of this observation. Word disabling tracks defects at word-level granularity, and then combines two consecutive cache entries into a single fault-free entry, halving both associativity and capacity [39]. Abella *et al.* bypass faulty subentries rather than disabling full cache lines, but this technique is suitable only for the first-level cache, where accesses are word wide [1].

More complex schemes couple faulty cache entries using a remapping mechanism [5], [23], [30]. They rely on the execution of a complex algorithm to group collision-free

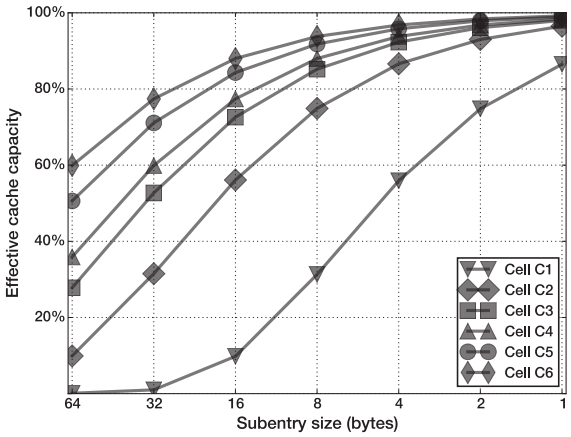


Fig. 1. Potential cache capacity that can be used by tracking faults at finer granularities for different SRAM cell sizes.

cache entries from the same or different cache banks, and on additional structures to store the mapping strategy. For example, Archipelago divides the cache into autonomous islands, where one entry is sacrificed to allocate data portions of other entries; they use a configuration algorithm (based on minimum clique covering) to create the islands [5]. The remapping mechanism adds a level of indirection to the cache access (increasing its latency), and the combination of cache entries to recreate a cache block adds complexity.

All of these combination or remapping strategies have a major inconvenience: to reconstruct cache blocks several cache accesses are needed, increasing the energy consumption and/or the latency per block access. Unlike the aforementioned proposals, an ideal fault-tolerant mechanism would not compromise cache capacity, associativity, or latency. Instead of combining faulty cache entries to recreate fully functional ones, we propose to compress cache blocks to fit into the still available capacity, avoiding any cache line remapping.

Several compression mechanisms have been proposed in the literature to increase the effective storage capacity of all the on-chip cache levels, potentially reducing misses and improving performance [2], [13], [16], [22], [34], [36], [40]. In general, these proposals seek to maximize the compression ratio either to store more than one block in each cache entry, or to reduce the energy consumption of each LLC access. However, ultra-low voltage operation sets different requirements for the compression scheme: most of the blocks need to be compressed to fit in faulty entries, but the compression ratio is relatively small. In an ideal case, where all blocks could be compressed to fit in the available space, neither capacity nor associativity would decrease.

We combine the compression mechanism with a smart allocation/replacement policy, which assigns cache blocks to faulty entries based upon their compressibility. Two recent studies have indicated the importance of taking compression information into account for a replacement policy tailored to on-chip caches implementing compression [6], [33]. Both studies are based on the fact that several blocks are allocated to the same cache entry and, therefore, prioritizing the replacement of large blocks might allow the allocation of several smaller blocks to the same space. In ultra-low voltage

operation, the compression ratio must also be taken into account in the replacement decision but in order to even out the pressure on the entries of a cache set.

4 Compression for Caches Operating at Ultra-Low Voltages

Data compression is a powerful technique for storing data in less space than originally required. In general terms, a compression mechanism seeks to be fast, especially at the decompression stage (decompression latency is on the critical path, while compression is done during replacement); and simple (the hardware and energy overhead should not overcome the benefit of the compression); as well as effective in saving storage capacity.

Compression is also an attractive idea for caches operating at ultra-low voltages because entries with defective bits could store compressed blocks, reducing the negative impact on capacity and associativity. To the best of our knowledge, this is the first time compression has been proposed for improving cache reliability at ultra-low voltages.

In this section, we first analyze the requirements imposed by near-threshold voltage operation for a compression scheme. Then, we present a simple yet effective compression scheme able to allocate most blocks to faulty cache entries.

4.1 Compression Scheme Requirements for Caches Operating at Ultra-Low Voltages

State-of-the-art compression techniques aiming to increase the effective cache capacity focus on maximizing the compression ratio (uncompressed block size divided by compressed block size) rather than coverage (fraction of compressed blocks) [11], [34], [40]. In contrast, at ultra-low voltages, the main goal is to enable all the cache capacity. Hence, there are special requirements for a compression scheme related to its coverage and compression ratio, according to our parameters of interest, the cell and cache subentry sizes:

Coverage. The fraction of cache blocks that have to be compressed depends on the probability of cell failure. As faulty cells are spread across the cache structure, when the probability of cell failure increases (as cell size decreases), more blocks need to be compressed. Table 1 shows that, for the cell sizes considered, high coverage is required, ranging from 40.1% (C6) to 100% (C1).

Compression ratio. For a block to be stored in a faulty cache entry, its size has to be less than or equal to the available space at the cache entry (the size of its fault-free subentries). Otherwise, the matching is not possible. Due to the random component of the SRAM cell failures, some entries have more faulty subentries than others. Hence, the required compression ratio varies across entries. Assuming that we can track faults at different granularities (distinct subentry sizes), Figure 2 shows the distribution of defective subentries (from zero to four or more) for cells C2 to C6. C1 is not considered as at 0.5 V there are virtually zero entries without faults.

Irrespective of cell size, when the subentry size is reduced, the average number of faulty subentries per entry increases, as a subentry with multiple defective cells may spread across several smaller faulty subentries. However, as Figure 2 shows, even for the smallest subentry size (1-byte, 64 subentries) and the smallest cell considered (C2), only 7.6% of the cache entries have more than four faulty subentries. This fraction drops to 1% for the C3 . Moreover, even with more faulty subentries, the disabled cache capacity drastically decreases when the subentry size is reduced (Figure 1).

Table 2 shows the compression ratio required to store a block in a cache entry for a range of defective subentries and subentry sizes. The compression ratio drops with decreasing subentry sizes. For instance, a 1.06 compression ratio suffices to allocate a block into a faulty cache entry with four faulty 1-byte subentries.

Summarizing, the main requirement of the compression scheme for caches operating at ultra-low voltages is a high coverage, while the required compression ratio is very small. In the rest of this paper, we focus on cells C2, C3, and C4, as they are the most discouraging scenario (highest P_{fail}).

TABLE 2
Compression ratio required to store a 64-byte block in a cache entry with several faulty subentries of different sizes

Subentry size (bytes)	Compression ratio Faulty subentries			
	1	2	3	4
32	2.00	∞^a	NA	NA
16	1.33	2.00	4.00	∞^a
8	1.14	1.33	1.60	2.00
4	1.06	1.14	1.23	1.33
2	1.03	1.06	1.10	1.14
1	1.02	1.03	1.05	1.06

[a] A cache entry with two faulty 32-byte subentries or four faulty 16-byte subentries has no available space.

4.2 Exploiting Zero Redundancy to Compress Cache Blocks

Content redundancy is prevalent among real-world applications. For example, zero is by far the most frequent value in data memory pages: it is used to initialize memory values, and to represent null pointers or false Boolean values, among others. Many compression schemes are based on compressing zeros or treating them as a special case [16], [34], [38], [40].

Exploiting zero redundancy can lead to a simple compression technique. We can compress aligned null subblocks[2] (subblocks whose content is all zeros) to effectively reduce the size of the original block. Compressing and decompressing null subblocks can be performed with low-complexity hardware: we just need to keep track of the locations of the null subblocks within a given block, and properly shift the non-null contents. This mechanism will be effective only if applications maintain a significant degree of compressibility during their execution. Hence, in this section we analyze the compressibility potential (number of null subblocks) of our target applications, and how it varies over the course of their execution.

To characterize null subblock occurrences in our applications, we conducted the following experiment. We ran the 29 SPEC CPU 2006 programs for one billion cycles

2. The same way we divide a cache entry into subentries, we divide a cache block into subblocks. Aligned subblocks simplify the design of the compression and decompression logic.

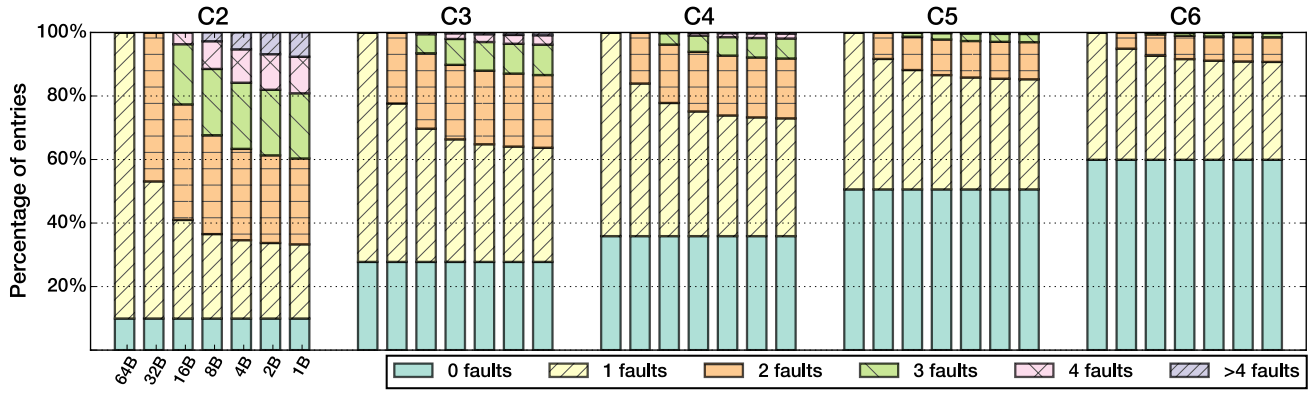


Fig. 2. Distribution of faulty subentries for different subentry and cell sizes.

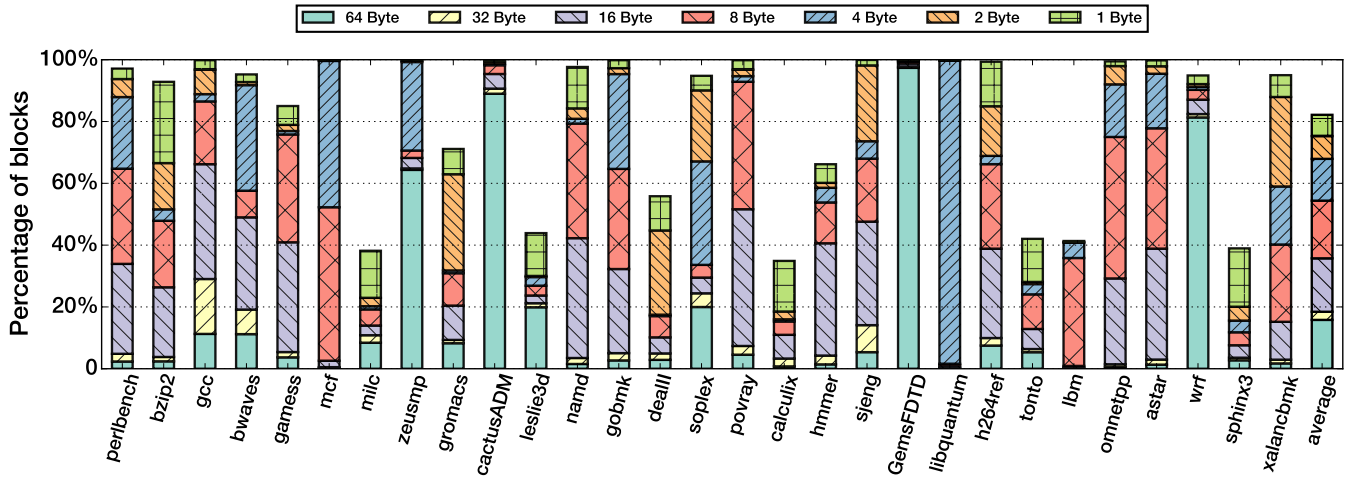


Fig. 3. Average percentage of LLC blocks that have at least one null subblock for different granularities (SPEC CPU 2006).

(Section 6 presents our methodology in more detail) and inspected the contents of the LLC every one million cycles. Each application ran alone in the system, making use of the whole shared LLC (8 MB). Then we counted up the number of blocks that had at least one null subblock for different subblock sizes (from 64 to 1 byte).

Figure 3 shows the average percentage of blocks stored in the LLC that contain at least one null subblock of a given size. Notice that large null subblocks include several occurrences of smaller null subblocks (e.g., one null 64-byte subblock is equivalent to two null 32-byte subblocks). Different workloads show distinct behaviours regarding subblock sizes and the amount of null subblocks. While some workloads such as `GemsFDTD` and `cactusADM` show a significant percentage of blocks that have large null subblocks (64 bytes), most of them show a noticeable increase in the amount of null subblocks when reducing the subblock size. On average, downsizing subblock sizes from 8 bytes to 1 increases the average fraction of compressible blocks from 54.4 to 82.2%. A large set of benchmarks (including `gcc`, `povray`, `sjeng`, and `gobmk`, among others) reach almost 100% coverage when considering 1-byte subblocks. Regarding the temporal evolution of blocks, we checked that the percentage of compressible blocks is also maintained over the course of the execution of the applications, especially when considering small subblock sizes. The detailed figures have been omitted, due to space constraints.

The conclusions extracted from our study are encouraging: not only can a significant percentage of blocks be compressed (high coverage), but also this percentage remains steady during program execution. This reinforces our belief that a null subblock compression mechanism, designed to be fast and simple, will also result in an effective technique to enable reliable LLC operation at ultra-low voltages.

5 Concertina Architecture

This section presents Concertina, our proposal to enable effective ultra-low voltage LLC operation by tolerating SRAM failures caused by parameter variation. Concertina implements null subblock compression and subblock rearrangement to enable otherwise non-usable cache entries when operating at voltages near the V_{th}. Concertina also incorporates a replacement policy to manage cache entries of different sizes, which may only contain compressed blocks. We first present the detailed implementation of Concertina. Then, we consider in detail the design of its replacement policy.

5.1 Operation and Component Description

Concertina consists of a conventional LLC plus the required logic to manage full blocks in faulty LLC entries (see Figure 4).

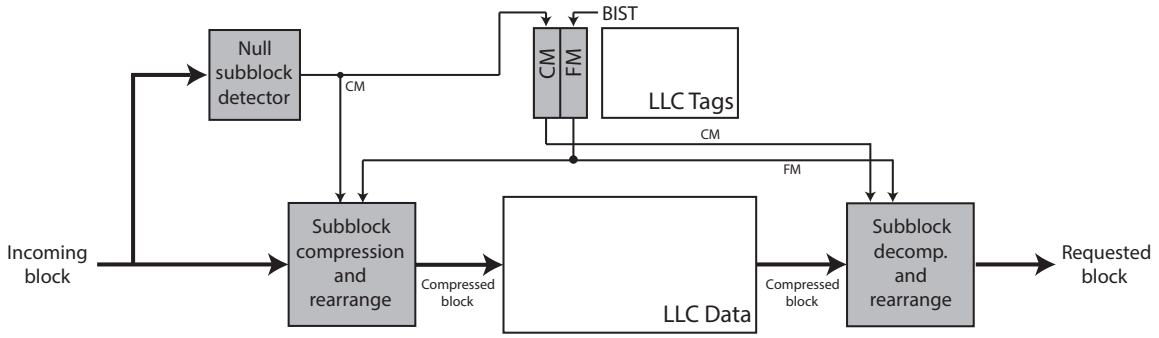
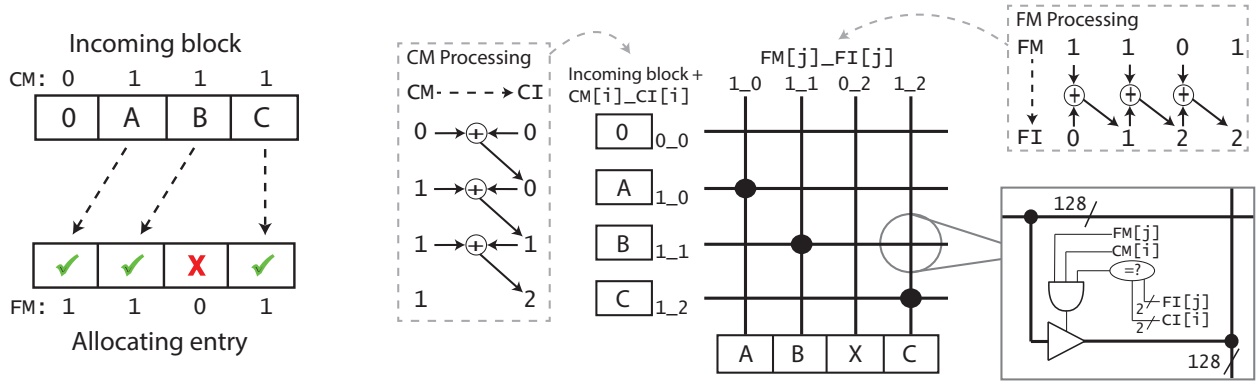


Fig. 4. Concertina design overview. Shaded boxes indicate added components.



(a) Example of compressible block to be stored at a faulty cache entry.

(b) Subblock compression and rearrangement logic.

Fig. 5. Implementation of the compression and rearrangement stage for a 64-byte block (512 bits) and a 16-byte subblock (128 bits).

To insert a block, either a refill from the main memory or a writeback from a lower cache level, Concertina performs two steps. First, it detects the location of its null subblocks if any. Then, it jointly uses this compressibility information and the location of defects in the corresponding cache entry to rearrange the non-null subblocks and store them in the functional subentries. To supply a block to a lower cache level or to evict a block, Concertina reconstructs the original block, rearranging the compressed block according to its metadata, which describes the location of null subblocks and defective subentries, and reintroduces the corresponding null subblocks.

To track the number and location of the defective subentries for each cache entry, and the number and location of null subblocks for each allocated block, Concertina uses two bit maps. Each map has as many elements as the number of subentries/subblocks considered; e.g., for a 64-byte block and a 16-byte subblock, the size of each map would be four bits. Each bit indicates whether the corresponding subentry is faulty (0) or not (1) for the fault map, FM, or if the subblock is compressed (0) or not (1) for the compression map, CM. For the sake of simplicity, we assume that the subentry and subblock sizes are the same. Later, in the evaluation section we present a more efficient implementation based on pointers for the two tracking structures.

The fault information, FM, comes from a characterization test (such as a built-in self-test, BIST) that can be executed at boot up time or during the first transition to the low-voltage mode [4], [5], [9], [39]. We consider that the number and location of faulty cells do not change during workload execution. The information about null subblocks, CM, is generated with a null subblock detector, and changes continuously during program execution. Section 7.3 discusses the implementation of these two components.

5.1.1 *Fault and Compression Metadata Sizing*

In terms of cell failures, finer granularities require lower compression ratios to allocate a block to a faulty cache entry (see Section 4.1). In terms of compression, smaller subblock sizes offer more compression coverage (see Section 4.2). However, finer granularities imply larger FM and CM sizes, which impacts area and power overheads.

We analyze this trade-off between performance and overhead varying the subblock size in our experiments. Using 16-byte or larger subblocks would cover less than one third of the cache blocks (see Figure 3), and, therefore, we will not consider these sizes.

5.1.2 *Subblock Compression and Rearrangement Logic*

In this section, we will use an example (Figure 5a) to explain the subblock compression and rearrangement logic for a cache entry divided into four subentries (Figure 5b). The goal of this circuit is to match non-null subblocks with non-faulty subentries. Let us suppose that an incoming block with one null subblock has to be inserted at a cache entry with one faulty subentry. The null subblock detector, implemented with logic that performs an OR operation on each of the subblocks, generates the CM for the incoming

block, indicating the location of null subblocks. The FM indicates the location of the faulty subentries of the given cache entry.

First, the CM and the FM are processed to assign increasing identifiers to each of the non-null subblocks and for each of the non-faulty subentries, respectively. Thereby, compression and fault vectors of identifiers (CI and FI) are generated. As shown in the dotted boxes in Figure 5b, this processing consists of adding the corresponding bits of the CM and FM, always using a zero for the first bit. In the general case, the bit width of each CI and FI entry is $log_2(N)$, N being the number of subentries (subblocks) in a cache entry (block).

Next, each non-null subblock will be assigned to the non-faulty subentry whose generated indexes match. A crossbar-like logic controlled by CM, CI, FM, and FI vectors implements this matching (see Figure 5b). The incoming block and the CM and CI vectors are input, on the left, and the FM and FI vectors, on the top. A crosspoint is activated only if both of its corresponding CM and FM bits are set; i.e., non-null subblock and non-faulty subentry, and if both CI and FI indexes are equal. This activation logic requires N^2 comparators of $log_2(N)$ bits, and N^2 tri-state buffers. For example, a 4-subentry cache requires 16 comparators of 2 bits. When the subentry size decreases (finer granularity), both the number of the comparators and the size of their inputs increase (cache entries with 8 subentries require 64 comparators of 3 bits, etc.), but overall, the overhead is still low.

A similar logic is used to reconstruct a compressed cache block. Decompression is on the critical path, but Concertina's decompression stage is fully parallel and, due to its simple logic, we estimate that it can be performed in one extra cycle.

5.1.3 Writeback Handling

A common problem for cache compression mechanisms is handling writebacks. Compression techniques aiming to increase the effective cache capacity have internal fragmentation issues, because several blocks are stored in a single cache entry. Thus, blocks might need to be shifted properly before insertion [2], [34]. This issue can be overcome by using sparse super-block tags and a skewed associative mapping [36]. Nevertheless, Concertina does not have this problem, as only one block is stored per cache entry. On a writeback event, the block is compressed again. If it fits in its cache entry, the entry is updated; if it does not fit, the block is written back to memory. On average, we found that less than 2% of the writeback blocks are evicted because they do not fit in the same cache entry after writeback.

5.2 Replacement Policy for Concertina

Existing cache management policies rely on the assumption that a block can be stored in any entry of a cache set. However, caches with faulty cells have entries with a variable number of defective subentries, which can store only blocks that are compressed at least to a certain size. To overcome these stricter size requirements, Concertina implements a compression- and fault-aware replacement policy that takes advantage of the various degrees of compression that a block might have, and is able to evenly distribute all the blocks (both compressed and uncompressed) across the available cache capacity (both defective and non-defective entries). In the rest of this section, we will assume a baseline least-recently used (LRU) replacement policy to simplify our explanations. Nevertheless, the algorithms described herein could work with other replacement scheme.

We present two replacement policies: *Best-Fit LRU* and *Fit LRU*. The first tries to find a precise match between cache entries and blocks in terms of the number of faulty subentries and the number of compressible subblocks. That is, once the block is compressed, the replacement algorithm searches for a victim among the entries whose effective size (number of non-faulty subentries) matches the size of the compressed block. The latter builds upon the first, but relaxes the matching condition, searching among all the cache entries where the block fits.

Let X_i be the group of blocks with i null subblocks of a given size, and E_j the group of entries with j faulty subentries of the same size, $0 \leq i, j \leq n$, where $n = \frac{entry\ size}{subentry\ size}$ (e.g., $n = 4$ for 64-byte cache entries and 16-byte subentries, as we might find entries with $0, 1, \ldots, 4$ faults). Therefore, *a block $x \in X_i$ can be allocated to an entry $e \in E_j$ if and only if $j \leq i$, but it cannot be allocated if $j > i$.* This premise is key for our replacement strategies.

A straightforward fault- and compression-aware allocation policy would try first to pair blocks and entries of the same index group, i.e., allocate blocks of type X_i to entries of type E_i. In the event that there is no entry of type E_i, it searches for entries of type $E_j, j < i$, until it finds a non-empty group. If the selected group has several entries, i.e., several entries have the same number of faulty subentries, the LRU information is used to select a victim among them. We call this the *Best-Fit LRU* policy, because it allocates blocks to the entries that best fit the size of the compressed block.

The Best-Fit LRU policy assumes that the distribution of blocks and entries of a given index group is well-balanced, but as we saw in Table 1 and in Figures 2 and 3, different cells and different workloads have different characteristics. For example, roughly 80% of the blocks in `wrf` contain 64 null bytes (X_n), while the majority of cache entries have between 0 and 4 faults, irrespective of the cell type. To compensate for the uneven distribution between entries and blocks, we can follow a different strategy and allocate a block X_i to any entry E_j where it fits, i.e., $j \leq i$. In this case, the victim block is selected using the LRU information about all the entries that belong to the E_j groups. We call this second policy *Fit LRU*, because it allocates blocks to entries where they fit, even if the allocating entry is not the best fit for a given block.

Note that both replacement policies make use of all the cache entries and allocate every block to the LLC. For instance, in the case where the percentage of non-compressible blocks (X_0) is higher than the percentage of non-faulty entries, blocks that cannot be compressed fight for non-faulty entries and, therefore, the pressure on the non-faulty entries is greater than the pressure on the faulty ones. In other words, although all the cache entries are utilized, in this example, the probability of a block of being replaced in a certain temporal interval is higher for blocks allocated to non-faulty entries than for blocks allocated to faulty ones.

It might be the case that a given block cannot be stored in

its cache set, because the block size is larger than any of the available entries (e.g., a block cannot be compressed and all the entries have at least one faulty subentry). To overcome this potential problem, we do as follows. When the block arrives to the LLC from main memory, it is sent to the L1 cache as usual, but the coherence state of the block in the LLC specifies that the content of the block in the LLC is not valid. Future requests to that block cannot be satisfied by the LLC, and they will be forwarded to the L1 cache. This functionality already exists in the coherence protocol to forward exclusively owned blocks. After the L1 replacement, if the block still does not fit in the LLC entry, it is written back to memory.

Certainly, more complex policies are feasible. It could be expected that starting from the distribution of blocks and entries observed and adjusting the allocation of blocks according to the frequency of each block type would lead to more promising trade-offs. Unfortunately, after thoroughly evaluating such a strategy, we obtained similar results to those obtained with the Fit LRU policy, the latter being much simpler and with lower storage requirements.

5.2.1 Replacement Policy Implementation

Both Best-Fit and Fit LRU replacement algorithms must be able to select the LRU element among a subset of the blocks in a cache set. Two hardware modules implement these actions: **Definition of the subset of candidate blocks**: the null subblock detector generates the CM of the incoming block. A hardware bit-counter computes the number of zeros in the CM. This value (CM_{count}) is the number of null subblocks in the incoming block, and is used to classify the block into a given index-group X_i. Likewise, a group of hardware bit-counters compute the number of zeros in the FM for all the cache entries in the cache set, and the result is a vector, FM_{count}, whose elements indicate the number of faulty subentries in each cache entry. These values (FM_{count_k}) are used to classify the entries into E_j groups. The comparison between each FM_{count_k} and CM_{count} generates a bit vector that represents the subset of candidate blocks to replace according to the fit criteria (Best-Fit or Fit).

LRU selection in the subset: our mechanism maintains the LRU order among all the elements in a cache set. The replacement algorithm applies the LRU selection logic to the subset of candidate blocks [19].

6 Methodology

6.1 Overview of the System

Our baseline system consists of a tiled chip multiprocessor (CMP), with an inclusive two-level cache hierarchy, where the second level (L2) is shared and distributed among eight processor cores. Tiles are interconnected by means of a mesh. Each node has a processor core with a private first level cache (L1) split into instructions and data, and a bank of the shared L2, both connected to the router (Figure 6). The CMP includes two memory controllers located on the edges of the chip. Table 3 shows the parameters of the baseline processor, memory hierarchy, and interconnection network.

We assume a frequency of 1 GHz at 0.5 V (our target near-threshold V_{dd}). Note that the DRAM module voltage is not scaled like the rest of the system. Thus, the relative speed of main memory with respect to the chip gets faster as the voltage decreases. This model is consistent with prior work [5], [39].

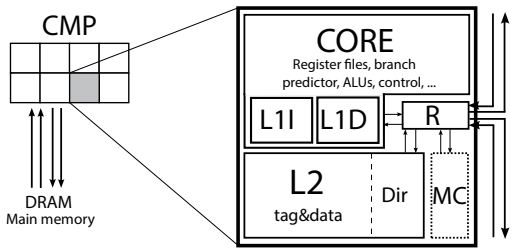


Fig. 6. Modeled 8-core CMP.

The 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 and, therefore, they can run at lower voltages without suffering the effects of parameter variation, while L2 data banks are built with conventional 6T SRAM cells and, therefore, they are sensitive to failures [25]. As in previous studies [5], [39], we assume that the LLC tag arrays are hardened through upsized cells such as 8T [10]. The same applies for the fault and compression metadata in Concertina.

6.2 Experimental Set-up

Regarding our experimental set-up, we use Simics [29] with GEMS to model the on-chip memory hierarchy and interconnection network [32], and DRAMSim2 for a detailed DDR3 DRAM model [35]. For timing, area, and energy consumption estimations, we rely on the McPAT framework [28].

We use a set of 20 multiprogrammed workloads built as random combinations of the 29 SPEC CPU 2006 programs [18], with no special distinction between integer and floating point. 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. To ensure that no application was in its 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 create random fault maps and run Monte Carlo simulations to ensure results are within an error of 5%, and a confidence level of $(1-\alpha) = 0.95$[3]. To ensure all simulations had similar numbers of faults but at different locations, we compute the faultiness of each memory cell randomly and independently of other cells.

7 Evaluation

In this section, we present the evaluation results for Concertina. We first analyze the different design options for Concertina in terms of LLC misses per kilo instruction (MPKI).

3. The number of samples was increased as necessary to reach the target error within the given confidence level.

TABLE 3
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 L2 cache banks
Consistency model	Sequential
L1 cache	Private, 64 KB data and inst. caches, 4-way, 64 B line size, LRU, 2-cycle hit access time
L2 cache	Shared, inclusive, interleaved by line address, 1 bank/tile, 1 MB/bank, 16-way, 64 B line size, LRU 8-cycle hit access time (4-cycle tag access), 1 cycle extra latency decompression stage (compressed cache lines)
Memory	2 memory controllers, distributed on the edges of the chip; Double Data Rate (DDR3 1333 MHz) 2 channels, 8 Gb/channel, 8 banks, 8 KB page size, open page policy; raw access time 50 cycles
NoC	Mesh, 2 Virt. Networks (VN): requests and replies; 16-byte flit size, 1-cycle latency hop, 2-stage routing

Then we explore a new implementation that reduces the storage requirements and evaluate the Concertina overhead. Finally, we compare our proposal with prior work.

7.1 Replacement Policy

Here we explore the impact on the LLC MPKI of the replacement policies proposed in Section 5.2: Best-Fit LRU and Fit LRU, with respect to an unrealistically robust cell, i.e., a cell operating at an ultra-low voltage but with no failures and no power or area penalization.

Figure 7 shows the increase in LLC MPKI with respect to the unrealistically robust cell for both strategies and different subentry/subblock sizes (8, 4, 2, and 1 byte), for cells C4, C3, and C2. The Best-Fit LRU strategy needs an even balance between the distribution of blocks X_i and entries E_i. However, as the distribution is usually not balanced, this replacement strategy fails to exploit the full potential of Concertina. In general, the smaller the subentry size, the poorer the performance. This happens because there are more non-empty block groups (X_i) than non-empty entry groups (E_j): for example, considering 1-byte subblocks, there are 65 X_i groups, most of them likely to be non-empty, while due to the small number of faults per cache entry (Figure 2), the number of non-empty groups E_j will be around 5 ($E_{0\ldots4}$), which increases the pressure on the E_j groups with the higher number of faulty entries (E_3 and E_4).

If we focus on the Fit LRU replacement strategy, Figure 7 reinforces the intuition that the smaller the subentry/subblock size, the smaller the increase in MPKI. The best configuration is 1-byte subblock size, irrespective of the cell size, because we recover more capacity and fit a larger number of compressed blocks in faulty entries (compression coverage increases). In comparison with the unrealistically robust cell, there is no MPKI increase when using C4 or C3 cells, and the number of MPKI increases only by 3% when C2 cells are used. However, for the smaller subblock sizes, the overhead is intolerable in terms of area: the bit map for implementing 1-byte subblock size requires 128 bits (64 bits for the CM and 64 bits for the FM), which corresponds to 25% of the 64-byte cache entry. In the next subsection, we will explore alternatives to significantly reduce storage overhead with a small impact on performance. From now on, we will use the Fit LRU policy.

7.2 Reducing Concertina Storage Requirements: Implementation with Pointers

The fault and compression metadata (FM and CM) can be encoded in several ways. As stated in Section 4.1, even for the smallest subentry size and the smallest cell, only 7.6% of the cache entries have more than four faulty subentries, which leads us to propose a new implementation based on pointers. For the sake of brevity, we only detail the implementation for the FM, but the same applies to the CM.

Instead of using a bit map, we could use four pointers to identify faulty subentries. Entries with more faults than pointers are disabled, because there is not enough space available to store the required fault information, reducing fault coverage. However, as the storage requirements are still high, we also consider implementations with three and two pointers.

Let us consider an example with 8-byte subentry size and three pointers, each of three bits. Specifically, the three pointers 000-011-101 are equivalent to the bit map 11010110. Although the maximum number of pointers is fixed per cache entry, not all of them are always valid (an entry may have fewer faulty subentries than the three available pointers). To overcome this issue, we can store redundant pointers: the pointers 000-011-011 are equivalent to the bit map 11110110, which represents a cache entry with only two faulty (0) subentries. One extra bit is needed to distinguish between an entry with one fault—000-000-000-(1)—and a non-faulty entry—000-000-000-(0). Finally, another bit indicates whether the entry has more faults than the number of available pointers. Thus, the storage required by this pointer-based implementation (fault and compression metadata) is $2 \times (number\ of\ pointers) \times log_2(number\ of\ subblocks) + 2$ bits per cache entry.

Table 4 compares the storage and fault coverage values of both approaches (bit maps and pointers). We can conclude that using three to four pointers per cache entry offers a good trade-off between storage overhead and fault coverage. For C4 and C3, three pointers would suffice to cover more than 96% of the cache entries. Conversely, due to the higher probability of SRAM cell failure, C2 would require four pointers per cache entry for a fault coverage of at least 92%. If we seek a lower overhead implementation, two pointers would cover around 90% of cache entries for C4 and C3, but the coverage would drop to 60% for C2.

In terms of the design of the compression/decompression and rearrangement mechanism, the pointer proposal only affects the processing of the FM and CM. After recovering the pointer information from the corresponding metadata structure, a small set of decoders can transform these pointers to bit maps, leaving the compression/decompression, rearrangement, and replacement logic unaffected.

The drawback of the pointer-based approach is that all entries with more faults than the number of available pointers

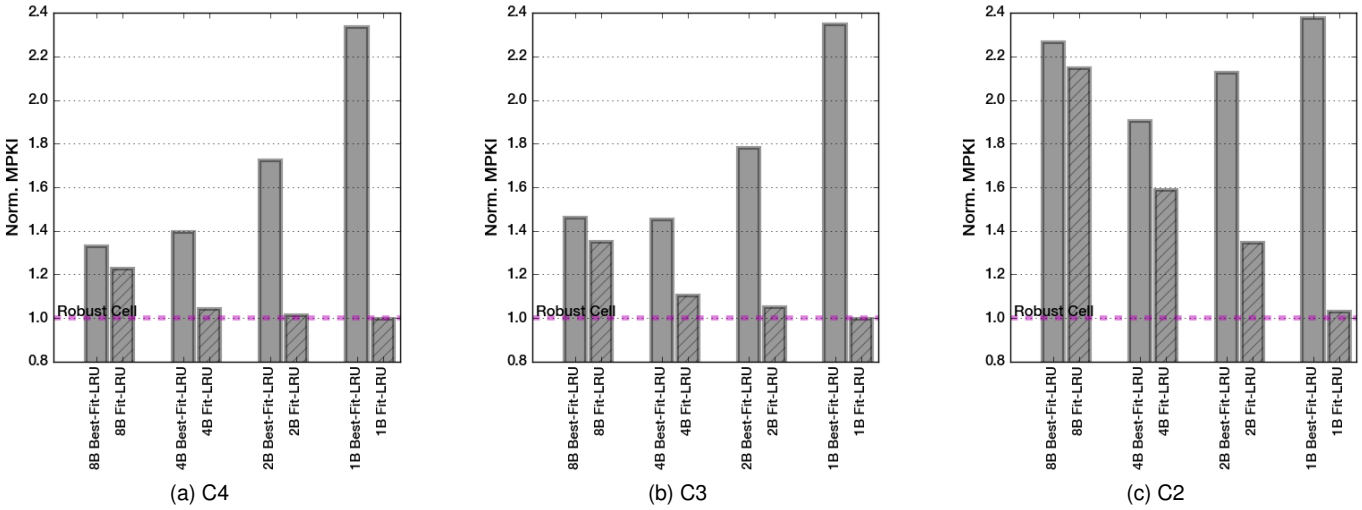


Fig. 7. LLC misses per kilo instruction (MPKI) relative to an unrealistically robust cell, for the different replacement alternatives and subblock sizes.

TABLE 4
Concertina storage requirements (bit map vs. pointer implementation) and potential fault coverage for cells C2, C3, and C4. The fault coverage for the bit map approach is always 100%.

Subblock size (bytes)	Storage Requirements (bits)				Fault Coverage (% entries)								
					C4			C3			C2		
	Map	4 Ptrs	3 Ptrs	2 Ptrs	4 Ptrs	3 Ptrs	2 Ptrs	4 Ptrs	3 Ptrs	2 Ptrs	4 Ptrs	3 Ptrs	2 Ptrs
8	16	26[a]	20[a]	14	≈ 100	99	94	≈ 100	98	90	97	88	68
4	32	34[a]	26	18		98	93		97	88	94	84	63
2	64	42	32	22		98	92		96	87	93	82	61
1	128	50	38	26		98	92		96	87	92	81	60

[a] The storage overhead is greater using pointers than a bit map.

have to be disabled. Nevertheless, these entries can store whole null blocks with neither extra complexity nor storage overhead, enabling 100% of the LLC capacity. The bit used to mark that the entry has more faults than the available number of pointers now indicates that the cache entry, if valid, contains a null block (all zeros).

7.2.1 Pointer-based Implementation Evaluation

Figure 8 shows the LLC MPKI increase for cells C4, C3, and C2 . Results are showed for different granularities (8, 4, 2, and 1 byte) and for both implementations: map and pointers (considering 2, 3, and 4 pointers per entry). For simplicity, we will follow a *subentry-size implementation* name convention, e.g., *1-byte 3-pointer* refers to subentry size of 1 byte, implemented using 3 pointers. We will only consider design choices that reduce the storage requirements, because in terms of performance the implementation based on maps will always be preferable.

We find that pointers are consistently a good alternative to bit maps. We obtain results within 1-2% of those obtained with bit maps for cells C4 and C3 and greatly reduce the storage requirements (e.g., from 64 to 32 bits for the *2-byte 3-pointer* configuration). In the case of C2, the high percentage of defective cells compels Concertina to use 1-byte subblocks. In this context, the pointer-based approach significantly reduces the implementation overhead, from 128 bits to 50, 38, and 26 bits, at the cost of MPKI increases of 5%, 11%, and 26% for 4, 3, and 2 pointers, respectively.

7.3 Concertina Overhead

Concertina's main overhead is the storage of the FM and CM metadata. We saw that the use of pointers greatly reduces the storage requirements with respect to bit maps. Here we evaluate the impact in terms of latency, area, and energy of the added storage for the pointer-based implementation. We consider the configurations with the best performance results: 1-byte 4-, 3-, and 2-pointer configurations. To quantify these costs we use CACTI [28].

Assuming the cache organization of Table 3 and 40 bits of physical address in a 64-bit architecture, a data array entry has 512 bits, while a tag array entry has 39 bits: 21-bit tag identifier, 14 bits for coherence information (5-bit coherence state, 8-bit presence vector, and 1 dirty bit), and 4 bits for the replacement related information (LRU implementation of $log_2(associativity)$ bits per cache entry). Concertina adds 50, 38, and 26 bits per cache entry for 4, 3, and 2 pointers, respectively. Although the increase in storage requirements with respect to the tag array is significant, the impact is small with respect to the whole LLC structure. Table 5 summarizes the overheads for 4, 3, and 2 pointers in terms of area, latency, and leakage with respect to the whole baseline cache structure. We also show the dynamic energy consumption for a cache read access to a non-faulty entry, and a cache read access to a faulty entry. However, static energy dominates the overall consumption of the LLC, even more when operating at voltages near V_{th} [20]. Hence, the observed variations in the dynamic energy have no significant impact on the total consumption of the system.

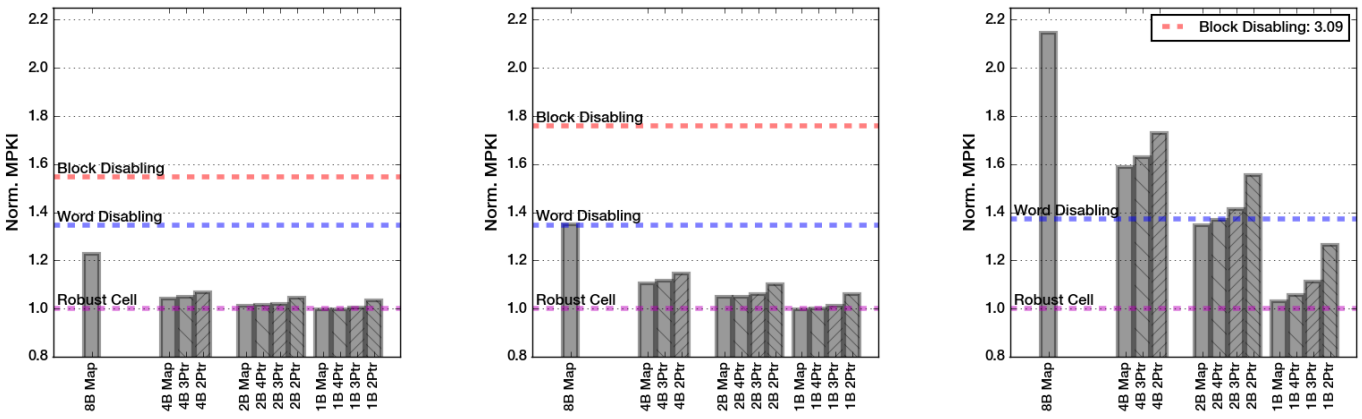


Fig. 8. LLC misses per kilo instruction (MPKI) for C4, C3, and C2 cells, and the different compression alternatives and subblock sizes.

The FM and CM can be organized in different ways, and these offer different trade-offs. As a first approach, both FM and CM are integrated into the tag array. For 3 pointers, area and leakage increase 5.4% and 5.9%, respectively, with respect to the baseline cache structure. After a tag hit, the logic for decompression can be pre-charged with the metadata. On every tag access, we are reading extra information that is only useful in case of a cache hit, which translates to an increase in the tag array latency of 14%, 10%, and 7% for 4, 3, and 2 pointers, respectively. If there is not enough slack available, this design would increase the tag array latency, penalizing every cache access. Nevertheless, a one cycle latency increase in the tag array access would have a minor impact in the overall performance (less than 0.2% performance drop).

An alternative design stores the FM and CM in separate SRAM structures. In this case, a bit in the tag array indicates if the entry is faulty and, therefore, the FM and CM must be accessed. The metadata is now only accessed in case of a tag hit to a faulty entry. The decompression logic can still be pre-charged before the data block is available, because the access to the metadata arrays is faster than the access to the data array. For 3 pointers, area and leakage increase 0.8% and 0.3%, respectively, with respect to the first approach. At the cost of slight area and static power penalties, the separate design avoids the extra latency to access the tag array. Besides, storing the metadata in separate arrays makes it easier to power gate the structures in high-voltage/high-frequency modes, minimizing the impact on power and energy consumption.

In both designs, the power overhead is negligible compared to the reduction in both static and dynamic power obtained by working at ultra-low or near-threshold voltages (dynamic and static power scale quadratically and linearly with supply voltage, respectively), while the area overhead is negligible compared to the area savings obtained by using smaller SRAM cell transistors.

Regarding the compression/decompression logic, and with the *1-byte 3-pointer* configuration, we could take advantage of the restriction in the number of subblock shifts: 3 pointers mean that a given subblock can move a maximum of three positions to the right or three positions to the left.

TABLE 5
FM and CM overheads with respect to the cache structure.

Config.	Area	Latency	Leakage	Read Energy [a] (access/access-faulty)
Baseline	1.000	1.000	1.000	1.000/NA
FM and CM integrated in tag array				
1B 4Ptr	1.071	1.041	1.078	1.035/1.035
1B 3Ptr	1.054	1.030	1.059	1.027/1.027
1B 2Ptr	1.038	1.020	1.041	1.019/1.019
FM and CM separate structures				
1B 4Ptr	1.082	1.000	1.081	1.000/1.066
1B 3Ptr	1.063	1.000	1.062	1.000/1.051
1B 2Ptr	1.044	1.000	1.044	1.000/1.036

[a] With respect to a read access in the baseline configuration.

Thus, the original matrix of comparators becomes a band matrix with 7 elements per row, and a 7:1 multiplexer can substitute each column of tri-state buffers. The corresponding logic could be implemented with 436 6-bit comparators and 512 7:1 multiplexers, which roughly translates to 8 K logic gates. This overhead is insignificant, in comparison with the total LLC size: it represents approximately 0.01% of the data array of the on-chip LLC (i.e., without accounting for the tag array and the decoder logic). Since cache memories normally have zero slack, we assume an extra cycle in the read operation to decompress the cache blocks. Compression latency is hidden, as it occurs during the cache block allocation.

7.4 Comparison with Prior Work

Regarding the compression mechanism, at ultra-low voltage operation, our objective is to maximize compression coverage rather than compression ratio. State-of-the-art techniques aiming to increase the LLC capacity seek to maximize compression ratio, e.g., $B\Delta I$ has an average compression ratio of 1.5 for the SPEC CPU 2006 benchmarks, but its coverage is around 40% [34]. Any compression technique aiming to maximize coverage could be used, but we opted to use null subblock compression in Concertina given its simplicity and effectiveness.

Regarding the replacement algorithm, recent studies on cache block compression have underlined the importance

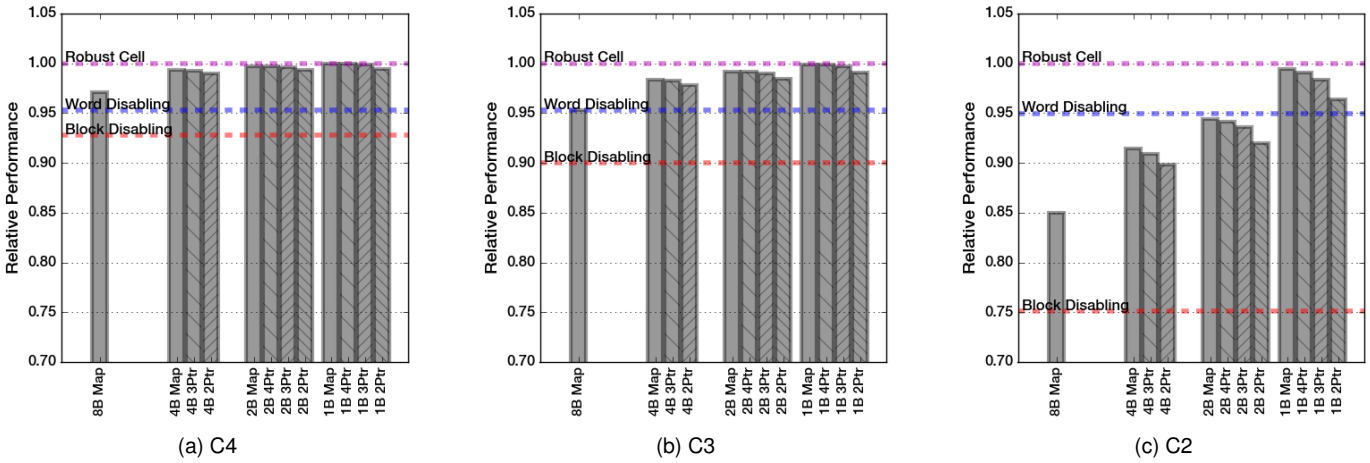


Fig. 9. Performance relative to a robust cell for cells C4, C3, and C2, and the different compression alternatives and subentry/subblock sizes.

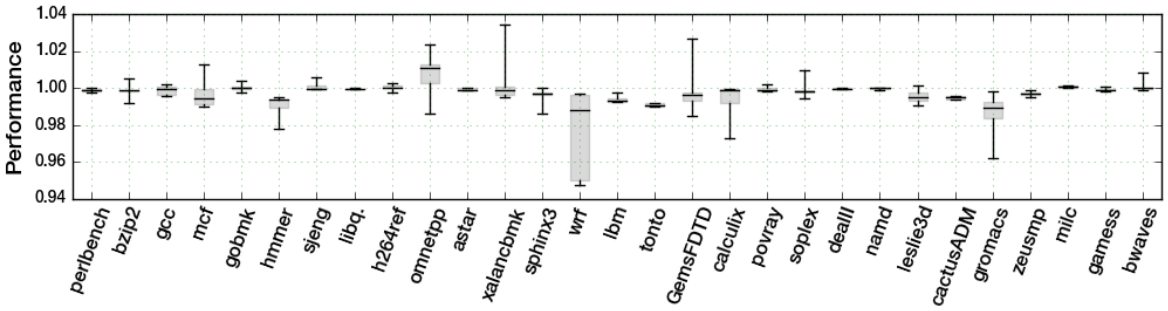


Fig. 10. Per-application performance analysis for the *1-byte 3 pointers* implementation (cell type C3).

of taking into account compression information for the replacement decision [6], [33]. These studies are based on the fact that several blocks are allocated to the same cache entry, while in the case of Concertina only one block is stored per cache entry. Thus, evicting a block with a given compression ratio does not directly translate into an increase in the number of blocks stored at the LLC.

ECCs are orthogonal to the Concertina design, and they should be implemented as well to protect against soft errors. Although ECCs have also been proposed to correct and detect hard errors, they require either large storage overhead or complex detection and correction logic. For example, by using orthogonal Latin square codes as in [14], the decoding latency would be comparable to Concertina, but at the cost of storing 138 check bits to correct 3 errors, while the *1-byte 3-pointer* implementation has a storage overhead of just 38 bits.

Combining techniques such as [5], [30] and [23] require complex algorithms to find collision-free cache entries across all the cache structure, and the storage of all the remapping strategy. Moreover, schemes that distribute blocks into several entries need to access two or more entries to recover a single block, with the corresponding latency and energy penalty.

We focus our detailed comparison on block and word disabling. Block disabling is implemented in modern processors operating at nominal voltages, and some authors advocate its use at lower voltages [26]; word disabling is similar to ours in complexity. Figure 8 shows the increase in MPKI for these techniques with respect to the unrealistically robust cell. We see that, irrespective cell size, Concertina always results in lower LLC MPKI values than both the other techniques. In fact, for C4 and C3, the implementation with least area overhead (*8-byte map*) produces fewer MPKI than word disabling. In the case of C2, the *8-byte map* implementation fails to reach the performance of word disabling, but the 1-byte implementations greatly improve on word disabling results.

8 System Impact

This section analyzes the impact of Concertina on the system in terms of performance (instructions per cycle or IPC), area, and power consumption. We compare Concertina with the unrealistically robust cell and the fault-tolerant mechanisms block and word disabling.

8.1 Performance

Figure 9 shows the performance relative to the robust cell for the two Concertina implementation alternatives (bit maps and pointers), and cells C4, C3, and C2. This confirms the *1-byte map* as the best design approach in terms of performance, achieving the same performance as an unrealistically robust cell for C4 and C3, and a minimal 0.5% degradation for C2. For C4 and C3, the pointer approaches have a performance within 1% of that of the bit maps, while it degrades up to 1%, 2%, and 3% for C2 when using 4, 3, and 2 pointers, respectively.

Performance follows the same trends as LLC MPKI regarding block and word disabling. Word disabling has a uniform performance across the cell sizes because there are usually fewer than four faults per cache entry, and hence it succeeds in combining consecutive entries to store a complete cache block. On average, it degrades performance approximately 5% with respect to the robust non-faulty cell. Block disabling impairs performance by 25% for C2 when the available cache capacity is scarce, and most of the LLC accesses become misses.

Concertina performance results are also consistent across all the programs considered. Figure 10 shows the distribution of IPC per application for the *1-byte 3-pointer* implementation (cell C3), with respect to the robust cell. For each application we show a boxplot with the minimum, first quartile, median, third quartile, and maximum of the distribution. Most applications show minimal performance degradation with respect to the robust cell: in the worst cases, the median represents a 1% performance degradation. In some cases, performance improves slightly, since applications with more compression potential (more blocks that can be compressed), such as `omnetpp` or `mcf`, have higher chances of finding an available entry where a block fits.

8.2 Area and Power

Larger SRAM cells have lower probability of failure, but at the cost of an increase in area and power consumption. Even with small cells (C2), Concertina provides performance comparable with an ideal cell. Even the largest cell considered by Zhou's study (C6), which increases the area by 41.1% relative to C2, does not accomplish fully functional performance: 40.1% of the cache entries are faulty at 0.5 V (Table 1). In comparison to this increase, the extra area required by Concertina (6.3% for the *1-byte 3-pointer* implementation) is a reasonable overhead, largely compensated for by its performance results.

Regarding power, our design focuses on the LLC, a structure where static energy dominates the overall consumption. Therefore, we estimate that the use of larger cells mainly affects the static energy of the LLC. I_{sub} increases with transistor width, and hence, based on the transistor widths of the cells considered [42], we estimate the increase in I_{sub} relative to C2. We assume that the unrealistically robust cell has the same dimensions as C2, but with a null probability of failure. We also add the dynamic and static overhead of the metadata computed in Section 7.3, assuming the separate design. Finally, we account for both the on-chip power and the power of the off-chip DRAM module.

Figure 11 shows the power consumption for Concertina's *1-byte 3-pointer* implementation, word disabling, and block disabling, with each of the cell sizes considered (C4, C3, and C2). Concertina is always the best option, very close to the ideal configuration in the case of C2. Concertina and word disabling follow a similar trend, the power consumption decreasing when downsizing the cell size. For block disabling, when the LLC is implemented using C2 cells, the large increase in off-chip DRAM traffic translates to a significant power consumption increase in the overall system.

The above results confirm Concertina to be an attractive LLC cache design to operate at ultra-low voltages, as it exhibits performance and power requirements comparable to a robust cell with a null probability of failure, even with a limited fault-free fraction of the LLC (9.9% when using C2 cells).

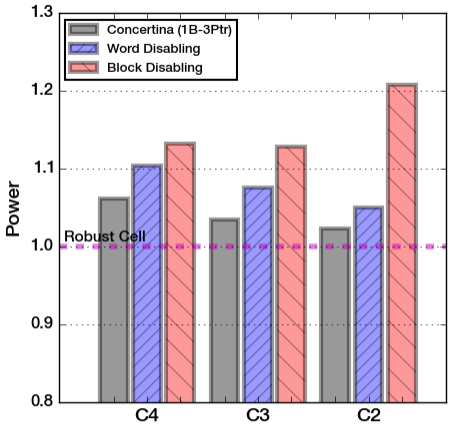


Fig. 11. System power consumption relative to the robust cell.

9 Conclusions

Scaling supply voltage presents a significant challenge to improving processor performance, especially for SRAM cell transistors used in cache memories. Lower voltages reduce cell reliability, which effectively reduces cache capacity, and, hence, performance.

Existing microarchitectural approaches to increasing cache reliability focus on enabling and combining the valid cells, thereby reducing the available cache capacity. For programs with large footprints and/or working sets, the extent of performance degradation is substantial. We depart from these approaches and propose Concertina, an LLC that compresses cache blocks to reduce their size in order that they can fit into entries with non-functional cells. Concertina ensures the use of 100% of the LLC capacity through a low overhead insertion/replacement policy that combines block compressibility and fault awareness to enable smart allocation of blocks to cache entries.

We explore three implementations with cells that trade off area and power for reliability, namely C4, C3, and C2. Concertina's *1-byte map* design exhibits a negligible MPKI increase relative to a defect-free LLC for C4 and C3 cells, and just a 3% MPKI degradation for the C2 implementation, but at the cost of large storage overhead. A lower-overhead design based on pointers (the *1-byte 3-pointer* configuration) greatly reduces Concertina storage requirements, with a minimal performance degradation of less than 1% for cells C4 and C3, and 2% for C2, with respect to the unrealistic defect-free LLC.

Acknowledgments

The authors would like to thank Javier Olivito for his support with logic prototyping, Víctor Viñals for his insightful contributions to this paper, and the anonymous referees for their helpful comments and suggestions. This work was supported in part by grants gaZ: T48 research group (Aragón

Gov. and European ESF), TIN2013-46957-C2-1-P, TIN2012-34557, and Consolider NoE TIN2014-52608-REDC (Spanish Gov.).

REFERENCES

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González, "Low Vccmin fault-tolerant cache with highly predictable performance," in *42nd IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 111–121.

[2] A. R. Alameldeen and D. A. Wood, "Adaptive cache compression for high-performance processors," in *31st Int. Symp. on Computer Architecture*, 2004, pp. 212–223.

[3] A. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, and S.-L. Lu, "Adaptive cache design to enable reliable low-voltage operation," *IEEE Trans. on Computers*, vol. 60, no. 1, pp. 50–63, Jan. 2011.

[4] A. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *38th Int. Symp. on Computer Architecture*, 2011, pp. 461–471.

[5] A. Ansari, S. Feng, S. Gupta, and 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.

[6] S. Baek, H. Lee, C. Nicopoulos, J. Lee, and J. Kim, "Size-aware cache management for compressed cache architectures," *IEEE Trans. on Computers*, vol. 64, no. 8, pp. 2337–2352, Aug. 2015.

[7] B. Calhoun and A. Chandrakasan, "A 256-kb 65-nm sub-threshold SRAM design for ultra-low-voltage operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, Mar. 2007.

[8] A. Chandrakasan, D. Daly, D. Finchelstein, J. Kwong, Y. Ramadass, M. Sinangil, V. Sze, and N. Verma, "Technologies for ultradynamic voltage scaling," *Proc. IEEE*, vol. 98, no. 2, pp. 191–214, Feb. 2010.

[9] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S.-L. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, pp. 846–852, Apr. 2007.

[10] L. Chang, R. Montoye, Y. Nakamura, K. Batson, R. Eickemeyer, R. Dennard, W. Haensch, and D. Jamsek, "An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 956–963, Apr. 2008.

[11] G. Chen, D. Sylvester, D. Blaauw, and T. Mudge, "Yield-driven near-threshold sram design," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 11, pp. 1590–1598, Nov. 2010.

[12] L. Chen, Y. Cao, and Z. Zhang, "Free ECC: An efficient error protection for compressed last-level caches," in *IEEE 31st Int. Conf. on Computer Design*, 2013, pp. 278–285.

[13] X. Chen, L. Yang, R. P. Dick, L. Shang, and H. Lekatsas, "C-Pack: A high-performance microprocessor cache compression algorithm," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1196–1208, Aug. 2010.

[14] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *42nd IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 89–99.

[15] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah, "A 22nm IA multi-CPU and GPU System-on-Chip," in *IEEE Int. Solid-State Circuits Conf.*, 2012, pp. 56–57.

[16] J. Dusser, T. Piquet, and A. Seznec, "Zero-content augmented caches," in *23rd Int. Conf. on Supercomputing*, 2009, pp. 46–55.

[17] H. Ghasemi, S. Draper, and 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.

[18] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[19] C. Johns, J. Kahle, and P. Liue, "Implementation of an LRU and MRU Algorithm in a Partitioned Cache," U.S. Patent US 6,931,493 B2, Aug. 16, 2005.

[20] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, "EnergySmart: Toward energy-efficient manycores for near-threshold computing," in *IEEE 19th Int. Symp. on High Performance Computer Architecture*, 2013, pp. 542–553.

[21] S. M. Khan, A. R. Alameldeen, C. Wilkerson, J. Kulkarni, and 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.

[22] S. Kim, J. Lee, J. Kim, and S. Hong, "Residue cache: A low-energy low-area L2 cache architecture via compression and partial hits," in *44th IEEE/ACM Int. Symp. on Microarchitecture*, 2011, pp. 420–429.

[23] C.-K. Koh, W.-F. Wong, Y. Chen, and H. Li, "Tolerating process variations in large, set-associative caches: The buddy cache," *ACM Trans. on Architecture and Code Optimization*, vol. 6, no. 2, pp. 8:1–8:34, Jul. 2009.

[24] J. Kulkarni, K. Kim, and K. Roy, "A 160mV robust schmitt trigger based subthreshold SRAM," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 10, pp. 2303–2313, Oct. 2007.

[25] R. Kumar and G. Hinton, "A family of 45nm IA processors," in *IEEE Int. Solid-State Circuits Conf.*, 2009, pp. 58–59.

[26] N. Ladas, Y. Sazeides, and V. Desmet, "Performance-effective operation below Vcc-min," in *IEEE Int. Symp. on Performance Analysis of Systems Software*, 2010, pp. 223–234.

[27] H. Lee, S. Cho, and B. Childers, "Performance of graceful degradation for cache faults," in *IEEE Computer Society Symp. on VLSI*, 2007, pp. 409–415.

[28] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd IEEE/ACM Int. Symp. on Microarchitecture*, 2009, pp. 469–480.

[29] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[30] T. Mahmood, S. Kim, and 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.

[31] W.-K. Mak and J.-W. Chen, "Voltage island generation under performance requirement for SoC designs," in *Asia and South Pacific Design Automation Conf.*, 2007, pp. 798–803.

[32] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[33] G. Pekhimenko, T. Huberty, R. Cai, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Exploiting compressed block size as an indicator of future reuse," in *IEEE 21st Int. Symp. on High Performance Computer Architecture*, 2015, pp. 51–63.

[34] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Base-delta-immediate compression: Practical data compression for on-chip caches," in *21st Int. Conf. on Parallel Architectures and Compilation Techniques*, 2012, pp. 377–388.

[35] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[36] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *47th IEEE/ACM Int. Symp. on Microarchitecture*, 2014, pp. 331–342.

[37] M. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013.

[38] L. Villa, M. Zhang, and K. Asanović, "Dynamic zero compression for cache energy reduction," in *33rd ACM/IEEE Int. Symp. on Microarchitecture*, 2000, pp. 214–220.

[39] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *35th Int. Symp. on Computer Architecture*, 2008, pp. 203–214.

[40] J. Yang, Y. Zhang, and R. Gupta, "Frequent value compression in data caches," in *33rd IEEE/ACM Int. Symp. on Microarchitecture*, 2000, pp. 258–265.

[41] R. Zahir, M. Ewert, and H. Seshadri, "The medfield smartphone: Intel architecture in a handheld form factor," *IEEE Micro*, vol. 33, no. 6, pp. 38–46, Nov. 2013.

[42] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and 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.

Alexandra Ferrerón (S'13) received the BS and MS degree in Computer Engineering in 2010 and 2012 from the Universidad de Zaragoza, Spain, where she is currently working toward the PhD degree in Systems Engineering and Computing. Her interests include high-performance low-power on-chip memory hierarchies, ultra-low and near-threshold voltage computing, and High Performance Computing. Ms. Ferrerón is a member of the Instituto de Investigación en Ingeniería de Aragón (I3A) and the European HiPEAC NoE.

Darío Suárez-Gracia (S'08, M'12) received the PhD degree in Computer Engineering from the Universidad de Zaragoza, Spain, in 2011. Since 2012, he has been working at Qualcomm Research Silicon Valley on power aware parallel and heterogeneous computing for mobile devices. His research interests include parallel programming, heterogeneous computing, memory hierarchy design, networks-on-chip, and processor microarchitecture. Dr. Suarez Gracia is also a member of the IEEE Computer Society and the Association for Computing Machinery.

Jesús Alastruey-Benedé received the Telecommunications Engineering degree and the PhD 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.

Teresa Monreal-Arnal received the MS degree in Mathematics and the PhD degree in Computer Science from the Universidad de Zaragoza, Spain, in 1991 and 2003, respectively. Until 2007, she was with the Departamento de Informática e Ingeniería de Sistemas (DIIS) at the Universidad de Zaragoza. Currently, she is an Associate Professor with the Departamento de Arquitectura de Computadores (DAC) at the Universitat Politècnica 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 (gaZ) from the Universidad de Zaragoza.

Pablo Ibáñez received the MS degree in Computer Science from the Universitat Politècnica de Catalunya in 1989, and the PhD 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.