Journal of Systems Architecture 91 (2018) 11–40

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

A survey of techniques for improving error-resilience of DRAM

Sparsh Mittal*, Maruthi S. Inukonda

IIT Hyderabad, India

ARTICLE INFO

Keywords:
DRAM
Reliability
Memory
Error correcting code (ECC)
Chipkill
Stacked DRAM
In-DRAM ECC
Data compression

ABSTRACT

Aggressive process scaling and increasing demands of performance/cost efficiency have exacerbated the incidences and impact of errors in DRAM systems. Due to this, improvements in DRAM reliability has received significant attention in recent years from both academia and industry. In this paper, we present a survey of techniques for improving reliability of DRAM-based main memory. We classify the works based on key parameters to emphasize their similarities and differences. This paper is expected to be useful for computer architects, chip-designers and researchers in the area of memory/system-reliability.

1. Introduction

Recent trends in memory design and operation have made DRAM memory increasingly susceptible to faults [1,2]. With ongoing feature-size scaling, process-variation has become severe, which leads to manufacture-time failures and variation in DRAM retention time [3,4]. Also, due to voltage scaling, even lower-charge particles, which are far more numerous in atmosphere, can flip the stored bit and cause a soft error [5]. Further, to mitigate the challenges of memory wall, stacked DRAM has been used [6]. However, stacked DRAM has higher integration density and hence, higher operating temperature compared to 2D DRAM [7]. Due to this, stacked-DRAM also shows higher fault rate than 2D DRAM [8]. Furthermore, since DRAM is widely used in data-centers and servers which support massive online services such as social media and e-commerce, the social and financial impact of DRAM errors today is likely to be more severe than ever before. These factors have underscored the need of improving DRAM reliability.

Improving DRAM error-resilience, although important, also presents several challenges. Design of ECC schemes involves an intricate trade-off between power, reliability and storage overheads. For example, ECCs with stronger protection generally also incur higher storage/latency overheads. ECC techniques must also account for data-access granularity and different criticality of different data-objects. Further, to avoid redundancy, synergistic integration of in-DRAM ECC with rank-level ECC is required [9]. Recently, several techniques have been proposed to address these challenges.

Contributions: In this paper, we present a survey of techniques for improving DRAM reliability. Fig. 1 shows an overview of this paper. In Section 2, we present a background on key terms and concepts. In Section 3, we underscore the motivation for and challenges in improving DRAM reliability. Further, we classify the works based on key parameters to give a bird's eye view of the whole field.

We discuss (rank-level) ECC architectures in Section 4 and techniques for reducing their overheads in Section 5. In Sections 6 and 7, we review techniques based on in-DRAM ECC (IECC) and those proposed for mitigating retention failures due to VRT, respectively. In Section 8, we discuss reliability techniques that do not use ECC, such as repairing, hashing and check-pointing. We conclude this article in Section 9 with a brief mention of future research challenges.

Scope: For sake of a concise presentation, we limit the scope of this paper as follows. We focus on reliability techniques proposed for DRAM-based main memory, and not for NVM based cache/main memory [10] or SRAM-based caches [11], although some of the ideas/techniques proposed for them may also be applicable for DRAM-based main memory. We do not include works where the reliability is compromised intentionally, e.g., due to low-voltage operation. Also, we do not review works that address row-hammer problem. Since different works use different evaluation platforms and procedures, we focus on their qualitative insights and include only selected quantitative results. We believe that this paper will be useful for computer architects, memory designers and researchers in the field of memory reliability.[1]

[1] The following acronyms are used frequently in this paper: bit error rate (BER), Bose-Chaudhuri-Hocquenghem (BCH), check-symbol (CS), chipkill (CPK), coarse/fine-grained (CG/FG), code-word (CW), column checksum group (CCG), cyclic redundancy check (CRC), detectable uncorrectable errors (DUEs), double data rate (DDR), dual in-line memory modules (DIMM), error detection (ED), error correction (EC), failures in time (FIT), floating-point (FP), Galois

* Corresponding author.
E-mail addresses: sparsh@iith.ac.in (S. Mittal), cs18resch01001@iith.ac.in (M.S. Inukonda).

https://doi.org/10.1016/j.sysarc.2018.09.004
Received 19 April 2018; Received in revised form 18 September 2018; Accepted 21 September 2018
Available online 22 September 2018

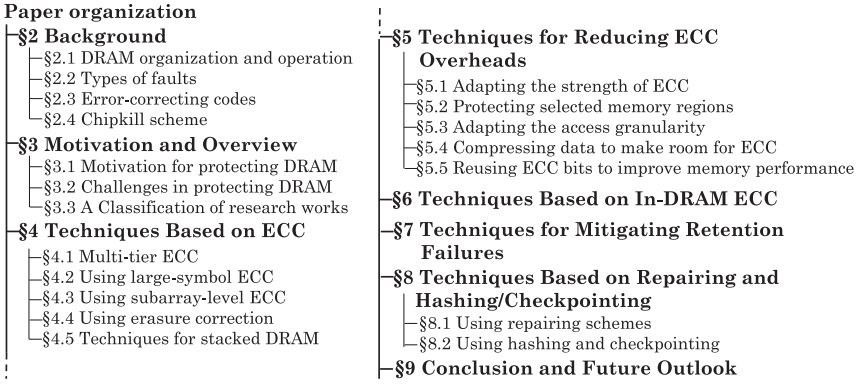


Fig. 1. Organization of the paper.

2. Background

We now introduce some concepts which will be useful throughout this paper. We refer the reader to previous works for a detailed background on DRAM architecture [12,13], chipkill schemes [14,15], and field data on memory errors [16].

2.1. DRAM organization and operation

DRAM has a deeply hierarchical architecture [12]. The DRAM cells are organized into a two dimensional structure called bank, which consists of multiple rows and columns. A DRAM device (also called a chip) has multiple banks. DRAMs have multiple "data pins" (DQs) for providing a parallel interface. An ×*N* chip (e.g., a ×4 or ×16 chip) has *N*-bit DQ interface, e.g., eight ×8 chips may be used to achieve a bus-width of 64b.

A rank is composed of the group of DRAM chips that are accessed concurrently for providing the target data bus-width. A DIMM consists of one or more ranks. The group of DIMMs which time-share command/address and data buses form a channel. For exploiting data locality and amortizing command overhead, a read/write operation transmits a block of data in multiple cycles: at a time, a rank transmits one 'beat' of data. Number of beats transferred in every access is termed as the "memory burst length".

Subranked memory: Subranked memories use less number of devices than the traditional DDR*x* memories and have a data-bus width of less than 64b [17]. For example, two ×8 devices may form a subrank which provides a bus-width of 16b.

Stacked DRAM: Recently, several prototypes for stacked DRAM have been proposed, e.g., high-bandwidth memory (HBM) [18], hybrid memory cube [19] and Octopus [20]. Although these prototypes use different data organizations and sharing of TSVs, their fundamental layout is similar.

2.2. Types of faults

Soft and hard faults: In charge-based memories such as SRAM and DRAM, striking of a charged particle to the memory cell can flip its state from '0' to '1' or vice versa. This is referred to as a soft (or transient) fault since only the data-value has been changed which can be corrected by ECC [5]. A hard or permanent fault refers to a faulty cell which may not store any value reliably, for example, due to manufacturing-time defects or getting stuck at a value (either 0 or 1) [21,22].

Inherent and operational faults: An inherent fault is a permanent fault which arises during manufacturing, whereas an operational fault, which can be either permanent or transient, occurs during system operation.

Errors and erasures: "Errors" refer to erroneous symbols at unknown locations, whereas erasures are errors whose location is known.

Error types based on their handling by ECC: Based on the handling of the errors by the ECC, they can be classified in four categories: (1) "detectable and correctable error" (DCE) (2) a "detectable but uncorrectable error" (DUE) (3) "detectable but miscorrected error" and (4) "undetectable and uncorrectable error". In case of misdetection or miscorrection, incorrect data propagates in the system, leading to "silent data corruption" (SDC).

2.3. Error-correcting codes

Parity bit: A parity bit, or check bit, is a bit added to a string of binary code to ensure that the total number of 1-bits in the string is even or odd. Parity bits are used for error-detection.

ECC codeword: An ECC word refers to the pair of data and check value [13]. A valid word for which the check bits are consistent with the data is termed as a codeword, otherwise, it is called a non-codeword.

Classifications of ECC: ECCs can be classified based on their symbol sizes. Binary ECCs use binary symbols and are primarily used for protecting against random bit errors. BCH and Hamming codes [23,24] can correct a single erroneous bit in K bits of data using $\lceil log_2(K+1) \rceil$ redundant bits; also they can detect one additional bit-error by using an extra parity bit.

Non-binary ECCs divide data into fixed-size symbols [13]. Then, ED and EC happen at the symbol granularity. Reed-Solomon (RS) codes [25] are the most frequently used non-binary ECCs and they achieve the least possible redundancy for a given distance [13]. RS codes with 8b symbols can correct up to t symbol errors using $2t$ redundant symbols (and can detect one more symbol error using an extra symbol) on up to 255-symbol (2040-bit) words. It is noteworthy that erasure decoding of RS codes can correct t symbol erasures using only t redundant symbols, doubling the correction efficiency compared to random error decoding [13].

In a Reed-Solomon code represented as RS(n, k) with s-bit symbols, k data symbols are encoded to codeword of length n by adding $n-k$ parity symbols of s bits each. From these codewords, up to t symbols that contain errors can be corrected, where $2t = n-k$. For example, for RS(255,223) with 8-bit symbols, $2t = 255 - 223 = 32$. So, up to 16 erroneous 8-bit symbols can be corrected.

Rank-level ECC and In-DRAM ECC: Conventionally, ECC has been used at rank-level, for example, an "ECC DIMM" may use an additional ninth chip for ECC compared to a non-ECC DIMM which has only eight chips. IECC, which is also called "on-die ECC", corrects scaling errors

field (GF), high bandwidth memory (HBM), hybrid memory cube (HMC), last level cache (LLC), memory controller (MC), Reed-Solomon (RS), silent data corruption (SDC), single error correction, double error detection (SECDED), single symbol correction double symbol detection (SSCDSD), through-silicon-via (TSV), variable retention time (VRT).

inside DRAM itself. With IECC, every DRAM chip has both storage and ECC logic embedded in it, and the DRAM external interface remains mostly unchanged. IECC encodes data written to DRAM and corrects it before transmitting it back to memory channel. IECC is used for handling errors internally within the DRAM chip without the knowledge of MC. For IECC, generally, 8b SECDED code is used for protecting 64b data. IECC helps in increasing the yield by allowing shipping of chips with inherent faults. IECC does not require generating a fault-map and also allows addressing newly occurring faults [26]. With increasing ECC encoding speed and access granularity per DRAM device, the latency and area overhead of IECC are reducing rapidly [4]. For example, an LPDDR4 DRAM design from Samsung reports the latency, area and power overhead of IECC to be 10%, 6.3% and 3%, respectively [27]. In this paper, unless otherwise mentioned, ECC refers to rank-level ECC.

2.4. Chipkill scheme

Background: Chipkill (CPK) is an IBM trademark for an ECC policy which protects against up to a full DRAM chip failure [28]. Any ECC technique which provides similar level of protection is said to provide CPK-level protection. CPK-correct protection is referred to as "single device data correction" (SDDC), extended ECC, and ChipSpare protection by Intel, Sun (now Oracle), and HP, respectively [29,30].

Motivation for using CPK scheme: Previous studies have shown that the large-granularity failures such as column/row/bank-failures occur nearly as frequently as bit-failures [31]. While protection against bit-failures can be achieved using an ECC-DIMM, protecting against large-granularity failures requires CPK-level protection. For example, compared to SECDEC, CPK-correct techniques can reduce uncorrectable error rate by 4X [32] to 36X [31]. Hence, CPK techniques can significantly reduce system outage due to DRAM device failures [33,34]. Due to these reasons, the low-end servers use SECDED scheme, whereas high-end servers (e.g., those in data-center and supercomputer) use CPK scheme.

Working of CPK scheme: In a CPK-correct system, every memory word is stored as a CW, which has the data word and redundant check bits. A CW is split into groups of bits termed "symbols", which are uniformly distributed across multiple DRAM devices of a rank. This distribution converts a single-chip failure into a single-bit error in every symbol. The CPK-correct techniques correct one bad symbol and detect two bad symbols in a CW using SSCDSD linear block code. The two widely-used SSC-DSD codes are 3 check-symbol and 4 check-symbol based codes [14,35].

SSCDSD codes, however, incur large overhead since they need at least three CSes per CW, and CSes are simply redundant symbols which do not store actual data value. To reduce this overhead, the number of data-symbols in the CW can be increased. For example, commercial CPK-schemes use 32 data symbols and 4 CSes in each CW, leading to a rank with 32 data devices and 4 redundant devices, for an overhead of 12.5%. This, however, requires accessing all the devices on each memory request, which leads to high power consumption. By contrast, SECDED uses only 1 redundant device out of 9 devices. Compared to a system with 36 devices per rank, a system with 9 devices per rank consumes nearly half the power [15].

Further, CPK schemes distribute the data across multiple channels which decreases the locality at the DRAM core. This harms performance by reducing rank-level parallelism and also reduces the effective memory bandwidth, by up to 40% [36]. Also, achieving CPK-protection within 12.5% storage budget requires use of x4 devices which consume nearly 30% higher energy than the x8 devices [14].

Some works provide double-chipkill protection [14,37,38]. Note that double-CPK can correct at most two failed devices if the two faults do not occur until one of them has been detected.

3. Motivation and overview

3.1. Motivation for protecting DRAM

Limitations of sparing schemes: Conventional sparing scheme uses redundant rows and columns which are used for replacing faulty rows, columns and cells. In this approach, tolerating a single faulty bit may require an overhead of nearly 10K-100K bits and many laser fuses [39,40]. Although this method works well for low BERs, it incurs excessively high latency, area and energy overheads for high BERs [9,39], and hence, it is ineffective in tolerating large-granularity failures. Further, if the number of redundant rows/columns are insufficient to replace all the faulty cells/columns/rows, the device needs to be discarded.

High DRAM error rates: Experiments on large-scale production systems have shown that unlike on-chip SRAM, DRAM of next-generation machines will require stronger resilience schemes than what is currently being used [1]. In fact, Sridharan et al. [1] note that SECDED ECC may lead to undetected errors at a rate of up to 20 FIT per DRAM device, which cannot be tolerated in the real-world systems.

Financial impact of memory reliability: Memory errors can lead to system crash and server downtime [32], which has high monetary impact. Conversely, with strong protection schemes (e.g., CPK), cheaper DRAM with much higher failure rate can be employed while still reducing the total cost of ownership [41].

3.2. Challenges in protecting DRAM

Protecting DRAM, although important, presents several challenges, as we show below.

Storage/energy/reliability tradeoffs: As discussed before, design of ECC involves trade-off between multiple metrics of interest. For example, to bound the storage overhead while increasing the number of ECC bits in a word, the ECC can be computed over a larger number of data bits. This, however, exacerbates memory power consumption since more number of devices need to be accessed on each memory request. The techniques which store ECC in caches require changes in OS and caches. The techniques which store fault location information require collecting this information on every boot-up, which incurs high overhead and precludes detection of newly occurring faults such as random telegraph noise [4] and VRT.

Factors in use of IECC: IECC generally incurs storage overhead between 6.5 to 12.5% and complicates internal DRAM operations [9]. With IECC, the device access granularity equals the CW size (= data + parity) [4]. This exceeds the external data transfer granularity and leads to data overfetch compared to the design without IECC. Also, it internally turns a write operation into read-modify-write operations for updating the whole IECC codeword [42]. This incurs performance/energy overheads especially in case of bursty writes. These overheads are in addition to the IECC encoding/decoding latency [4]. Further, since high-reliability systems also use rank-level ECC (RECC), the total redundancy (IECC + RECC) may exceed that required for a desired level of reliability.

Challenges in protecting stacked-DRAM: In addition to the errors shown by 2D DRAM, stacked DRAM also shows errors due to TSV failures, which results in multiple bit errors in a single cache line [43]. Further, for allowing integration of TSVs, stacked DRAM uses much leaner DRAM dies than those used in 2D DRAM [44]. Also, since DRAM dies and TSVs are made of different materials, they have different thermal expansion coefficients [44]. Due to these factors, the dies in a stacked DRAM are more prone to failure under thermal-mechanical stress [8,44].

Further, in stacked DRAM, a data block from LLC is stored in a single bank and not distributed across multiple chips as in 2D DRAM [45]. While improving performance and energy efficiency, this optimization harms reliability since failure of a bank corrupts the entire cache line [7]. Also, conventional ECC schemes which work on the assumption

Table 1
A classification based on type of vulnerability, reliability technique and ECC architectures.

Category	References
Type of vulnerability	
TSV failure	[43,45,50–52]
VRT-induced faults	[3,49]
Soft or hard faults	nearly all others
Type of reliability technique	
Checkpoint	[53,54]
Scrubbing	[49,55,56]
Guardbanding	[3]
Remapping scheme	[39,40,50,55,57]
Retirement scheme	[51,58]
ECC	nearly all others
ECC types and architectures	
Parity	[15,38,43,50,59–62]
Checksum	[15,46,62]
CRC	[50,59]
Reed-Solomon code	[9,14,45,46,51,63]
BCH	[9,16]
Erasure code	[9,45,46,63–65]
In-DRAM ECC (IECC)	[4,9,38,40]
Multi-tier ECC	[9,14,15,43,45,46,66]
Multi-dimensional ECC	[8,46,50]
RAID-like design	RAID-5 like [8,9,43,45,50,59,61,62,66,67], RAID-3 like [38]

of data being striped in different chips (e.g., [14,15,46]) may not be applicable for stacked DRAM.

Choice of right granularity: The memory access may be performed at either coarse or fine granularity, which brings different trade-off of reliability and performance/energy efficiency. CG accesses exploit spatial locality, reduce miss rate and incur lower average memory access latency. However, in case of poor locality, CG accesses waste bandwidth, power and on-chip storage resources. Converse is true for FG accesses. CG access reduces the overhead of ECC compared to FG access, since with FG access, each FG data block needs its separate ECC.

Retention time variations: The retention period of different DRAM cells varies greatly due to process variation [22]. To ensure reliability, DRAM systems generally use a refresh period which ensures that even the weakest cell is protected. However, this wastes large amount of power due to redundant refresh operations. To avoid this, different cells can be refreshed at different refresh periods [3]. However, such multirate refresh schemes rely on accurate profiling of retention period and also incur complexity/metadata overheads. Another strategy to deal with this issue is to increase the refresh period for saving energy and correct few weak cells using the reliability techniques [4,39,47,48].

Apart from the above manufacturing-time variation which leads to fixed retention period for a cell, the retention period of a cell may also change over time, which is termed as variable retention time (VRT) [3]. When the retention period goes below the current refresh period, the cell shows a failure. VRT reduces the yield and increases the testing cost significantly. Moreover, some DRAM cells begin showing VRT behavior on being exposed to high temperature while the DRAM chip is packaged. Hence, pre-packaging experiments cannot detect all VRT failures. Further, some cells may maintain high retention state for a large time before transitioning to low retention state. Detecting whether such VRT cells lead to failure may require very long tests. Due to these factors, the rate of error due to VRT may be up to 2700 times higher than that due to soft errors [49].

3.3. A Classification of research works

Table 1 first classifies the research works based on type of vulnerabilities. Evidently, while soft and hard faults have received the maximum attention, other errors such as TSV faults and VRT-induced faults are also getting increasing attention from researchers. Some techniques distinguish between the criticality of address-TSV and data-TSV to improve efficiency [50].

Table 1 further shows various resilience schemes. Each of these schemes have their pros and cons. While checkpointing can allow recovering from any number of bit-faults, it incurs high storage overhead and requires roll-back. As for scrubbing, it needs to be performed before the number of bit-errors exceed the correction capability of ECC. Guardbanding refers to adjusting the specification limits to account for failures/uncertainty in DRAM operation. The remapping scheme requires provisioning of spare memory and it complicates the address mapping scheme. The retirement scheme reduces the effective capacity of memory, still, it may be better than the ECC scheme in case of high fault-rates since ECCs for correcting multi-bit errors incur high latency. Finally, use of ECC involves various tradeoffs that are discussed in detail in this survey.

There are different types of ECC schemes, some of which are discussed in Section 2. As for others, multi-tier codes separate ED and EC operations to optimize them individually. Multi-dimensional codes maintain ECC in multiple dimensions, such that even two faults in a single dimension are unlikely to fall into the same block in the second (or remaining) dimension(s).

Table 2 first shows the storage overhead of different techniques. While most techniques have a storage budget of 12.5%, some techniques incur higher storage budget since they seek to improve on another metric, such energy efficiency and/or protection-strength.

As shown in Table 2, different designs have different number of devices in a rank. The schemes with less number of devices in the rank reduce the latency and power consumption, since on each access, less number of devices are accessed. However, they also incur higher storage overhead for providing reliability.

Table 2 further shows the memory configuration used by some techniques. The techniques proposed for stacked DRAM are discussed further in Section 4.5. Some works which use adaptive access granularity utilize sectored cache design in LLC to serve both CG and FG accesses. In a sectored cache, a cacheline is composed of multiple sectors, each of which has its own valid and dirty bits [75]. However, each cache line has only one tag, thus, the additional overhead of sectored cache is due to extra valid/dirty bits. Finally, Table 2 shows the works that seek to save energy while also improving reliability.

Table 2
A classification based on storage overhead, memory configuration used and, optimization target of different techniques.

Category	References
Storage overhead of ECC techniques	
12.5%	commercial CPK and [9,45,47,52,54,56,61,63,64]
Higher	3.1% to 25% (for different codes) [51], 12.9% [46], 14% [50], 14.1% [62], 15.7% [43], 18.75% [14], 26.5% [15], 14.3% or 30.6% (depending on the mode) [59], 50% [68]
Number of devices in a rank to achieve SSCDSD capability	
9	[15,46,63]
18	[14,61,63]
36	commercial CPK and other CPK designs
Memory configuration	
Techniques for stacked DRAM	[8,43,45,50,52,59,65,69]
Subranked memory	[60,61,67,70]
Sectored cache	[61,67,71]
Optimization target	
Reliability	All
Energy	[8,9,14,15,38,40,45–47,50,52,54–56,59,61–64,66,67,70–74]

4. Techniques based on ECC

In this section, we discuss techniques based on multi-tier ECC (Section 4.1), large-symbol ECC (Section 4.2), subarray-level ECC (Section 4.3) and erasure coding (Section 4.4). Finally, we discuss ECC techniques for stacked DRAM (Section 4.5).

4.1. Multi-tier ECC

Single-tier codes use the same redundancy resources for performing both ED and EC. It is noteworthy that although ED needs to be performed on each access, EC is required only in the rare case of an error [66]. Based on this observation, *the multi-tier code designs* decouple ED and EC and generally use the tier-1 code (T1C) for ED and tier-2 code (T2C) for EC. Decoupling of ED and EC allows versatile optimizations, for example, optimizing the latency of ED, even at the cost of increasing EC latency [43]. Similarly, ED is performed on each DRAM access, which now requires only part of the total ECC information (T1C) and hence, incurs less energy. Only for DRAM writes and the rare cases of detected errors, both T1C and T2C need to be accessed. We now discuss several multi-tier ECC designs.

Yoon et al. [14] present a technique for virtualizing ECC such that the storage of data is decoupled from the storage of its associated ECC. This is illustrated in Fig. 2. Thus, the storage location of ECC can be decided dynamically. Based on this, they propose a two-tier ECC which separates ED process from EC process. In their two-tier design, T1C is stored in the ECC chip. T1C can detect the errors but cannot correct them without the support of T2C. The T2C is stored in the data chips such that the data and its associated T2C are in two different DRAM ranks, which has the disadvantage of increasing the memory access energy. Different pages can have different degrees of protection, e.g., T2C is not required for clean pages.

In their technique, the cache-fill operation is same as that in a traditional cache, however, on eviction and writeback of a dirty line, its T2C information must be updated. For this, its data address it translated to the location of the ECC address. If the translated ECC address is present in LLC, it is updated, else, an LLC line is allocated to store T2C. On a write operation, new T2C is computed and is written into LLC along with a mask which shows the portion of LLC line that stores the T2C information. When the T2C line is written back to DRAM, this mask helps in ensuring that invalid portions of LLC do not overwrite T2C data in DRAM. Any error in T2C itself is ignored. To reduce the performance loss due to the need of accessing both T1C and T2C during write operations, ECC information can be cached in the LLC which is possible since ECC information is stored in the same physical namespace as the data. With non-ECC DIMMs, only a single tier of virtualized ECC is used which performs both ED and EC.

Their technique allows using 3-check symbol error code which is more efficient than the 4-check symbol error code. With the two-tier scheme, the first two and the third check symbols are T1C and T2C, respectively. T1C can detect at most 2 symbol errors. A single symbol error is corrected using all 3 check symbols of both the tiers. They use 8b symbols for x4 and x8 devices and 16b symbols with x16 devices. By adding a second check symbol to T2C, two faulty chips can be tolerated.

With x4 chips, only 2 ECC chips are required (for storing T1C), compared to the use of 4 chips in conventional schemes. This saves energy and bandwidth. With x8 chips, chipkill protection is achieved without increasing the access granularity or redundancy overhead compared to traditional chipkill scheme. Thus, their technique maintains the desired protection level even with different DRAM chips, e.g., x4, x8 and x16. Without virtualizing T2C, an extra DRAM-chip access would be required, which increases power overheads, redundancy and requires non-standard DIMMs. With non-ECC DIMMs, they use a 2-check symbol RS code which can detect and correct one symbol error. This code can correct any number of errors in a single chip. Their technique saves power while maintaining reliability and incurring negligible performance loss.

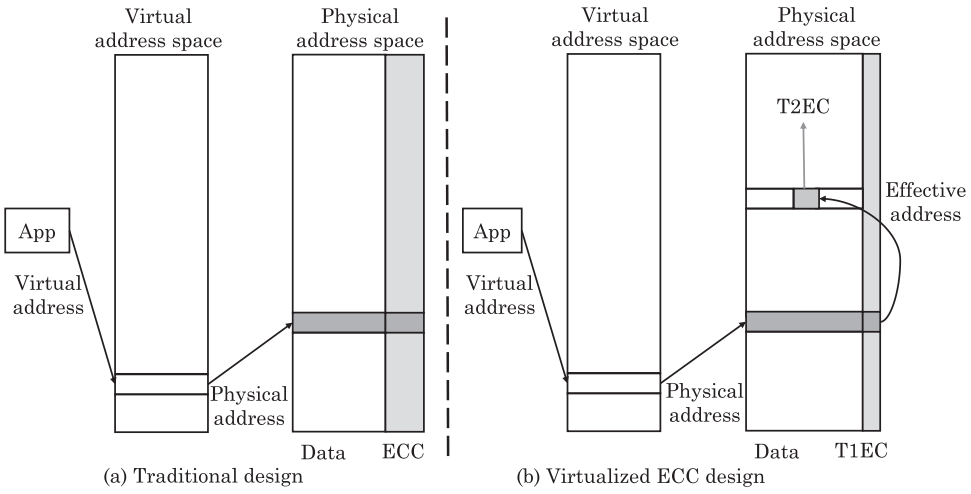


Fig. 2. Conceptual view of memory accesses in (a) traditional virtual memory with fixed ECC and (b) virtualized ECC [14] with a two-tier ECC scheme.

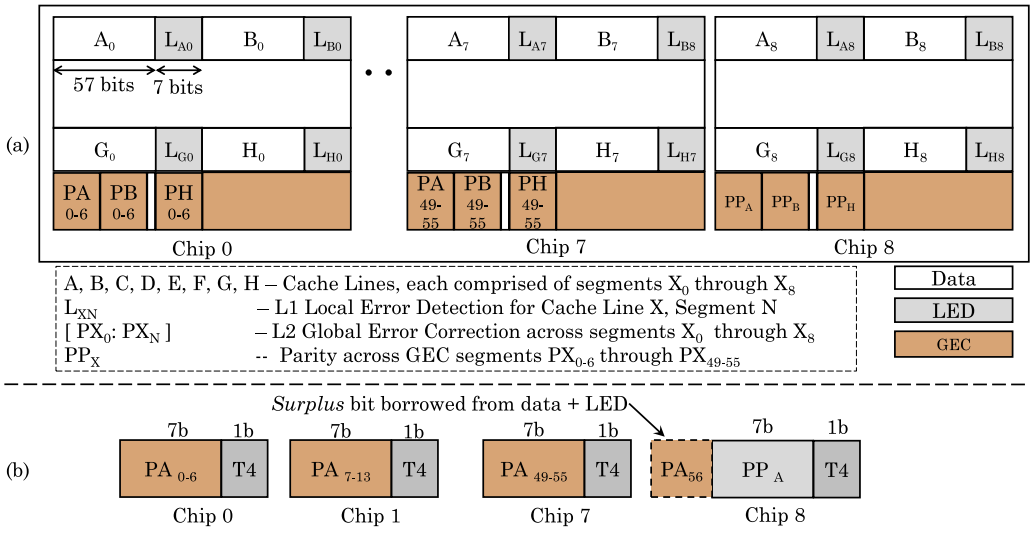


Fig. 3. (a) ECC design of Udipi et al. [15] for a single rank of nine x8 DRAM chips (b) data layout of GEC of one cache line in the red-shaded GEC region. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The limitation of their technique is that since the same pins are shared for transferring both data and ECC bits, the effective data bandwidth is reduced. Further, for read-requests that show errors, 36 device-accesses need to be performed since a second access to memory is required for retrieving the EC check-symbols if they are not found in LLC. On similar lines, for write accesses, 36 device-accesses need to be performed for updating the check symbols stored in memory if they are not present in the LLC.

Udipi et al. [15] present a reliability technique which divides the protection into multiple tiers of ED, location-identification and EC. The first level of protection is "local error detection' (LED), whereby data correctness is verified after every read operation. Further, the location of the fault in the rank is identified at chip-granularity. For this, LED information is tracked at chip-level corresponding to every cache line "portion" (the part of the line present in a single chip in the rank) and not the entire cacheline as done in the symbol-based ECCs. For an ECC rank with nine x8 chips, both data and LED information is stored in all nine chips. An x8 chip provides 64b in each access consisting of 57b data and 7b LED, as shown in Fig. 3(a). Thus, in nine chips, there are 513b of data, of which 512b are for data and one "extra bit" is used for the second level of protection (discussed below). Since LED is produced from the bits belonging to a single cache line, no additional reads/writes are required due to LED.

After the LED locates an error, the second level "global error correction" (GEC) helps in recovering the lost data. The second level has three tiers. The primary component is 57b columnwise XOR parity ("PX") of nine cache line portions. The data can be trivially reconstructed by XORing GEC code with error-free segments. The GEC code is stored in the same rank as its corresponding cache line, which reduces the number of chips consulted on each access. Also, it guarantees that the accesses to GEC bits lead to row-buffer hits when performed after access to the corresponding data-cache line. Thus, a 9KB row-buffer stores 8KB of data + LED and the remaining space is used for storing the corresponding GEC. Same as data bits, 57b parity ("PX") is spread across 9 chips: 7b each in the first eight chips and the last bit in the "extra bit" of the ninth chip mentioned above. This is shown in Fig. 3(b).

Since a chip-failure leads to loss of the GEC bits stored in it, GEC code is further protected by a third tier of parity, termed as "PPX". It is an XOR of eight 7b "PX" fields and is stored in the ninth chip, as shown in Fig. 3(b). On a chip-failure (found by LED), this parity along with the robust GEC portions from other chips are used for recovering the GEC. Using full GEC, the original data is reconstructed. When along with a fully-failed chip, there is an error in the second chip, their technique

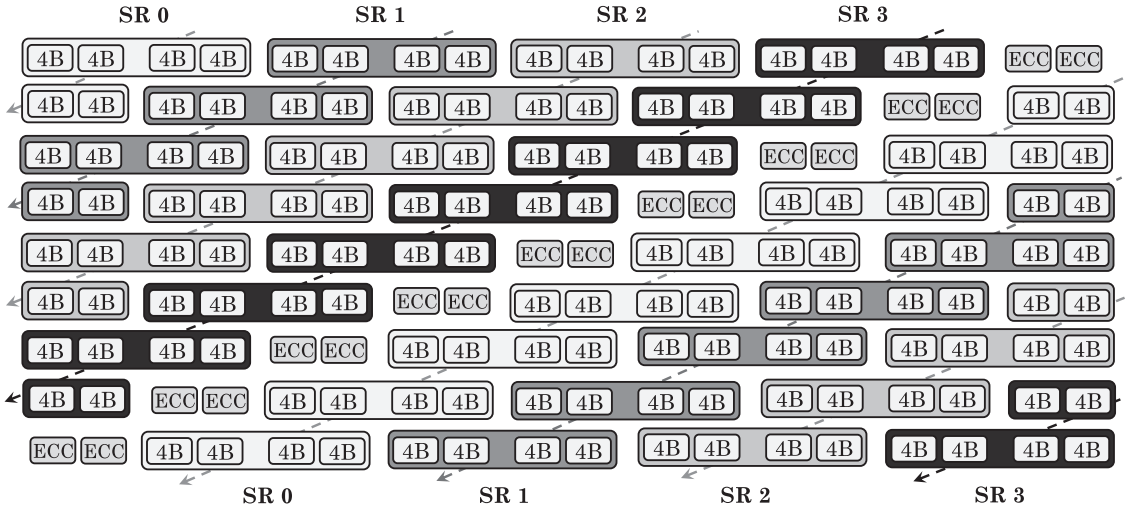


Fig. 4. The layout used by Gong et al. [61] for spreading ECC locations to reduce subrank conflicts.

can detect (if not correct) this failure. If the second error is failure of an entire chip, it is detected by LED and marked as uncorrectable double-chip failure. However, if the second error happens in the GEC region of another chip, reconstruction using PPX may be incorrect. To avoid this, the left-over 9 bits are used for building an ED code over GEC (PX and PPX) bits, which is termed as T4 (tier-4), as illustrated in Fig. 3(b). Since neither correction, nor exact error-location identification is required, T4 bits are formed using different permutations of bits from different chips.

Every cache line write leads to two writes, one to the data location (576b of data + LED + extrabit) and second to the corresponding GEC (72b of PX + PPX + T4). Since the write to GEC leads to a row-buffer hit, the performance impact of additional write is small, although they do consume significant bandwidth which is harmful for write-intensive workloads. To further reduce this impact, multiple GEC updates can be combined. In their technique, after failure of a device, every memory read requires a second access to fetch EC resources. The storage overhead of their technique is 26.5% (63b LED + 57b GEC parity + 7b PPX parity + 9b T4), of which 12.5% is provided by ninth chip and the remaining 14% is provided by the data memory.

The limitation of their technique is that compared to commercial CPK, it offers weaker guarantee of reliability since the one's complement checksums used for ED in their technique guarantee detection of device-level faults only if the output from a device with a device-level fault are all 1 or 0. However, device-level faults such as reading of an incorrect row/column due to faulty row/column address decoders cannot be detected by their technique.

Gong et al. [61] present a two-tier ECC technique for adaptive-granularity systems which can provide CPK-level reliability with CG accesses while enabling FG access to uncorrupted memory addresses. They assume memory ranks with 18 x4 DDR3 DRAM chips (16 for data and 2 for ECC) which has a storage overhead of 12.5%. They split the 18 chips into five effective subranks. Four subranks provide data at the granularity of at least 16B since every subrank has four x4 chips, as shown in Fig. 4. The fifth subrank has two x4 chips which store inner and outer code, respectively. Thus, 16 chips form four data subranks and 2 chips form ECC subrank.

To support both FG and CG accesses, they use sectored cache, where a 64B cacheline is divided into four 16B sectors. The granularity of individual DRAM accesses is determined based on spatial-characteristics similar to the technique of Yoon et al. [67]. EC needs to be performed for errors from a single chip. In every full rank and for every CG access, the outer chip stores the per-bit-location XOR of the 16 data chips. The outer code has 32 parity bits, one for every bit in a 32b burst, computed from 16 data chips. Correction simply requires XORing the data from 15 non-faulty data chips and the outer chip. For the outer code, a CG read needs to be performed irrespective of whether the error was detected on an FG access or not. This CG access before each partial writeback leads to high memory traffic. To reduce this overhead, they note that (1) the codes used in their technique are linear and systematic and hence, the updated ECC can be obtained from only the original ECC and the modified data (2) a partial writeback occurs when there are invalid sectors in the cache block. Hence, to avoid reading the whole rank or even ECC from memory before a partial writeback, the original ECC read at the time of the cache-fill is stored in an empty sector, as illustrated in Fig. 5(b). On writeback to a partial line, this ECC is updated and written back to memory along with the data, as shown in Fig. 5(c). Thus, compared to a CG read, shown in Fig. 5(a), an FG read avoids an additional memory access while writing back a partial cache block.

To allow independent FG access to multiple subranks, their technique provides inner codeword for each data sub-rank in the inner ECC chip. The 32b data (4b x8 beats) from every access to ECC chip is partitioned into four 8b symbols, each of which works as the check symbol for one sub-rank. The outer code can allow augmenting the error-coverage of inner code and avoid the risk of SDC. On a CG access, if the inner-code does not report any error, the outer-code information can be used for verifying the parity of each bit location in the access. If any single chip except the outer chip has an error, CG detection can always detect the error, regardless of the number of erroneous bits in the data. To find the faulty chip(s) in a subrank, they use additional ECC information of the outer code and its XOR correction scheme.

As for the overall flow, their technique checks each subrank accessed against its corresponding inner codes. If no error is detected and if the access was CG, errors are again checked with the outer code. On detecting an error, their technique tries to correct all likely faulty chips (based on the inner code) using the outer code. Every correction effort is checked against corresponding inner code and successful corrections are counted. After trying all the chips, the result is again verified using the outer code. If the outer code confirms a correct value and only one correction effort was successful, the conclusion is that the error is detected and corrected. If no correction effort was successful, but no error was detected by OC, it is concluded that the inner chip had an error, but there is no data error. If multiple correction efforts were successful or if the correction was not verified by the OC, it is an uncorrectable error. To reduce the number of SDCs, when the errors detected from a rank exceed a threshold, further accesses are forced to be CG. Their correction scheme is much simpler than the symbol-based CPK. In their technique, ED is performed for both CG and FG accesses, but EC is limited to when the whole rank is read. Compared to CPK, their technique

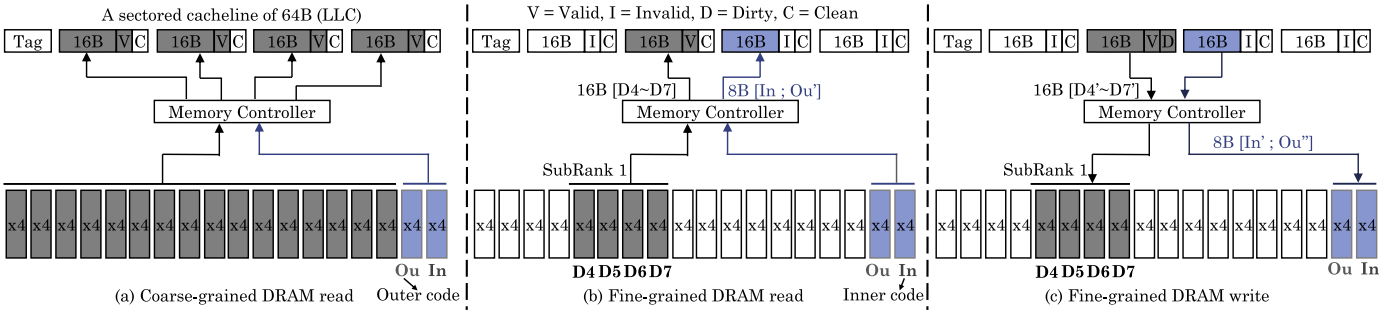


Fig. 5. Read/write operations in the technique of Gong et al. [61].

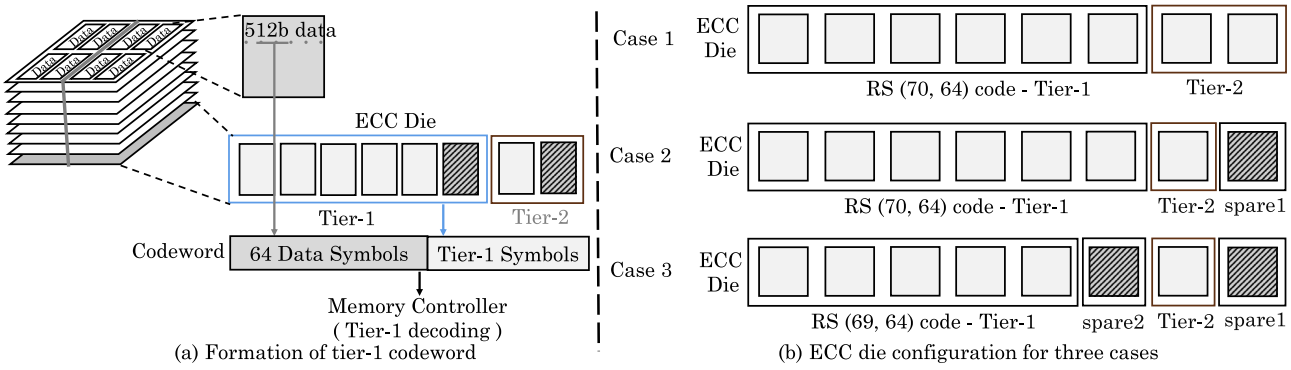


Fig. 6. (a) Formation of tier-1 CW in the technique of Chen et al. [45]. (b) The configuration of ECC die in the first three cases (see the text for details).

improves performance and power efficiency while maintaining similar level of reliability.

Chen et al. [45] present a two-tier ECC for improving reliability of HBM-type 3D DRAM system. The tier-1 code (T1C) corrects errors due to small granularity faults e.g., single-bit/column/TSV failures and detects errors due to large granularity faults e.g., row/bank failures. The tier-2 code (T2C) is an XOR-based code which corrects errors detected by the tier-1 code. To stay within 12.5% storage budget, they use only one additional ECC die for protecting eight data dies. Hence, both T1C and T2C should fit into eight banks in the ECC die. A strong T1C decreases the need for activating T2C and the rate of SDCs since T2C performs only EC and not ED. Hence, they allocate 6 and 2 banks for T1C and T2C, respectively. For T1C, they use RS(70,64) code over GF(2^8) and its parity is distributed in the six ECC banks. As shown in Fig. 6(a), the tier-1 CW is formed by concatenating 512b or 64 symbols (that are read from one of the 64 data banks) with tier-1 ECC symbols (which are read from one of banks in the ECC die). For handling permanent bank failures, T1C and T2C can each provide one spare bank, which are shown as the shaded bank in Fig. 6(a).

They discuss the working of their technique for four scenarios: (1) *When no data bank is faulty:* T1C and T2C use 6 and 2 ECC banks, respectively as shown in case 1 of Fig. 6(b). The six parity symbols of T1C are distributed in multiple banks, which allows power-of-two mapping and ensures that failure of an ECC bank does not lead to six symbol errors which cannot be corrected by T1C. As for the exact distribution, they choose 4 + 2 distribution compared to 2 + 2 + 2 and 1 + 1 + 1 + 1 + 1 + 1, since the 4 + 2 distribution requires only two accesses and even a failure of the bank with 4 symbol can be corrected by T1C. The first four and the last two symbols are referred to as T1Ca and T1Cb, respectively. As for T2C, data in banks 0–3 from all the eight dies are XORed and stored in one ECC bank and data in banks 4–7 are XORed and stored in another ECC bank. To reduce access latency, T1Ca, T1Cb and T2C symbols are stored in an ECC cache. On a read operation, first T1Ca is read and from this, ED is performed. Only in case of error, T1Cb is read. T1Ca and T1Cb can together correct single error and in case of multiple-errors, T2C is activated.

(2) *One data bank is flagged as faulty:* here, every read access from the faulty bank would require activating both T1C and T2C banks. To avoid this penalty, they free-up a bank used by T2C and use it as a spare for the faulty bank, as shown in case 2 of Fig. 6(b). For this, the data from all 64 banks and not just 32 banks, are XORed and stored in one ECC bank. For this, contents of 2 ECC banks of T2C are XORed and stored in one ECC bank, which allows using another bank as the spare bank. This modification impacts EC latency of large-granularity faults only.

(3) *Two banks are flagged as faulty:* For this case, they leverage the rate-adaptive nature of RS codes and use RS(69,64) code for T1C. This code needs only 5 banks and thus, another spare bank is also freed, as shown in case 3 of Fig. 6. RS(69,64) is used to support single-error correction and quadruple-error detection. Four out of five symbols of RS(69,64) are stored in one bank and one symbol is stored in another bank. Thus, the storage pattern of T1Ca of RS(69,64) remains same as that of RS(70,64). (4) *Permanent data TSV failure*: A failure of one data TSV causes only one symbol error, which requires accessing both T1Ca and T1Cb. If the TSV failure is flagged as permanent failure, the corresponding symbol can be treated as an erasure and can be corrected much more easily than random errors. The procedure for handling permanent TSV failure is shown in Fig. 7. Their technique can mitigate errors due to two data TSV faults (of which one is flagged as faulty) without any performance penalty. Their technique reduces raw FIT rate by 10^{10} and incurs only small performance loss. They also show the use of their technique for increasing the refresh period from 64ms to 256ms for saving refresh power without harming reliability.

Jian et al. [66] note that generally, ECC bits can be split into ED and EC bits, and the EC bits account for at least 50% of the total ECC bits. Since the EC bits are required only in the rare case of occurrence of faults, EC bits lead to significant wastage. Further, since memory channels are mutually-independent, at a time, only a single channel shows error and needs to be corrected. Hence, for multi-channel memory systems, they store the bitwise parity of ECC correction bits of different

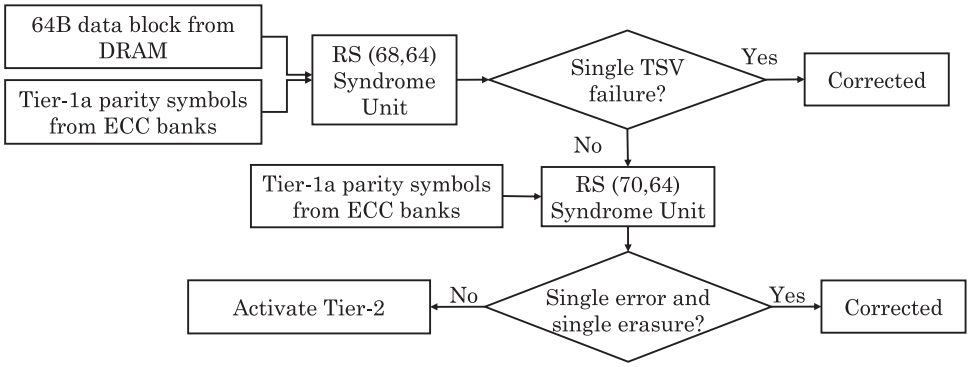


Fig. 7. Decoding procedure of erasure and error correction of RS (70,64) code for handling permanent TSV data failures [45].

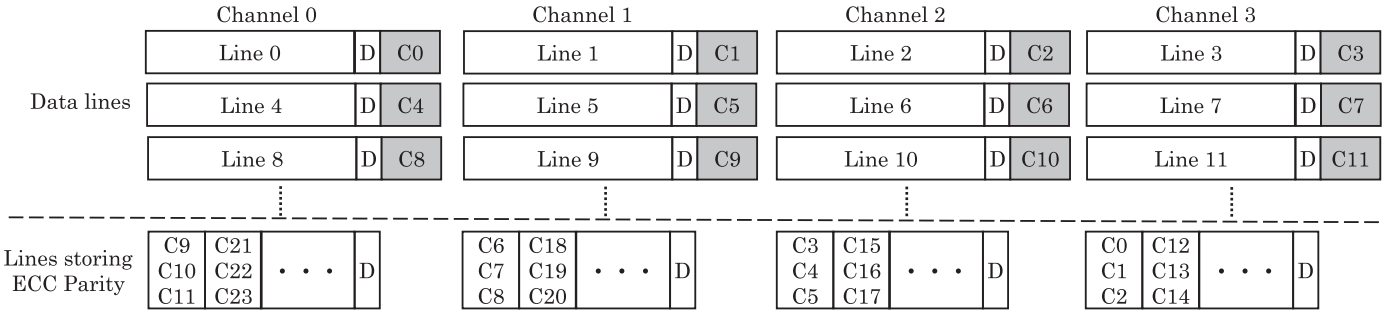


Fig. 8. Example ECC parities. 'D' and 'C' represent eight detection and eight correction check symbols (respectively) for every line. 'C0C1C2' represents C0 ⊕ C1 ⊕ C2. Shaded boxes show the values which are only calculated for the data lines but are not actually stored in memory.

channels for fault-free memory portions. The actual ECC correction bits are stored only for faulty memory portions which reduces the capacity overhead of ECC. This helps in avoiding the situation where faults accumulate across multiple channels over time. This is different from conventional systems which always store the EC bits of each channel in memory. The ED bits are still stored in memory for each channel, and this enables runtime ED on performance-critical read accesses.

When required, the EC bits of a line in a faulty channel are computed by XORing the line's ECC parity with the ECC correction bits of suitable lines in the rest of the robust channels. The ECC correction bits of lines in the robust channels are directly obtained from the lines.

For the 36-device CPK-correct ECC, their ECC parity design stores a four check-symbol code per word in the memory. Of these, 2 check symbols are required for ED and 2 check symbols are required for EC. The correction check symbols are utilized for designing ECC parities. Assuming 4 words per line, every line has 8 symbols for ED and 8 symbols for EC. The data lines of a quad-channel memory system are shown in Fig. 8. The figure also shows the ED check symbols, their ECC parities which substitute the EC check symbols and are shown by the shaded boxes.

On detecting errors in a channel, their technique re-generates the EC bits of the data lines in the channel from ECC parities and data lines in other channels, assuming that the failure location of different memory channels are different. To protect against the case that a channel may later fail at the same location where another channel had earlier failed, the actual EC bits of every faulty memory region are computed and stored in memory. This also avoids the performance penalty of EC due to reading of data lines from different channels for re-generating the EC bits. Their technique can work with various ECCs, e.g., CPK, double-CPK, DIMM-kill, etc. The reduction in ECC storage overhead can be traded for better energy efficiency. Their technique lowers memory energy compared to CPK and DIMM-kill correct ECC.

Future challenges: A limitation of multi-tier codes is that they become ineffective for write-intensive applications and in cases of high error rate. Since error rates are expected to increase due to scaling of feature size, increasing integration density and new sources of error such as VRT, the existing multi-tier codes need to be evaluated under projected high error rates to ensure that existing multi-tier codes remain relevant for future systems also.

4.2. Using large-symbol ECC

Kim et al. [51] note that several fault-types such as subarray fault, TSV faults lead to errors on a single data I/O. Further, aligning ECC symbols to frequent error patterns enables more frequent correction. Their technique provisions large-symbol ECC to the data-burst obtained from single/multiple neighboring I/O pins or TSVs. This vertical ECC layout enables effective correction of DRAM pin/TSV and chip errors. Use of large symbols allows minimizing the redundancy for a given protection-strength and use of large ECC codewords allows detecting nearly all critical errors. Their technique groups per-pin data as ECC symbols and uses an 8-bit symbol RS code for providing effective pin and chip protection.

Based on these, they design multiple ECC organizations: (1) Single-pin correcting (SPC) ECC which uses 2 redundant pins for correcting a bit or a pin error. This is shown in Fig. 9(a). It incurs only 3.1% storage overhead but still provides better uncorrectable error rate than SECDED. However, given the fixed granularity of DRAM chips (e.g., x4 or x8 DDR), SPC uses the pins and storage inefficiently. (2) Single-pin correcting, triple-pin detecting (SPC-TPD) organization uses an extra x4 chip for 4 redundant ECC symbols for a 6.25% overhead. It is shown in Fig. 9(b). It can detect all up-to-3-pin errors and 99.9996% errors beyond this point. (3) Quadruple pin correcting (QPC) organization, shown in Fig. 9(c), uses two redundant x4 chips for a 64b data channel, thus, it has an overhead of 12.5%. It can correct at most 4 symbol errors or a single chip-error. Compared to the commercial CPK, QPC can additionally correct pin errors scattered over different chips. Two pin faults on two chips can both be corrected by QPC, whereas the commercial CPK may lead to DUE or SDC. By virtue of using large codeword size, QPC

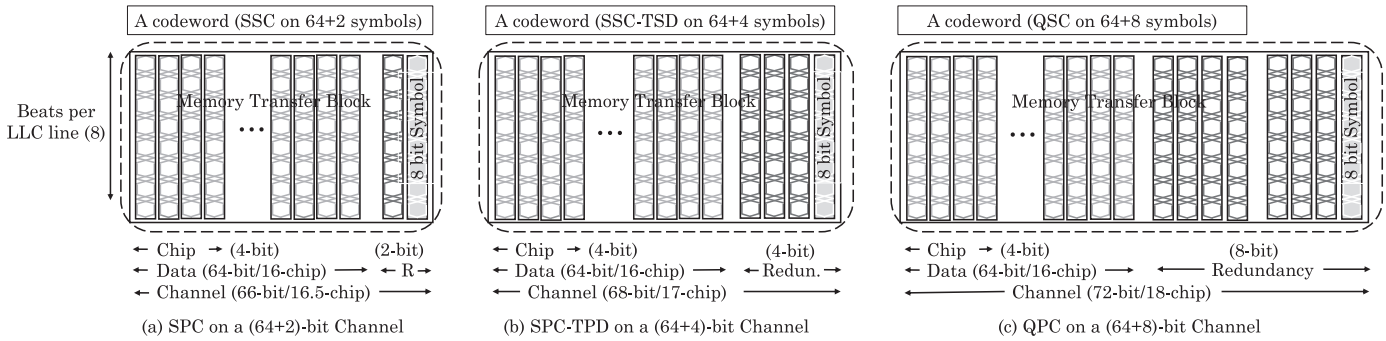


Fig. 9. ECC layouts in the technique of Kim et al. [51] for a 64b data channel (with an 8b burst length).

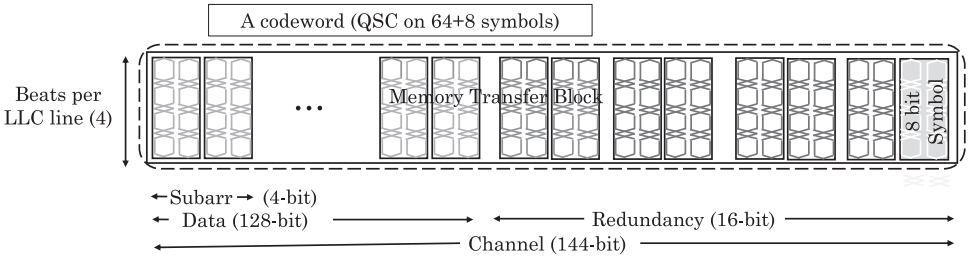


Fig. 10. QDPC [51] on a 128b data channel (2-pin x4-beat symbols).

also has stronger detection capability and thus it can detect practically all the errors.

(4) Octuple pin correcting (OPC) ECC uses 16 redundant pins to correct 8 pins, which incurs 25% and 12.5% overhead on 64b and 128b data channels, respectively. (5) Double double-pin correcting (DDPC) corrects an x4 error by correcting two 2-pin symbols and quadruple double-pin correcting (QDPC) corrects two x4 errors or one x8 error by correcting four 2-pin symbols. DDPC/QDPC utilize 8/16 redundant pins and incur 6.25%/12.5% (respectively) overhead in a 128b channel. Recent on-package memories, such as Wide-IO memory [5], use 4b burst lengths to realize 64B access granularity over a 128b data interface. For them, their technique can build 8b symbols over 2-pin x4-beat blocks, as shown in Fig. 10, which still have the minimum redundancy with an 8b symbol. They also discuss a strategy for limiting the correction capability of QPC, OPC, DDPC and QDPC for reducing their SDC rates.

They further present a gracefully downgrading scheme whereby with ongoing pin/chip retirement, an ECC of decreasing strength is used from the above mentioned ECCs. This scheme can correct more initial chip faults, more sequential chip faults, more end-of-life faults and provides better end-of-life detection capability than Intel's double-CPK correct scheme [29], which tolerates only two sequential chip failures and a bit failure. Overall, their ECC technique provides high error coverage and improves memory lifetime.

Future challenges: A limitation of the technique of Kim et al. [51] is its higher latency due to the requirement of the alignment of the pin-based symbols. Also, their technique has high decoding complexity. In near future, techniques with even lower overhead need to be developed to justify their adoption in product systems.

4.3. Using subarray-level ECC

Udipi et al. [62] present a "single subarray access" (SSA) design where a whole cacheline is read from a single subarray of cacheline width in a single DRAM chip. This design is shown in Fig. 11. It saves dynamic energy by activating only those bitlines which are required for reading a single cacheline. To save background energy, other idle subarrays can be transitioned in low-power sleep modes. In this design, the whole cacheline is provided through the limited pins on a chip which

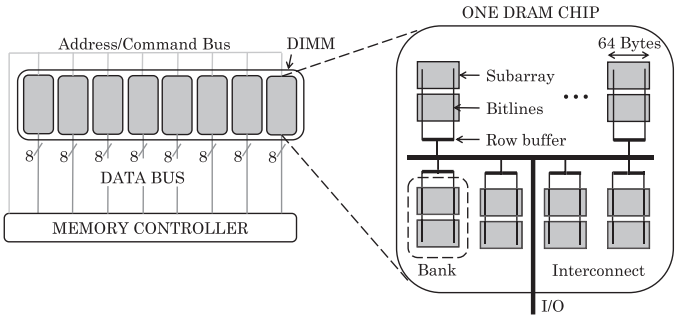


Fig. 11. Architecture of SSA DRAM [62].

leads to higher access latency. This design supports only closed-page policy, which may have positive or negative performance impact on different workloads. Since different subarrays in a chip can each provide a different cacheline, this design provides very high concurrency.

To ensure reliability, a mechanism is required for reconstructing data from a faulty chip, since a whole cacheline is stored inside a single chip. Hence, they use a scheme similar to RAID-5, which is shown in Fig. 12. The DIMM has 9 chips of which one chip stores the parity for the remaining 8 chips, thus, incurring an overhead of 12.5%. The checksum is stored along with the data. For a 64B cacheline, an 8b checksum is used for 1.625% overhead. To also detect stuck-at-0 faults, the checksum function uses bit inversion.

On a read operation, a single chip is accessed. The correctness of read is verified through the checksum. In case of no error, no further access is required and thus, in the common case of error-free read operation, their technique accesses only a single chip compared to many chips in the CPK schemes. On an error, the data is reconstructed by reading the other seven chips and the parity chip. In case where data + checksum are faulty but checksum happens to be correct, an SDC error occurs. However, by increasing the checksum length, the probability of such SDCs can be reduced. The limitation of this scheme is that the writes incur high overhead since parity bits computed over multiple cachelines need to be updated whenever even one of the lines is written to. This limita-

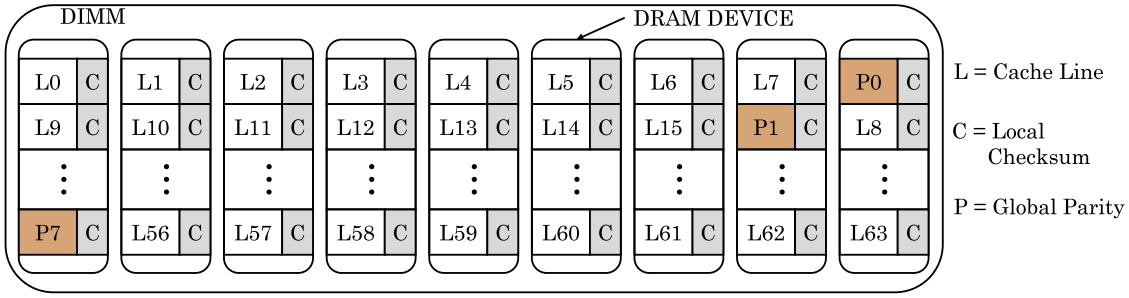


Fig. 12. Chipkill support in SSA [62] (illustrated for 64 cache lines).

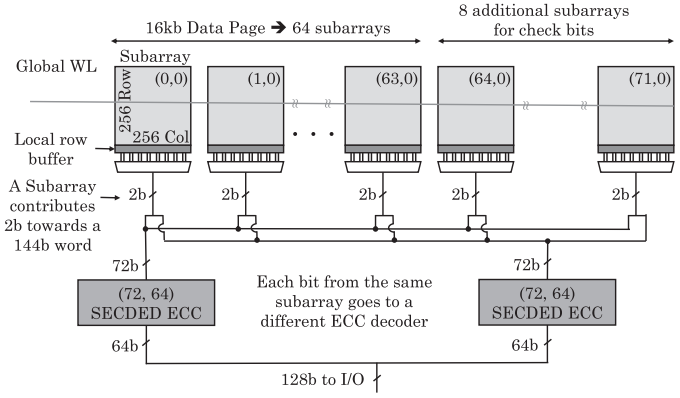


Fig. 13. SECDED-based Subarraykill design for 16kb page [52].

tion is mostly mitigated by the high concurrency provided by the SSA design. Compared to CPK schemes, their technique reduces the storage and energy overheads of reliability. Note that some techniques store the cacheline in a single chip [62] or a single bank [50] to improve performance and energy efficiency.

Giridhar et al. [52] present a "Subarraykill" technique for protecting subarrays in a bank. Their technique is analogous to the chipkill-technique. Their technique protects against soft errors (particle strike leading to burst errors in a subarray) and hard errors (e.g., multi-bit faults along columns/rows up to entire subarrays, and TSV failures). The subarraykill technique spreads a data word uniformly across a whole page for tolerating multiple errors in a subarray. These errors can be handled using either SECDED or SSCDSD-based codes. Assuming a 16Kb page, on using SECDED scheme, the 144b CW is distributed across 64 + 8 subarrays, as shown in Fig. 13. Each subarray provides 2b for the data-word. On a read, these 2 bits are sent to different ECC units, so that in case of a soft error, or a subarray failure, every ECC word will have at most 1b error, which can be corrected using the (72,64) SECDED scheme.

However, with decreasing page-size, SECDED code becomes smaller. This increases its relative storage overhead and the energy consumed in fetching the check bits for normal access and refresh operations. Hence, they propose using rotational SSCDSD-based code. Rotational codes incur lower latency, area and power overhead while providing the same EC-strength as SECDED. For example, (144,128) SSCDSD with 4b symbol can correct up to 4 neighboring bit errors and its area overhead for a 4Kb page is only 12.5%, compared to 21.9% for SECDED. For a 4Kb page, SSCDSD is organized as follows and is shown in Fig. 14(a). The page is opened in 16 + K subarrays where the K extra subarrays store the check bits. The 144b word is distributed across this page with 32b spacing, and thus, a particle strike cannot create a burst error of more than 2 bits. A (144,128) SSCDSD code (with 4b symbol) guarantees that at most one symbol will have error, which can be corrected.

This scheme can detect but not correct hard errors, e.g., failure of an entire subarray. To correct them, stronger codes, such as (152,128) SSCDSD code (with 8b symbol) can be used which is shown in Fig. 14(b). The above-mentioned SSCDSD codes have more than 99% likelihood of detecting 3 or 4 bytes of errors, which is useful when multiple subarrays have soft and hard errors. In case of faults spanning across multiple subarrays, entire-chip, etc., the failures can still be detected by ORing the error-detect signal from different subarrays. Their experiments confirm the efficacy of their technique.

Future challenges: A challenge in use of a large number of subarrays is that it complicates the design and operation of on-die network and periphery (such as sense amplifier, decoder, etc.). This harms area and cost efficiency. Also, subarray-level techniques may require significant changes to DRAM architecture, for example, the technique of Udipi et al. [62] requires latching capability in each device. Moving forward, efficient solutions to these challenges are required for leveraging the full potential of subarray-level techniques.

4.4. Using erasure correction

Jian et al. [46] note that the traditional linear block code-based CPK schemes use a dedicated check symbol for each codeword for EC which allows individually correcting the bad symbol in each codeword (refer Section 2.4). This is useful when errors occur in different symbols in different codewords. However, in practice, errors generally occur in the same symbol in different codewords of a CPK-correct memory. Since every device contributes the same symbol to each codeword, every time, a faulty device causes the same symbol in the codeword to become faulty. They propose allowing multiple lines with the same column address in neighboring rows to share the same set of checksums for reducing checksum overhead. This brings slight reduction in the error coverage such that errors in a group of codewords can be corrected only if every error is located in the same symbol position. In other words, errors due to two faults in two different devices in a rank is correctable only if the two faults do not impact a single checksum group, since in that case, two error locations would be reported, which cannot be handled since every codeword has only one RS check symbol.

CPK schemes that use a dedicated correction check-symbol need at least three check symbols in each CW. To reduce the ED overhead, they use erasure correction for correcting the CW since it does not require using correction-check symbols. Their approach requires only one redundant device for each rank since every CW requires a single check symbol for both ED and erasure-correction. To localize the error, their technique uses checksums stored in every device in the rank, including the data devices. Since allocating checksums for each CW incurs large overhead, their technique shares a group of checksums across multiple lines in neighboring rows that have the same column address. Every CW belongs to a column checksum group. A CCG is composed of CWs in the same column and the checksums computed from these CWs. Fig. 15(a) shows a CCG with only two CWs per checksum for simplicity, although their technique actually uses 256 CWs in each CCG to limit the overhead. The checksum and the corresponding data are stored in the same

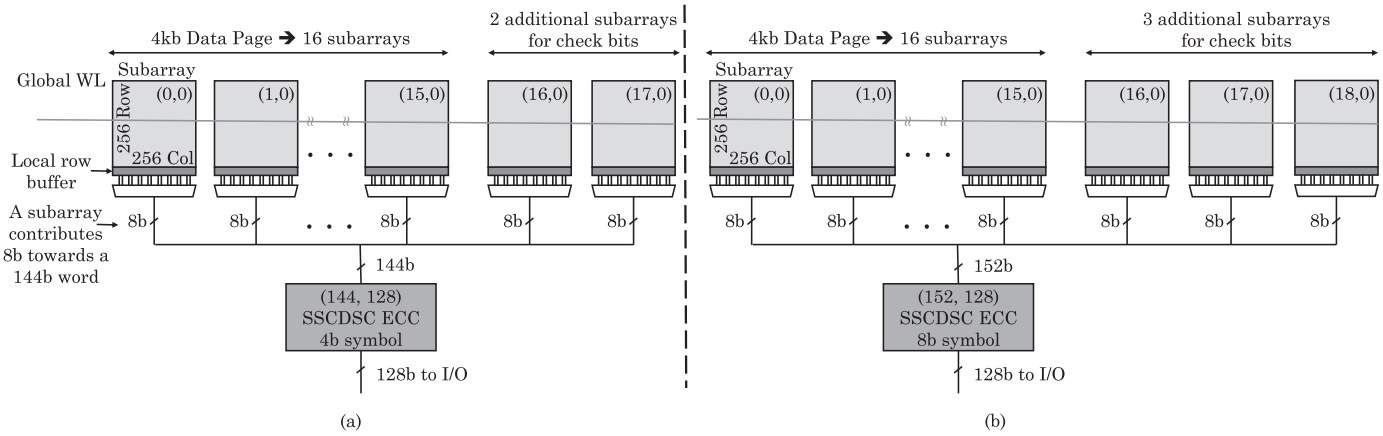


Fig. 14. (a) (144,128) SSCDSD-based Subarraykill design for 4kb page. (b) (152,128) SSCDSD-based Subarraykill design for 4kb page.

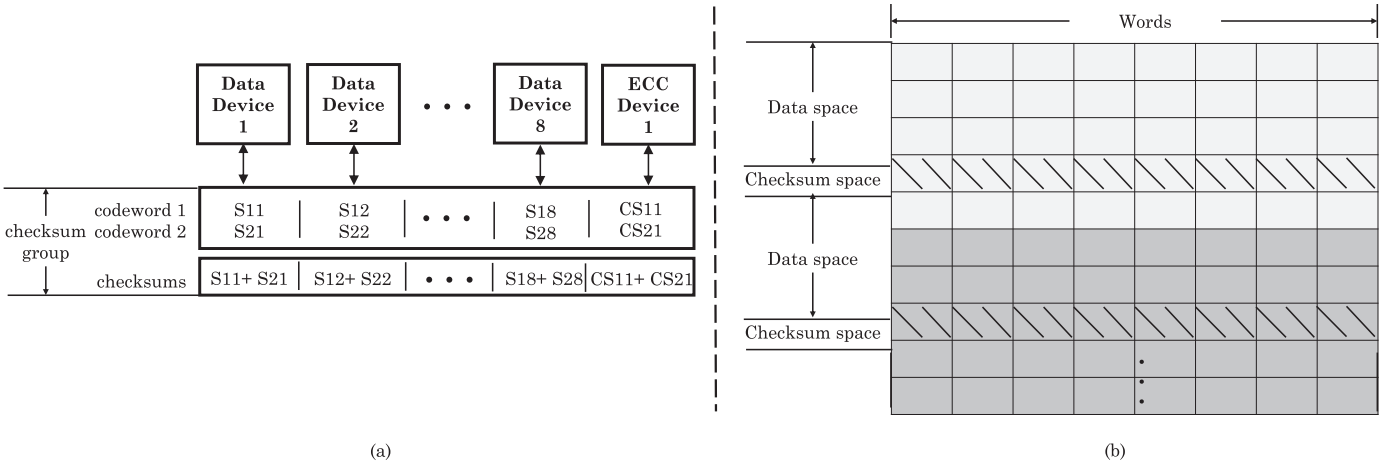


Fig. 15. (a) Example of a checksum [46]. Here, + shows the computation of checksums from data symbols, 'S' refers to symbol, and CS refers to an RS check symbol. (b) The data and checksum regions in a bank. Each grid shows a CW. Here, five neighboring CWs in the same column belong to the same CCG, with the fifth being the checksum of the CCG.

device. For a faulty device, the stored checksum does not match the checksum newly computed from the data symbols and based on this, a faulty device can be easily identified. In the memory, the checksums are stored in the rows adjacent to the checksum groups they protect, as shown in Fig. 15(b).

For ED, a linear block code (viz., 16b RS code) is used with a single check-symbol. This check-symbol ensures detection of all single-symbol errors and offers high detection coverage of multi-symbol errors. In RS code, use of 16b per symbol increases detection coverage compared to using 8b, and keeps decoder overhead low compared to using 32b. In RS codes, the CS used for ED can be re-utilized for providing erasure correction, i.e., the faulty symbol detected by a CS can also be corrected by the same CS if the location of faulty symbol is known. For finding the location of fault, they use intra-device checksums which are stored in every device. Specifically, they use one's complement checksums which can detect stuck-at-0 and stuck-at-1 faults. Their technique can also be extended to provide double-CPK correct protection by using two check symbols in each codeword. Compared to CPK scheme, their technique reduces memory power consumption with negligible impact on storage overhead and reliability.

Jian et al. [64] note that device-level faults (e.g., those impacting a whole column, sub-bank or device) lead to multiple errors. Hence, these faults can be localized with small overhead and then, errors due to them can be corrected through erasure codes. Further, a local fault impacts only a single word, hence, sharing a single unit of EC resources for hundreds of words can correct the resultant errors with high coverage. Their technique decouples correction of errors from local faults and device-level faults. This halves the number of devices consulted on each access for the same rank size without increasing the overhead compared to traditional CPK-correct memories.

In their technique, the codeword is generated using an erasure code. Every symbol of a CW is stored in a separate device in the rank for ensuring ED in case of device failure(s). For keeping the CW overhead same as that in SECDED, in their double CPK-correct design, a codeword has 18 symbols, of which 2 are check-symbols. For keeping the same code overhead in the single CPK-correct design, the codeword has 9 bits, of which one bit is an odd parity bit for the remaining 8 data bits. Each CW is part of a "row checksum group" (RCG) and a "column checksum group" (CCG), as shown in Fig. 16. A RCG/CCG consists of a collection of CWs in the same row/column and a row checksum (RCS)/column checksum (CCS). RCS and CCS are together known as "multidimensional parity checksums" and for bounding their overhead, 256 CWs are assigned to each checksum group. Let the number of RCGs per row and CCGs per column be *NRG* and *NCG*, respectively. A CW in the *i*th row and *k*th column is mapped to *p*th RCG in the *i*th row and also to the *q*th CCG in the *k*th column, where $p = mod(k, NRG)$ and $q = mod(i, NCG)$. This mapping distributes every faulty cell across different checksum groups and thus, maximizes the physical distance between the members of the same checksum group for mitigating the impact of multi-bit cluster faults.

Their technique performs ED using the linear block code. On detecting a faulty CW, the rest of the CWs in the faulty CW's RCG are examined. If no errors are found in these CWs or the RCS of the RCG, then

Table 3
ECC codes proposed by Chen et al. [63] for x8 DRAM systems. Both codes have a storage overhead of 12.5%.

	# of chips per access	Granularity	ECC capability
Code 1	18	64B (4 beats)	Corrects 2 errors or 2 erasures; Corrects 2 erasures and 1 error
Code 2	9	64B (8 beats)	Corrects 1 error and detects 3 errors; Corrects 4 erasures

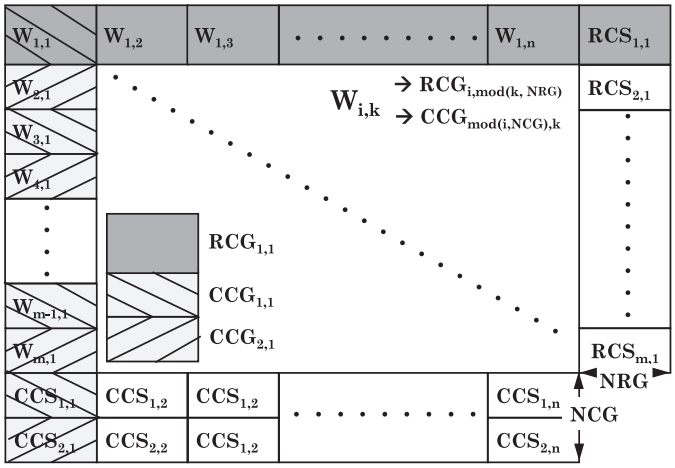


Fig. 16. An example design of CPK-correct memory with multidimensional parity checksums [64]. Here, each row has 1 checksum group ($NRG = 1$) and each column has 2 checksum groups ($NCG = 2$).

the correct CW is obtained from the result of XOR of the CWs and the RCS using bitwise odd parity. If a different CW or the RCS is found to be faulty, the CW cannot be corrected by RCG and the process is repeated using the CCG. A faulty checksum is corrected using its checksum-group after any faulty CWs in this group have been corrected.

A device-level fault cannot be corrected by the RCG or CCG. To correct them using the erasure CWs, first, the faults need to be localized. Their technique first localizes individual column faults. After localizing multiple column faults to the same sub-bank, a sub-bank fault is detected. If multiple sub-bank faults happen in the same device, a device-fault is flagged. Their technique does not localize row faults. After localizing a fault, errors arising due to the fault are corrected using the standard syndrome correction for erasure codes. Errors arising from alignment of a device-level fault and local fault impacting the same CW are corrected by the combined use of erasure correction and checksum group correction for them, respectively. Their experiments confirm the effectiveness of their technique.

Chen et al. [63] present two erasure and error correcting codes (E-ECCs) for x8 DRAM system which can handle both transient and permanent errors. Both codes are based on RS (36,32) code over GF(2^8), incur 12.5% storage overhead and do not require additional accesses for error correction. These codes offer different tradeoffs between performance, energy and reliability, as shown in Table 3.

Code 1: Two ranks, each with 9 chips, are activated in every access. With burst chop mode operation with 4 beats, access granularity is 64B. In every beat, 144b (128b +16b) are read from 18 chips. RS (36,32) code over GF(2^8) is used which provides stronger error correction capacity than rotational (144,128) code over GF(2^4). This code can correct single error, double error, double erasure, and "double erasure and single error". As for the data access mechanism, two consecutive 8b symbols are read from a single bank in two successive beats. From 18 chips, 36 symbols are read and provided to the decoder. The decoding algorithm is shown in Fig. 17(a). The special algebraic structure of RS code allows efficiently distinguishing between zero error, single error (case 1A in Fig. 17(a)) and double error (case 1B) based on the syndrome vector. On a chip failure, two error symbols are generated which can be corrected. If the location of the faulty chip is known, the two error symbols can be taken as erasures and corrected. If a chip is flagged faulty due to persistent errors, an extra error in another chip can also be corrected. For handling double erasure correction (case 1C) and double erasure and one error correction (case 1D), their technique first performs double erasure correction. If the event is not ascertained to be a double erasure event, the double erasure and single error correction unit is activated. This code provides stronger reliability than CPK-correct techniques.

Table 4
Prototype used and protection strength of techniques for stacked-DRAM.

Category	References
Prototype used for evaluation or comparison	
HBM	[8,43,45,50,59]
Octopus	[52]
HMC	[52,65]
Protecting against up to complete ...	
Bank failure	[45,50]
Channel failure	[43,69]
Die failure	[8]

Code 2: This code activates only one rank (i.e., 9 chips) and thus, reduces latency and power consumption, although it provides less reliability than code 1 and other CPK solutions. Here, four successive 8b symbols are read out from a bank in four successive beats. Thus, from 9 chips, a total of 36 symbols are read and provided to the decoder. The decoding scheme is shown in Fig. 17(b). Although RS(36,32) can offer either (a) single error correction and triple error detection or (b) single and double error correction, they select the former, since it can detect up to 3 errors with 100% probability and 4 errors with 99.9986% probability. This is shown as case 2A in Fig. 17(b). Code 2 corrects errors due to single-bit, single-word and single-column failure with 100% probability and row failure due to random errors with 99.9986% probability. All symbols of a faulty chip are taken as erasures and the 4 consecutive erasure symbols are corrected (case 2B). As such, row failures due to permanent errors are handled as erasures and corrected with 100% probability. The decoding latency of their codes is much lower than the DRAM access latency. Their proposed codes provide higher performance and energy efficiency than CPK-correct techniques. Chen et al. [76] also propose E-ECCs for x4 and x16 DRAM systems.

Future challenges: GPUs, which were initially used mainly for error-tolerant applications, are now being deployed for a wide-range of general-purpose and mission-critical applications [5]. As such, improving reliability of GPU memories has become important. The existing DRAM reliability techniques have been evaluated in the context of CPUs. Since architectural characteristics and optimization objective of GPUs differ significantly from those of CPUs, evaluating existing DRAM reliability techniques in context of GPUs and proposing novel techniques for GPUs will be a major research challenge for researchers.

4.5. Techniques for stacked DRAM

We now discuss the techniques proposed for protecting stacked DRAM. Table 4 highlights the protection strength of these works and also shows whether they perform experimental evaluation with (or comparison against) HBM, Octopus or HMC, although their technique may work with multiple stacked-DRAM prototypes.

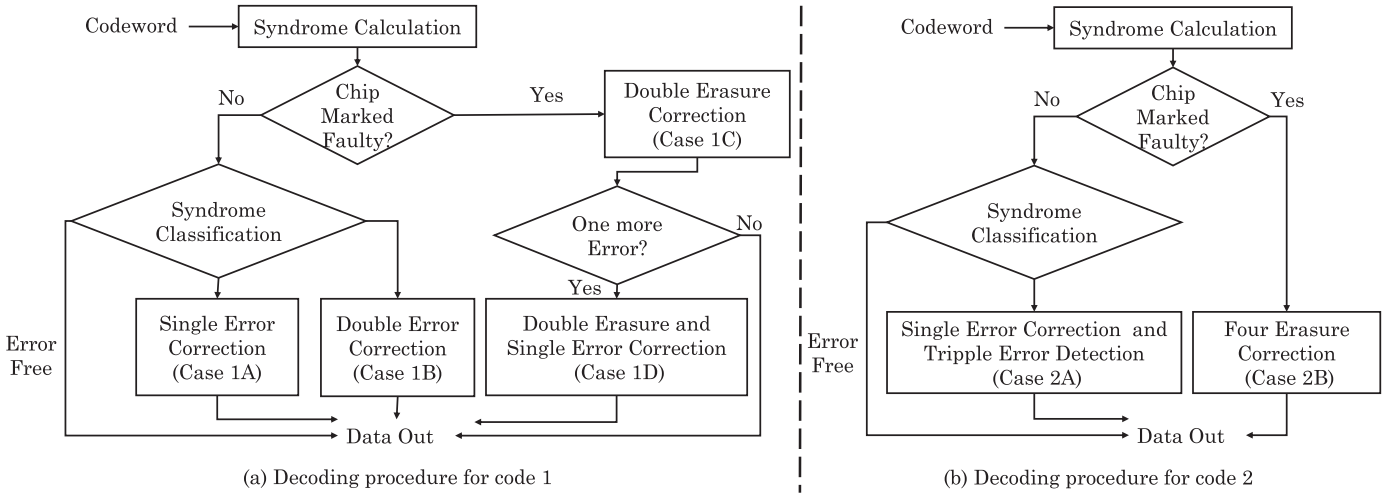


Fig. 17. Decoding algorithm for (a) code 1 and (b) code 2 in the technique of Chen et al. [63].

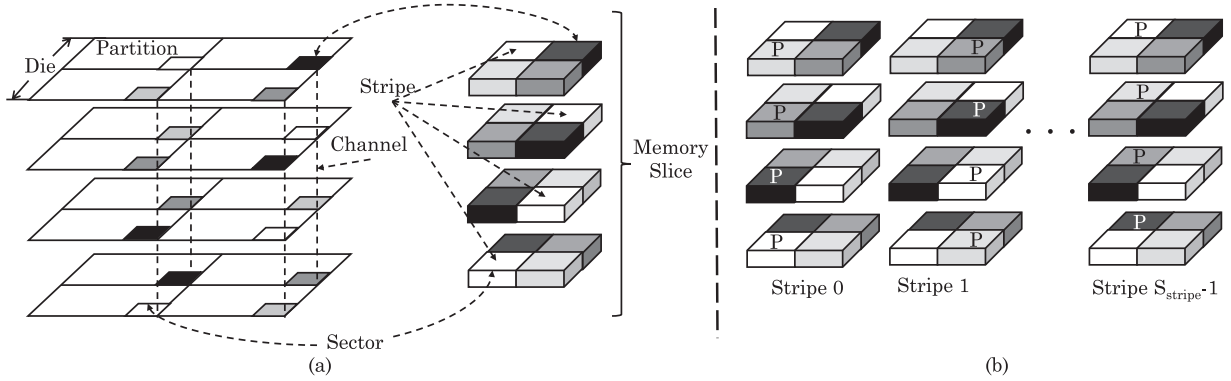


Fig. 18. (a) Design of stripe/sector etc. in the technique of Jian et al. [8]. Every stripe is shown with a different color in the figure. (b) Layout of parity. The parity sectors rotate over S_{stripe} memory slices. This distributes the parity sectors uniformly over all partitions which avoids bandwidth issues while updating the parity sectors.

In stacked DRAM, multiple DRAM banks laid out in a horizontal plane constitute a die and multiple vertical DRAM banks constitute a channel. Thus, a die/channel fault impacts horizontal/vertical set of banks, respectively. Using different ECC resources for protecting against faults in each dimension incurs large overheads. A one-dimensional fault is one which impacts memory banks in only a subset of dimensions in a multi-dimensional memory. Examples of one-dimensional faults are a die fault and a channel fault. Jian et al. [8] present a technique for protecting multi-dimensional (e.g., die-stacked) DRAM against single-dimensional faults. Similar to a helix which intersects a horizontal plane and a vertical line segment in at most a single point, their technique designs each parity group (i.e., data line and its parity line) in helix-manner.

Both their technique and RAID5 use separate ECC bits for each line for detecting errors and correcting small errors, and parity lines for correcting large errors affecting a whole line. However, the difference in them is in the construction of parity lines. As shown in Fig. 18(a), they divide memory into partitions which are further divided into sectors. A sector is a set of row-adjacent line locations in memory. A data sector has data and a parity sector has bitwise XOR of the contents of a group of data sectors. A slice is formed from the fixed sector in all partitions. A stripe is formed from a helix-like collection of sectors in a slice. Their technique maps all sectors of a die in a memory slice to a different stripe and all sectors of a channel to a different stripe. Hence, as shown in Fig. 18(b), a channel fault or a die fault impacts up to one sector in every stripe and hence, can be corrected by the parity stored in the parity sector of every stripe.

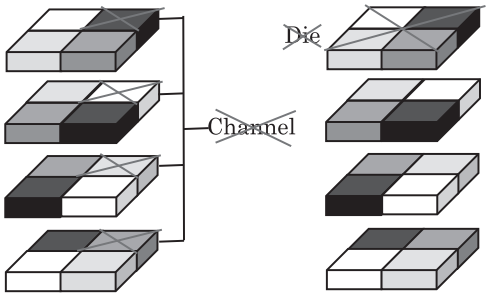


Fig. 19. A channel fault (left) or a die fault (right) affects up to a single sector per stripe and hence, it can be corrected by a single parity sector in each stripe.

In a DRAM stack, parity sectors are uniformly distributed across all partitions of a stack, which reduces bandwidth consumption due to updating of the parity sectors. For this, parity sector is rotated over all possible positions in a stripe. Since sectors of a stripe come from different partitions, alternating which sector in every stripe is the parity sector, uniformly spreads the parity sectors across all partitions. This is shown in Fig. 19.

When a line shows an error which cannot be corrected by the dedicated ECC of the line, their technique uses parity line and other data

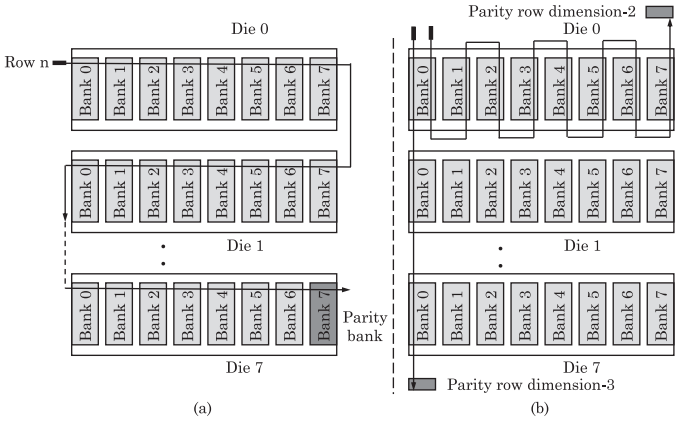


Fig. 20. Computation of parity [50] in (a) dimension 1 and (b) dimension 2 and 3.

lines in the same stripe for obtaining the faulty line. Although reading multiple lines incurs high overhead, it is required only in case of large-granularity faults, e.g., die faults and channel faults. To reduce the overhead of these cases, they propose gradually retiring faulty dies/channels, which avoids future access to them and obviates the need of EC. Compared to a technique which maintains EC resource for each dimension, their technique increases memory capacity by sharing EC resources in all dimensions. Also, on each write, their technique requires update of only a single parity and not multiple parities. The limitation of their technique is that it has high DUE rate.

Nair et al. [50] present a technique for protecting stacked DRAM from TSV- and bank-failures. For 8 data dies, their technique uses one extra die (ECC die) with smaller rows. For each 512b cacheline, 64b metadata is used. Their technique uses three schemes: TSV-SWAP, "tri-dimensional parity" (3DP) and bi-modal sparing. They note that while a data TSV (DTSV) affects only one bit in a data line, a fault in an address TSV (ATSV) can prevent access to half of the memory. Hence, ATSVs are more critical than DTSVs. TSV-SWAP computes CRC-32 based on data and address information. A TSV error leads to a wrong checksum. To distinguish between data-faults and TSV-faults, two extra rows are used in each die. These rows store fixed data sequence and are at locations such that their bit-addresses are mutually inverse, e.g., address 0x0000 and 0xFFFF. On a CRC mismatch, data from these fixed rows are compared with the pre-decided sequence. An error indicates that error is most-probably due to a TSV fault. To mitigate TSV faults, TSV-SWAP creates standby TSVs from the available DTSVs and uses them to repair the faulty ATSVs and DTSVs. To create stand-by TSVs, data of predefined TSV locations are duplicated into 8b swap-data in the metadata.

Apart from TSV-faults, stacked DRAM also sees DRAM die faults. To detect data errors, 32b CRC (CRC-32) is used with each line. For correcting data errors at multiple granularities, 3DP is used which has the property that even two faults in a dimension are unlikely to fall into the same block in the other two dimensions. As for 3DP, in dimension 1, parity is taken for a row in every bank across dies. In dimension 2, parity is computed over all rows in all banks within a die and in dimension 3, parity is computed across dies for all rows in a single bank. This is shown in Fig. 20.

To mitigate the parity-update overhead for dimension 2 and 3, the parity information is kept on-chip. For dimension 1, parity information is cached in the on-chip LLC, due to its large size and the fact that accesses to parity lines show high temporal locality. Small granularity faults such as bit/word/row faults are corrected by dimensions 2 and 3 parity, whereas large granularity faults such as column/bank faults are corrected using dimension 1 parity. Use of 3DP scheme allows storing the cacheline in a single bank while still correcting up to bank failures.

However, recomputing data from parity incurs large penalty which is unacceptable for permanent faults. To avoid this issue, a corrected data item is redirected to a spare location. They note that sparing at small (e.g., row) granularity is inefficient for tolerating large (e.g. bank) granularity failures and vice versa. They note that a bank generally has either few (<4) row failures or thousands of row failures (due to a sub-array or bank failure). Based on this, their bimodal sparing scheme spares either at a row or bank granularity. This scheme can spare two erroneous banks along with many row failures. Overall, their technique provides much higher reliability than CPK scheme while incurring 14% storage overhead. While their technique can efficiently correct errors due to permanent faults (e.g., bank, row, TSV), it incurs high overhead in handling errors due to transient faults.

Jeon et al. [43] present a technique which decouples ED and EC to avoid incurring the overhead of EC on every access. In their design, every bank has rows for normal data, ED code (EDC) and XOR correction code (XCC). EDC detects errors in both data and XCC regions. As for EDC, they note that using 8b interleaved parity for a 64B cacheline fails to protect a single TSV failure, since in a stacked DRAM, a single TSV failure creates multiple errors in a cacheline, each of which is located at exactly the same location in every 128b read. To resolve this issue, they use a two-level 8b interleaved parity, which divides the computation in two levels. In the first level, four intermediate 8b interleaved parities are computed for every 128b data unit, which is the least transfer granularity of wide I/O stacked DRAM. In the second level, the intermediate parities of the first, second, third, and fourth 128b data are rotated by 0, 1, 2, and 3 bits, respectively. The four rotated intermediate XOR values are XORed together to obtain the final 8-bit interleaved parity. Rotation helps in ensuring that bits affected by TSV failure are XORed into different bit positions in the final parity. Hence, the two-level parity can detect any single TSV failure. The data and its corresponding EDC are stored in different banks of different channels to avoid bank conflicts and allow reading them concurrently.

For XCC, they use RAID5-like parity, since in stacked DRAM, failure of multiple TSVs leads to failure of an entire channel. For a DRAM with N channels, $1/(N-1)$ rows in each bank are reserved for XCC. XCC address computation is slow since it uses non-power-of-two calculations, however, since it is required only during write operations and error recovery, the performance impact remains small.

In their technique, a write operation requires removing old information from XCC. To accelerate writes, their technique starts removing old information when the corresponding cacheline in LLC first becomes dirty in LLC, by utilizing the original clean copy of the data. Since a cacheline is generally written several times before being evicted, there is sufficient time between the first write to a cacheline and its eviction from LLC. Since placing the data and EDC in different channels increases power consumption and reduces the bandwidth to half, they cache EDC information in a direct-mapped cache, which is implemented on the logic layer. On the first access to any cacheline in a 2KB data row, the entire 32B EDC is fetched into EDC cache. Future accesses to other cachelines in this row search the cache before fetching EDC from memory. Their technique improves reliability with negligible performance impact. A limitation of their technique is that it does not provide any scheme for handling permanent bank failures.

Mappouras et al. [68] present a technique for protecting 3D DRAM. They assume that the ECC is co-located with the data in every row. They assume Hamming (72,64) ECC code which offers SECDED protection with 12.5% overhead. They note that in multi-tier ECC, the T2C corrects large-scale faults. T2C protects multiple (say K) blocks by using a redundant block which is the XOR (i.e., parity) of the K data blocks. The K data blocks and one parity block are spread to ensure that any single fault can be corrected. However, they note that T2C has four key limitations. Assume a data block A in a parity group with parity block P and their old values are A' and P'. (1) On each write to A, first, A' and P' need to be read. Then, XOR between A and P, and A' and P' need to be performed, and finally A and P can be written. (2) Write to A and write

to P need to be atomic with respect to other writes to remaining blocks in A's parity group. (3) To correct an error in A with T2C, all blocks in A's parity group need to be accessed. Further, a block in A's parity group can be written only when the correction is completed. This impacts the working of MC and the coherence protocol. (4) An integrated MC is cognizant of the entire memory, however, in HBM, every channel has its own controller which is aware of only the memory of that channel. Without an integrated controller, the LLC bank which sends a memory request need to compute parity bits, issue additional requests for loading/storing parity bits and detect/correct errors.

To resolve these issues, their technique adds redundancy inside each block and not across multiple blocks. Each memory block A(72B) is split into two sub-blocks A1 and A2, each 36B. By XORing them, a redundant subblock A3 (36B) is obtained which is the parity of A1 and A2. To tolerate channel and die failures, the three sub-blocks are striped in different channels and different dies. One third of total capacity is reserved for storing the parity subblocks (e.g., A3) and remaining capacity is used for storing data. On a read, half-block accesses are performed over two different channels on two different chips. By concatenating A1 and A2, A is obtained. By increasing the number of accesses, their technique increases bank conflicts, although the total data fetched remains same (ignoring packet overheads). A write requires computation of A3 from A1 and A2, and then writing them in different channels. Thus, their technique consumes higher write bandwidth.

If SECDED cannot correct an error in a subblock (say A2), then A1 and A3 are read. By XORing them, A2 and hence, A is obtained. Since the write and error recovery operations do not involve multiple blocks, their complexity is reduced compared to previous techniques. Their technique can tolerate multiple row and bank failures if multiple rows or banks storing subblocks of the same cache block do not fail. One channel or chip failure can be tolerated, and two channel failures can also be tolerated if both channels are on the same chip. Similarly, even multiple TSV failures can be tolerated if the failed TSVs are separated by a minimum of 64 TSVs. Their technique reduces EC latency and does not require modifications to HBM protocol.

Future challenges: Since the operational temperature of stacked DRAM is higher than that of 2D DRAM, it needs to use lower refresh period. The refresh period will reduce even further with decreasing feature size. To overcome these challenges, the integration density or the number of layers may need to be reduced. Evidently, to continue to increase the capacity and performance of stacked DRAM, highly efficient reliability techniques are required that can also reduce the requirement of refresh operations.

5. Techniques for reducing ECC overheads

Given the highly performance/power-sensitive DRAM design market, a reliability technique with high overheads is unlikely to get integrated in product systems. Hence, strategies for reducing their overheads are vital for ensuring their adoption. Table 5 summarizes several strategies for reducing performance/energy overheads of reliability techniques. Some works propose caching ECC bits on-chip to avoid the latency incurred in accessing ECC bits from DRAM. While storing the data and its corresponding ECC in different devices improves reliability, it requires both the devices to be accessed which increases the energy consumption. By comparison, some works store the data and ECC in the same row to increase row-buffer hits, although it harms reliability since an error in the row may corrupt both the data and the ECC.

Use of strong ECC allows reducing the refresh rate since ECC can correct the errors caused due to lack of timely refresh operations. Through this approach and by writing-back only dirty words, energy can be saved. To reduce storage overhead, compression can be used to create space for storing ECC and selective memory protection can be used. Similarly, memory-efficient structures such as Bloom filter can be used and sparing can be performed at optimal granularity.

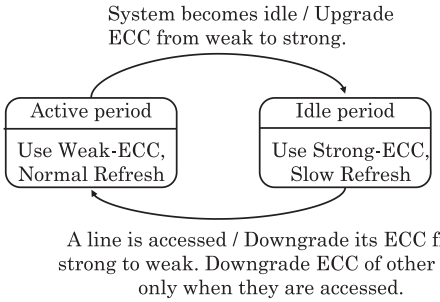


Fig. 21. The working of adaptive ECC technique of Chou et al. [47].

In this section, we discuss techniques which adapt the strength of ECC (Section 5.1), protect selected memory regions (Section 5.2) and adapt the access granularity (Section 5.3). Further, we review techniques that perform data-compression to make room for ECC (Section 5.4) and use ECC bits to reduce read-stall caused due to refresh operations 5.5.

5.1. Adapting the strength of ECC

Table 6 highlights the features of works that adapt strength of ECC and protect only selected memory portions. These works are discussed in this Section and in Section 5.2.

Chou et al. [47] note that mobile devices have short periods of intense user-interaction and long periods of idleness (e.g., more than 90%). During idle times, refresh power consumes a large fraction of total power, although refresh power contributes only a small fraction of total power in active mode. Further, using a strong ECC (e.g., ECC-6 which can correct up to 6b in a line) allows increasing the refresh period from 64ms to 1s, however, decoding the multi-bit ECC incurs high latency which is unacceptable in active periods. They propose a technique which provides high performance in active periods using weak-ECC (e.g., SECDED) and refresh-power saving in idle periods using strong-ECC (ECC-6). Fig. 21 summarizes the working of their technique. During idle periods, the whole memory uses ECC-6 and refresh period is increased to 1s. When a memory access to a line arrives, the line would have ECC-6 on the first access and hence, it is decoded with ECC-6 decoder and saved with weak-ECC. Thus, the performance penalty of strong-ECC is incurred only on the first access and not subsequently. When the system becomes idle, only those lines which were using weak-ECC are converted to ECC-6.

They further note that short-duration periodic events such as interrupts from network devices, bluetooth signal check, etc. may lead to frequent ECC-mode transitions and thus, nullify the benefits of their technique. To resolve this issue, their technique tracks memory access-intensity and transitions to weak-ECC only if the access-intensity exceeds a threshold. While incurring only small performance loss, this strategy avoids ECC transition in several applications which have small memory footprint and hence, are insensitive to memory latency. Overall, their technique brings large reduction in refresh power in idle mode, while achieving performance close to a design which does not use ECC.

Jian et al. [56] note that CPK-correct schemes provide a fixed protection-level irrespective of the memory age. However, field studies reveal that in a year, only a small fraction (e.g., 3% [31] to 8% [32]) of DIMMs show faults and a majority of faults impact a small portion of the DIMM. Hence, even in worst-case, only a small fraction of pages show faults during the operation lifetime of 5 to 7 years. Based on this, their technique initially uses a weak-protection scheme and transitions to strong-protection scheme on a per-page basis as the pages start showing faults. The weak and strong protection schemes are realized using CWs with different number of check-symbols. This is because, in a symbol-based linear block code, the protection strength increases with increasing number of check-symbols per CW, e.g., with 2 check-symbol

Table 5
Strategies for reducing overheads of reliability techniques.

Category	References
Reducing latency overheads	
Caching ECC bits	in invalid subsectors of LLC [61,67], in data array of LLC [14,50,66] or in a separate cache [43,45,46,70]
Storing a cacheline in	a single chip [62] or a single bank [50] for improving performance
Storing data and corresponding ECC in	different memory devices for improving reliability [43,71], same row to improve row-buffer locality [15,68,72]
Row-buffer locality-aware optimizations	[15,60,68,71,72]
Reducing energy overheads	
Reducing refresh operations	[4,39,45,47,48]
Writing-back only dirty words	[61,67]
Reducing storage overheads	
Compressing data to make space for ECC	[73,77]
Providing reliability even with non-ECC DIMMs	[9,14,59,72,73,77]
Use of Bloom filter	[40]
Adapting granularity of sparing (row or bank)	[50]
Exploiting properties of "linear" block codes	[46,56,61,64]
Sharing same set of checksums between multiple lines	[46]
Use of virtual memory support	[14,54,71]

Table 6
A comparison of works that use adaptive protection-strength and protect selective memory portions.

Category	References
Adapting ECC strength over time	based on utilization (busy/idle) [47], occurrence of faults [56], access granularity [54]
Adapting ECC strength in different memory portions	based on data-criticality [59,60,72]
Granularity at which protection-strength is varied	DIMM [60], a pool of pages [59], portion of the row [72]
Storing correction bits of only faulty memory regions	[66]

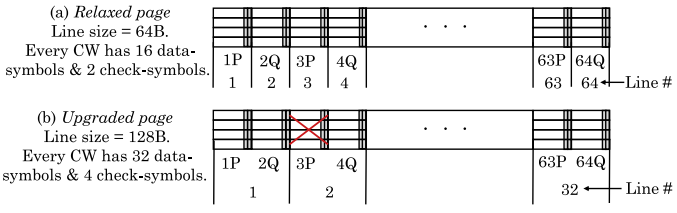


Fig. 22. Data layout of a page in relaxed and upgraded CPK-correct modes [56]. The letters 'P and 'Q after a line number show which channel the line belongs to. Each shaded rectangle shows a check symbol in a CW. The line with the cross has a fault, due to which the page is upgraded.

per CW, presence of 2 bad symbols in a single CW may go undetected, however, with 4 check-symbols per CW, the second bad symbol in the CW can be detected and even corrected [14].

They refer the pages with 2 and 4 check-symbols per CW as relaxed and upgraded pages, respectively. Fig. 22(a) shows a memory system with 2 channels (P and Q), each of which can independently serve a request for a 64B line. A 4KB page has 64 64KB lines and the adjacent lines are mapped to different channels to reduce their access latency. Every 64B line is composed of 4 CWs, which are delimited by the horizontal lines in Fig. 22. Each CW is composed of 16 data and 2 check symbols (shown as the shaded region). In a relaxed page, the 18 symbols of a codeword are stored across 18 DRAM devices.

On detecting an error, their technique strengthens the protection by increasing the number of check symbols per CW from 2 to 4. For this, 2 neighboring 64B lines of a page are combined in a single 128B line (called an upgraded line). Every CW of the 128B line has 32-data and 4-check symbols. Since the two 64B lines of a 128B line can be read in parallel from two different channels, the 128B line can be read in the same time as a 64B line. Fig. 22(b) shows one combination strategy, where every symbol maintains its original size and the number of CWs in upgraded line is same as that in a relaxed line. Thus, the ECC strength is enhanced without an increase in the storage overhead. In another combination strategy, the size of every symbol is reduced by half, which doubles the number of CWs in each upgraded line.

Their technique can be applied with CPK-correct schemes (e.g., [14,15]) to achieve reliability same as always using 4 check-symbols while incurring power consumption of using only 2 check symbols per CW.

Future challenges: A limitation of adaptive schemes is that they increase design complexity. For example, the technique of Jian et al. [56] requires modification of LLC to hold both 64B and 128B lines. Similarly, the technique of Chou et al. [47] requires converting between weak-ECC and ECC-6 dynamically. Evidently, the adaptive techniques are effective only when their benefits exceed their overheads and when such adaptions happen infrequently. Improving the efficiency of these techniques will allow them to be used in systems ranging from mobile systems to high-end servers.

5.2. Protecting selected memory regions

We now discuss the works that protect only selected portions of memory. Of these works, Malek et al. [59] and Luo et al. [60] focus on organization of data and ECC-bits when protection-requirement is reduced/alleviated. By comparison, Cao et al. [72] focus only on address mapping policies.

Malek et al. [59] present a technique which allows selecting the fault-tolerance of each allocated page based on the criticality of its data. Their technique provides three levels of protection: P0 (no protection), P1 (single-bit error correction, multi-bit error detection) and P2 (multi-bit error correction, up to failure of one entire chip). They first discuss the generic DRAM architecture with their technique and then discuss its implementation with 2D DRAM and 3D DRAM. As for the generic architecture, shown in Fig. 23, the DRAM block is organized as 8 rows, each of which stores 8 cachelines (512B). The 64 cachelines of a page are striped across 64 blocks at the same location. A pool of pages has 64 blocks, each of which has a cacheline from 64 different pages. This is the organization of P0.

In P1, one cacheline stores the ECC for the remaining 7 cachelines in the row, which leads to an ECC storage overhead of 14.3%. In practice, for each of the 7 protected cachelines, 8B ECC is stored next to them, and thus, 8B (=64-7*8 bits) out of 64B ECC cacheline remain unused. MC performs the required address re-coding so that, to the OS, the ECC

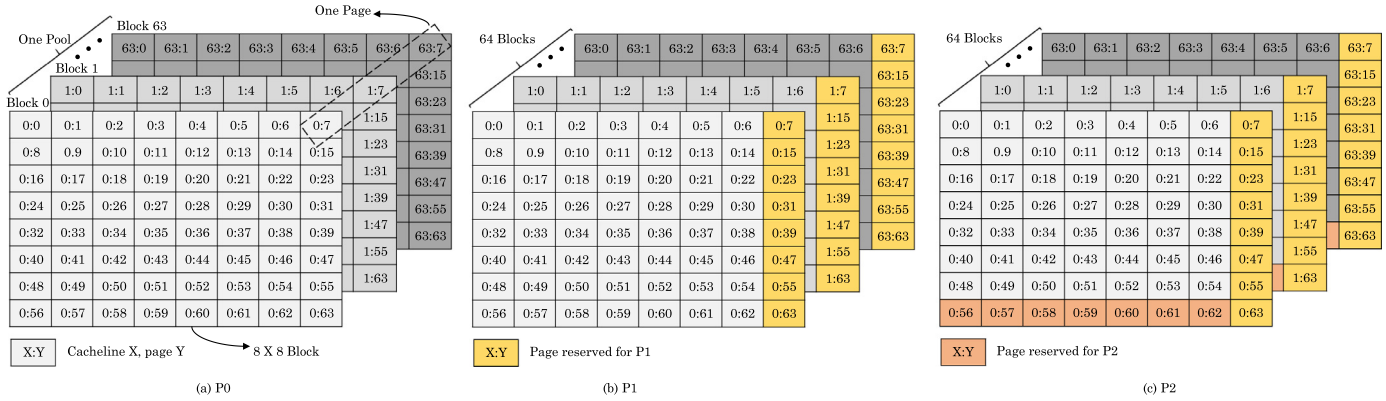


Fig. 23. Organization of pages and cachelines in one pool consisting of 64, 8×8 blocks [59]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

appears to be present in a single cacheline. Thus, the ECCs are stored in separate physical pages and are not visible to the user. Yet, these ECCs are aligned with the corresponding data to enable efficient access. As for the specific ECC, they use 16b SSC-DSD with strong 32b CRC, which requires only 48b (6B) storage.

In P2, compared to P1, one extra cacheline in the last block-row is used which stores the bitwise parity of the above seven cachelines for protecting them. Thus, only 49 out of 64 pages store data, which leads to an ECC storage overhead of 30.6%. To provide CPK protection, each P2 parity is followed by 1B interleaved parity of its own bytes. The MC allows dynamic burst lengths of 8 and 10 for accessing P0 and P1/P2 cachelines, respectively. The P2 code is accessed when P1 code detects a fault or while writing P2 protected data. In their technique, the latency of accessing ECC remains same as that in the uniform protection scheme. The OS manages the physical memory at the granularity of 256KB block (=64 * 4KB page) for reducing the storage and access latency overheads. Corresponding to P0/P1/P2, there are 3 types of blocks and depending on the protection level of a page, a frame is allocated from the suitable block.

They analyze how the faults in different data regions, such as stack, heap and global, etc. impact the application reliability. From this, they find the regions with high and low criticality and provide them stronger protection (P2 or P1) and weaker protection (P1 or P0), respectively. Compared to a technique which provides uniform protection to the entire memory, their technique reduces ECC storage overheads while achieving same reliability and also bringing minor improvement in performance and energy efficiency.

Luo et al. [60] present a technique which allows trading-off reliability to increase memory capacity. Their technique allows converting selected memory regions to no ED/EC mode or ED-only mode, whereas remaining regions continue to support both ED and EC. Providing only ED and no ED/EC increases memory capacity by 10.7% and 12.5%, respectively. For no-EC mode, they propose 3 designs; recall that the conventional SECDED scheme uses 9 chips to provide 64b data and 8b ECC. (1) In the first design, additional data pages are packed into ECC chip, as shown in Fig. 24(a). This design does not require any change in the ECC chip. However, its limitation is that reading the page stored in ECC chip requires eight 8B consecutive reads, since only a single chip provides the entire 64B data. Also, since a write to any page will modify the page stored in ECC chip, to avoid this, all write operations must now be performed as read-modify-writes. Further, row-buffer locality of regular pages is destroyed by an access to the page stored in ECC chip, although their data are stored in different chips. This is because all chips in a rank operate in lockstep manner. (2) In the second design, the layout remains same as that in the first design but here, rank-subsetting scheme is used. The first-eight chips operate in lockstep manner as x64 non-ECC DIMM. The ECC chip works as an x8 rank subset. Similar to the first design, this design requires eight accesses to read the additional page. However, this design avoids destruction of row-buffer locality and the requirement of read-modify-write on write operations since only the subset of chips holding the cache line data are enabled on a memory access. (3) In this design, all cache lines, including those in the additional pages, are striped across eight chips, as shown in Fig. 24. Hence, extra memory accesses are not required for any cache line which minimizes the memory latency. Also, this design improves bank-level parallelism since nine parallel requests can be served at each time, compared to eight in the baseline ECC DRAM. Thus, this design improves both capacity and performance, although it requires significant reorganization.

For ED-only memory regions, they propose a layout for providing 8b parity. Their proposed layout is shown in Fig. 25. In a bank, each of pages 0 to $N-1$ occupy one row across chips 0 to 7, as in the baseline DRAM. In chip 8, first the parity information is placed and the remaining space is used for allocating extra pages (page N onwards) which are spread across eight rows. Chips 0–7 and chip 8 operate in different sub-ranks. For the first N pages, read operation requires reading the data from chips 0–7 and the parity from chip 8. Write operation requires read-modify-write for avoiding modification of parity for unmodified cache lines. For the additional pages, a read request generates eight read operations for accessing the data and a ninth read operation for accessing the parity. A write request generates eight write operations for data and a read-modify-write for saving the parity data. For applications with reduced-reliability requirements, their technique improves performance by virtue of providing increased capacity.

Cao et al. [72] present a selective protection scheme which partitions DRAM rows into two parts, one with embedded ECC and one using no ECC (i.e., no protection). They note that the row-based partitioning can provide finer granularity than bank/rank/DIMM/channel-based partitioning and is especially effective for systems with small amount of memory. The column-based partitioning does not provide any benefit over row-based partitioning, but merely reduces row-buffer locality. A memory data block is mapped to protected or unprotected part using an address comparator and two mapping units. In the protected part, one 8B ECC chunk is stored in a column after every eight data columns. Thus, the data and corresponding ECC can be read from the same DRAM row. Thus, accessing ECC does not require additional precharge or activation operations.

The boundary between the two parts can be adjusted, such that they may have non-power-of-two rows/columns. This, however, complicates device-level address mapping. In the mapping function, use of division operation leads to high latency and the use of modulo operation is inefficient due to the variation in divisor on the change in boundary between the two parts. Hence, they propose “parameterized BCRM” design where the divisor is a configurable parameter. They present strategies to optimize its hardware implementation. To further lower the hardware

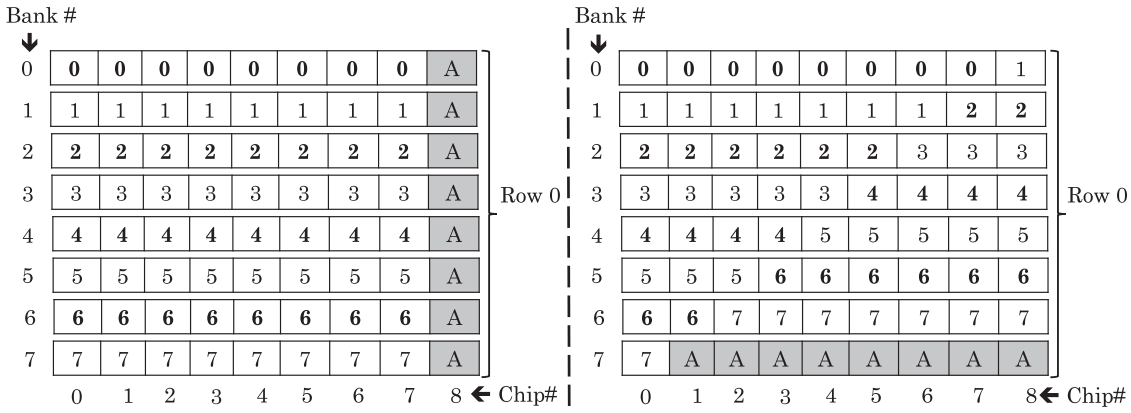


Fig. 24. (a) Packed data layout used in the first and the second design [60]. In the baseline scheme (not shown), chip 8 exclusively stores ECC, whereas in this layout, chip 8 stores an extra page (page A). (b) Inter-bank wrap-around layout used in the third design [60]. Notice the layout of page A.

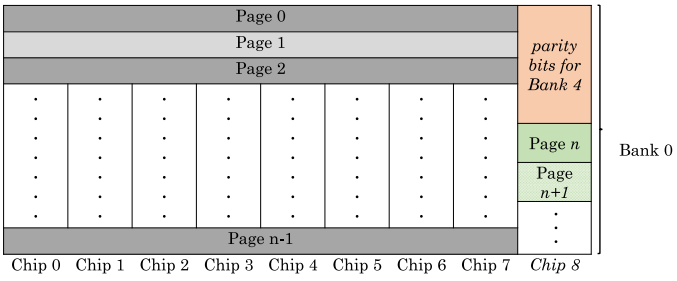


Fig. 25. Layout within an entire bank for 8b parity per cacheline. Pages 0 through $N-1$ require one row each, across chips 07. Pages N and $N+1$ show allocation of extra capacity, similar to the packed data layout shown in Fig. 24(a).

complexity, they propose a "segmented BCRM" design which uses non-continuous segmented physical address space. In this design, the ECC region physical address space is extended to the next larger power-of-two address. Then, this extended address space is divided into equal, power-of-two size segments. Both the parameterized and segmented designs have the same latency and row-buffer locality. Compared to the scheme which protects entire memory, their technique saves energy with negligible impact on the performance.

Future challenges: A limitation of selective-protection based techniques is that their benefits reduce greatly as the critical data becomes an increasing fraction of the total data. Moreover, they may constrain the options for placement of data, e.g., critical data can be stored only in the memory region that has strong protection. However, these constraints may interfere with the address-mapping and scheduling policies, which may harm performance. Clearly, the design and evaluation of selective-protection schemes need to account for its impact on other memory management schemes also.

5.3. Adapting the access granularity

For the same DRAM chip type (e.g. x4, x8, or x16), ECC techniques with large access granularity offer stronger memory protection, e.g., quad-chipkill, which has the access granularity of 256B, has 80X higher MTTF than single-chipkill, which has an access granularity of 64B [54]. However, use of large access granularity wastes memory access power and bandwidth for workloads with poor spatial locality. Table 7 compares different techniques for using adaptive access granularity. We now discuss techniques which adaptively perform CG or FG accesses to balance these factors.

Li et al. [54] present a technique which adapts memory access granularity and ECC mechanism based on memory characteristic of the workload. They use 72b memory channel with 64b data and 8b ECC. They provide three memory modes for systems designed with x4, x8 and x16 DRAM devices which are shown in Table 8.

They use a data-layout which does not change in different modes. Specifically, the data are interleaved at 256B boundary among all the MCs, then 64B data blocks are interleaved across all four 64b channels in the same MC. Thus, on a mode-switching, only ECC needs to be reconstructed. In their design, accesses with different granularities come to the MC. Since medium- and coarse-grained requests need to access multiple channels, they are likely to get lower priority and hence, see unfair slowdown. To avoid this, these requests are prioritized when they are delayed due to the pending fine-grained requests.

Adapting the access granularity at per-access and per-application level is too fine-grained and coarse-grained, respectively. Hence, they adapt the granularity in virtual memory manager which manages the mode for every physical memory page and provides this information to the memory controller. The adaptation of modes can be done statically based on compiler hint or dynamically based on runtime information such as failure rate, spatial-locality and usage of cache and memory channel. Of these, they experiment with a static scheme only. Since multi-granularity memory access also requires multi-granularity cache hierarchy, their baseline cache uses 64B cache lines; and for a 128B/256B data access, two/four (respectively) 64B cache lines are allocated in the same set of the cache which share the same tag. Compared to a system with fixed access granularity and ECC, their technique improves the energy-delay product significantly.

Yoon et al. [71] present a technique for adapting memory access granularity based on spatial locality. The programmer provides information about preferred granularity using annotations and compiler hints. By augmenting virtual memory, access-granularity can be specified at per-page level. To support both CG (64B) and FG (8B) accesses, they use sectored cache. Also, they use a subranked memory system [17]. As shown in Fig. 26, an 8B FG request is served by a burst of 8 transfers from a single x8 chip and a 64B CG request is served by a burst of 8 transfers from eight x8 chips of a rank. An FG access requires 8X more memory requests to obtain same memory throughput as the CG access, hence, the FG accesses may saturate the address/command bus. To resolve this issue, double data rate signaling can be used.

When both CG and FG requests are scheduled, CG requests may be unfairly slowed-down. To avoid this, CG requests can be prioritized or split into multiple FG requests such that the CG request is assumed to be serviced when all the component FG requests are serviced. An FG physical page is twice as large as the CG page, e.g., 8KB of storage for a 4KB data page. For FG request, 16B (8B data + 8B ECC) access is performed, thus, an FG access has low storage efficiency. Thus, different addressing is used with CG and FG accesses. The data and corresponding ECC are

Table 7
A comparison of works that use adaptive access granularity.

Category	References
Access granularity used	8B/64B [67,71], 16B/64B [61], 64B/128B [56], 64B/128B/256B [54]
Scheduling policies for avoiding slowdown of CG requests due to FG requests	Prioritize CG requests over FG requests [54,71], split CG requests into multiple FG requests [71]

Table 8
Memory access modes in the technique of Li et al. [54].

	Description	Access Granularity.	x4	x8	x16
Fine-grained	Every channel is 64b wide	64b	CPK	SECDED	SECDED
Medium-grained	Two memory channels are tied	128b	D-CPK	CPK	DEC
Coarse-grained	Four memory channels are tied	256b	Q-CPK	D-CPK	CPK

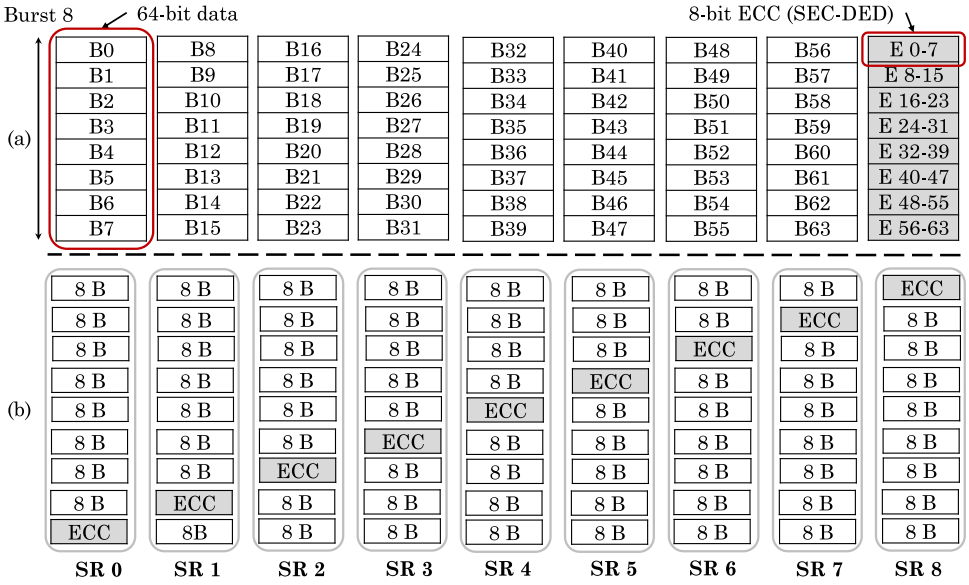


Fig. 26. Here, B*x* represents the *x*-th byte in a 64B block [71]. (a) CG access (E*y* – *z* is 8-bit SECDED ECC for data B*y* to B*z* (b) FG access (E*x* is 8-bit SECDED ECC for data B*x*.).

stored in different memory devices, which achieves stronger reliability than storing ECC in the same DRAM row as its data.

The decision about performing CG/FG access is taken based on the following 3 factors: (1) The average traffic for bringing a line from a page is 73B for CG access (64B data + 8B ECC + 1B control) and 17B for FG access (8B data + 8B ECC + 1B control). Thus, if less than 4 words of a cache line are accessed in a page, FG access can reduce the memory traffic. (2) With FG access, multiple accesses to the same line lead to cache misses (due to sectored cache design), which could be avoided with CG access. Hence, reducing traffic may not always improve performance, e.g., when memory bandwidth is not the bottleneck. (3) DRAM row-buffer hit rate: In case of low row-buffer hit rate, FG accesses allow quicker transition between pages and increase subbank level parallelism. However, a high row-buffer hit rate value reduces the benefit of FG access since the additional overhead of fetching an entire cache line is small. Hence, CG access does not harm memory throughput and can improve cache hit rate and load latency. Their technique reduces memory traffic and power and also improves performance.

Yoon et al. [67] present a design which uses same memory and ECC layout for both CG and FG accesses. Fig. 27(a) shows the data layout with their design. In their design, each 8b SECDED ECC is used for protecting 8B data from an independent, single DRAM chip over all bursts; this design supports both FG and CG accesses. Overall, 8B ECC protects the full 64B data, leading to the same redundancy cost as the traditional CG-only designs. To avoid bank conflicts in ECC chips from independent accesses, ECC blocks are uniformly distributed across the subranks, as in RAID5. This is illustrated in Fig. 27(b).

As for memory read, a CG access reads a 72B block including data and ECC. For FG access, one DRAM chip is accessed for 8B data and another for 8B ECC. Since this 8B ECC can correct errors in other data words, which are read later, their corresponding ECC bits are stored on-chip in the invalid subsectors of every line. On a miss to a subsector, the LLC controller, besides making FG request, sends the ECC from invalid subsector to the memory controller. Thus, the memory controller gets ECC from LLC instead of re-fetching it from DRAM. As for DRAM write, an FG writeback updates both 8B data-block and 8B ECC-block. ECC information is updated for only the words being written, but not for those corresponding to invalid subsectors of the cache line.

They use sub-ranked memory system to achieve FG (8B) access granularity. In their technique, FG accesses do not incur 100% ECC overhead, as in other works [71]. The decision about using CG/FG access is taken based on factors such as the average number of words accessed in every cache line (spatial locality within cache lines), row-buffer hit rate (spatial locality across cache lines), whether the majority of accesses at memory controller are CG/FG, etc. Compared to the technique of Yoon et al. [71], their technique reduces ECC-bit transfer when multiple words in a cache line are accessed. Their technique reduces off-chip traffic and memory power and improves system throughput.

Comments: In the technique of Yoon et al. [67] a strict isolation between FG and CG pages is not required and a page can service both FG and CG accesses at the same time. Hence, predicting and changing

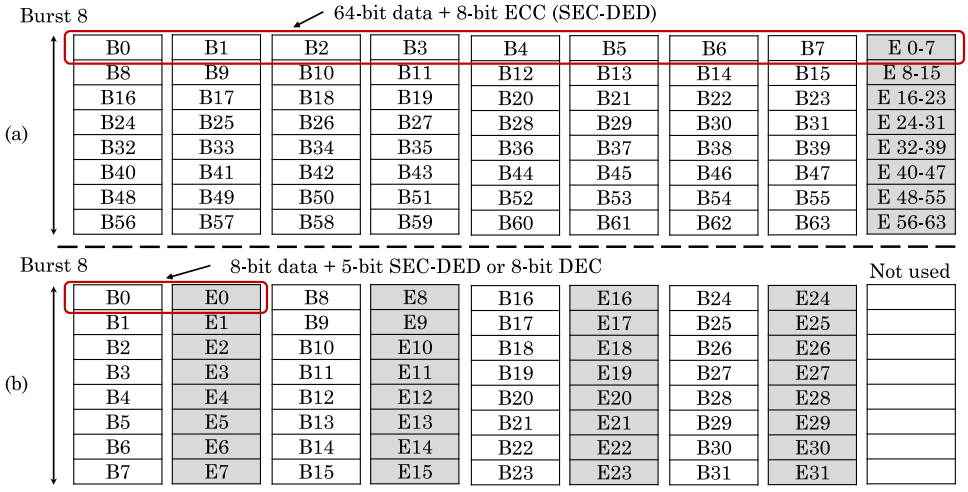


Fig. 27. (a) The data layout used by Yoon et al. [67] (B*x* shows the *x*-th byte in a 64B block, and E*y* – *z* is 8-bit SECDED ECC for data B*y* to B*z*. SR = subrank.) (b) Spreading of ECC locations to reduce bank conflicts.

access granularity does not require virtual memory support and thus, their technique can be implemented with only hardware-support. By comparison, the technique of Yoon et al. [71] requires virtual memory support for its operation.

Future challenges: The adaptive-access granularity techniques require scheduling requests of different granularities at the memory controller. Also, they may necessitate use of sectored cache which require additional metadata bits. Thus, these techniques introduce significant changes in both LLC and memory controller. Further, for applications with stable spatial-locality pattern, adaptive access granularity may not be required. Thus, extending the benefits of adaptive-access granularity techniques to a broad range of processor-architectures and applications requires will be a vital challenge for future research studies.

5.4. Compressing data to make room for ECC

Some techniques use data compression [78] to make space for ECC bits. The benefit of these schemes is that they do not change the block alignment and the addressing scheme, and provide reliability even with non-ECC DIMMs. Also, the effective memory capacity remains same and no separate access is required for ECC bits.

Palframan et al. [77] present a technique for protecting non-ECC DIMMs by using block-level compression for making room for ECC bits. Their technique requires only low compression ratio, due to which most (e.g., more than 90%) blocks can be compressed and the (de)compression circuitry becomes simple. Specifically, they compress a 64B block to 60B and add 4B of ECC. Further, each block is split into four (128,120) SECDED codes. Every code uses 1B ECC for protecting 15B data. This allows using simpler SECDED code, and easily distinguishing between a compressed (and protected) block and a uncompressed (and unprotected) block.

For determining whether a block is compressed, their technique ascertains presence of valid ECC CWs in a block read from DRAM. A valid CW is a 128b word, which on passing through a (128,120) SECDED decoder, emits a zero syndrome, indicating lack of errors. A block read from DRAM is treated as containing four (128,120) CWs, and is passed through the ECC logic, which is shown in Fig. 28(a). For a protected/compressed block that did not get any error, four CWs are detected with no errors. An unprotected block has 64B of uncompressed data and it is unlikely to contain a single valid CW and much more unlikely to contain multiple valid CWs. A block with 3 or more valid CWs is treated as compressed data. On a soft error in a protected block, three of the four CWs remain valid, and the fourth CW can be corrected using ECC before the block is decompressed. When less than 3 valid CWs are found, the block is considered as unprotected data and sent to LLC unmodified, as shown in the left part of Fig. 28(a).

In rare cases, a block of data with 3 or more CWs appear as compressed blocks to the decoder, although they are not. Such blocks are termed as aliases. The extremely rare blocks which are both incompressible and aliases are not stored in DRAM, as shown in Fig. 28(b), to ensure proper functioning of the decoder. These blocks reside in the LLC only. As for compression algorithms, they use modified versions of delta compression, run-length encoding and text compression. In one variant of their technique, uncompressed blocks are not protected, whereas in another variant, they are also protected using separate ECC. Their technique reduces DRAM soft-error rate significantly with negligible performance loss.

The limitation of their technique is that it provides poor reliability which is weaker than even SECDED protection [73]. Their technique tracks block compression state implicitly based on the number of valid CWs. While this approach suffices in case of single-bit faults, it does not work well in case of large-granularity faults. Specifically, chip, pin and rank faults into incompressible blocks lead to their decompression, which results in severe SDCs. Also, their technique cannot be easily extended to provide CPK-correct protection because their approach of tracking compressibility reduces the coverage of its ECC code [73]. The technique of Kim et al. [73] addresses these limitations.

Kim et al. [73] present a technique which mitigates ECC overheads by compressing main memory at cache-block granularity and using the saved space for storing ECC. Fig. 29 shows the read/write operations with their technique. On a write operation, data block is compressed. If sufficient space can be created for ECC bits, the compressed data and ECC bits are stored at the desired location. Thus, the beginning address of the block does not change, and both data and ECC bits can be fetched in a single access. On a read operation, decompression and ECC-verification happen in parallel since ECC is calculated on the compressed data. In case of no error, decompressed data are provided to LLC, else, lost data are recovered using ECC and decompression is performed again using the correct data.

For memory blocks, which cannot be compressed below a threshold-size, one entire block and one partial block is used for storing uncompressed data and ECC; this is termed as overflow data. As the number of such exception blocks increase, the performance/energy advantage of their technique reduces. The compressed and exception blocks are distinguished using flags, as shown in Fig. 30. Given the criticality of flags, strong protection is provided to them by using 3 to 5-bit flags. The value of the flag depends on the code design and compression-ratio of the block. They seek to free-up either 64b or 32b space for ECC. Thus,

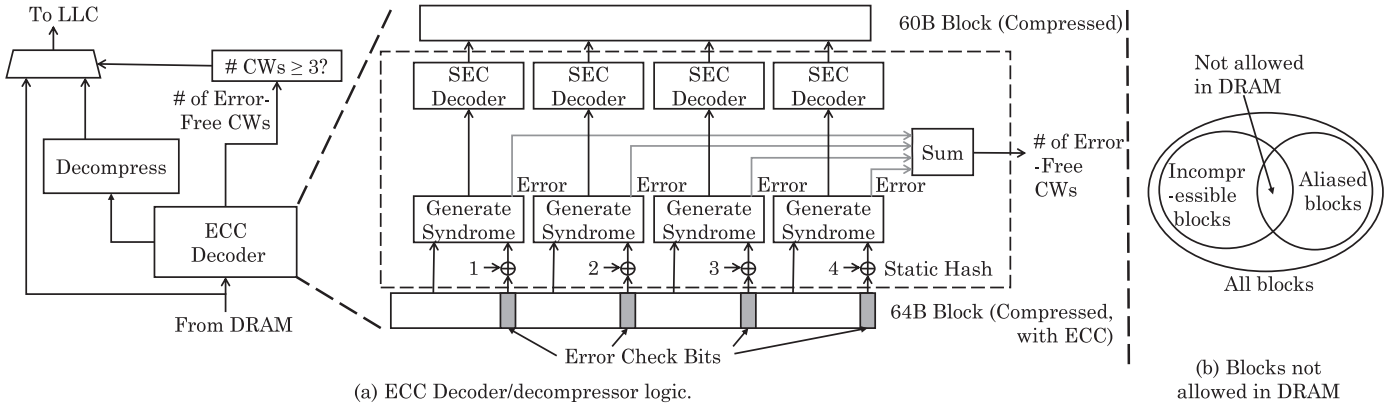


Fig. 28. (a) Working of ECC decoder/decompressor [77]. The number of valid error-free CWs seen are counted in the ECC decoder. If sufficient CWs are seen, the data (without ECC check bits) is decompressed and sent to the LLC, otherwise the data is passed unmodified to the cache. (b).

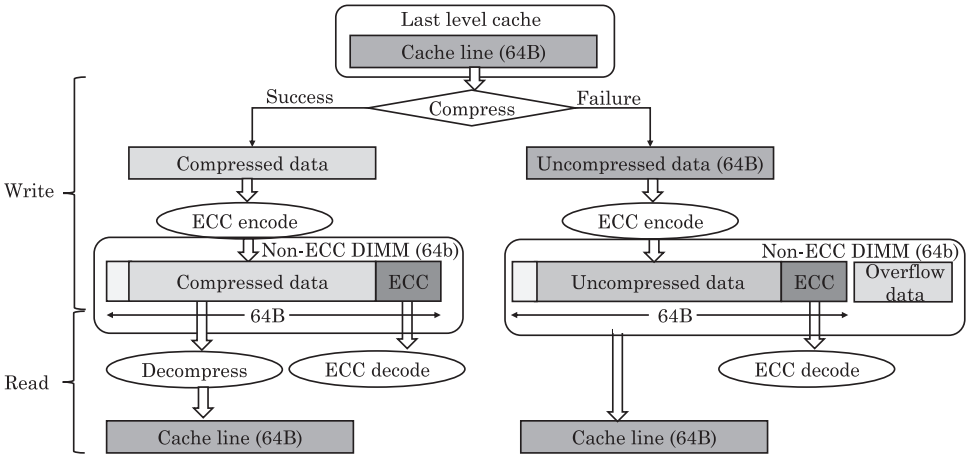


Fig. 29. Read and write operations with the technique of Kim et al. [73].

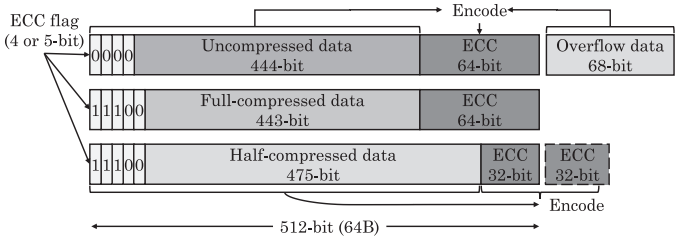


Fig. 30. ECC layout for CPK-correct (64-bit redundancy) in the technique of Kim et al. [73].

the flag shows one among three states: uncompressed, compressed (32b free) and compressed (64b free). The flag-bits are distributed over different chips to ensure that a chip error corrupts only a single bit of the flag and thus, can be corrected. Their technique also protects against errors in flag-bits.

They use a compression scheme which balances compression ratio with coverage, since uncompressed blocks incur memory-access/energy overheads, whereas excessive compression ratio provides no extra benefit since the extra space remains unused. This scheme uses delta compression scheme for homogeneous data (e.g., integers) and compresses only the sign and exponent bits in FP data. For heterogeneous-type data (e.g., `class`), delta compression does not provide high coverage since a single base cannot be found. For these incompressible data, their technique performs compression at 64-bit (qword) granularity. The qword is compressed as either a frequent pattern or as a delta from an earlier qword from the same block. Their technique can employ x4 chipkill ECCs to provide the same degree of reliability without using redundant devices. Also, it can employ x8 chipkill ECC to divide the 25% ECC overhead between the redundant chips and space freed by compression.

Future challenges: The limitation of compression-based schemes is that their effectiveness depends crucially on the compressibility of the data. Further, since different applications have different compressibility, use of compression approach complicates address-mapping scheme and/or may lead to fragmentation. To address these issues, the programmer can provide hints to the hardware about compressibility of the data and based on this, the reliability technique can reserve just right portion of memory for storing ECC of incompressible data.

5.5. Reusing ECC bits to improve memory performance

Nguyen et al. [79] note that memory systems in server processors use strong resilience scheme such as CPK that can tolerate up to failure of entire chip. Since the redundancy is provisioned to guard against the worst-case failure, it remains unused when there are no/few failures. Nguyen et al. propose refreshing a part of data of the memory block at a time, and using the ECC bits to compute the data, that is inaccessible due to being refreshed, for satisfying read requests. Since their technique refreshes only part of the data of a block at any time, it operates more frequently in the background to complete all the refreshes required, as performed by the traditional refresh scheme. Thus, their technique removes the impact of refresh on read-operations, which makes DRAM appear similar to SRAM at the system-level. Their technique improves memory performance for various levels of redundancy and failure pro-

Table 9
Salient features of techniques based on IECC.

Category	References
Integrating IECC with	RECC [4,40], DRAM-cells based sparing [4], SRAM side-cache based sparing [40]
Combining the storage space of IECC and RECC to design a single strong code	[9]
Exposing errors detected by IECC to MC	[38]

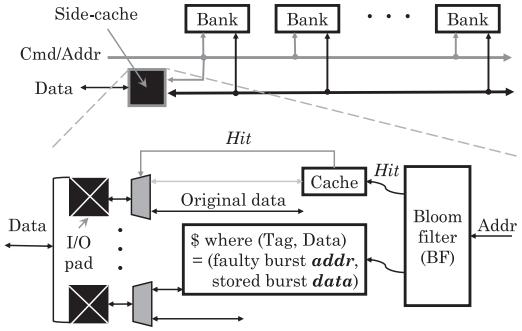


Fig. 31. Overall architecture in the design of Son et al. [40].

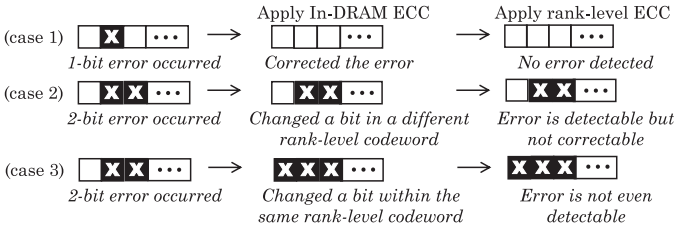


Fig. 32. Interaction between IECC and RECC. Case 1: IECC corrects a 1b error. Case 2 and 3: For a 2b error, IECC may change a correct bit, which leads to 3 faulty bits spread in the corresponding RECC codewords, such that RECC can detect but cannot correct the error (case 2) or cannot even detect it (case 3).

tection and also outperforms refresh-skipping schemes. Their technique allows using stronger ECC while maintaining the same level of performance.

Future challenges: Although using redundancy-bits for avoiding read-stall at the time of refresh operations obviates the requirement of separate redundancy bits, but the existing redundancy bits can be used only when no hardware errors need to be corrected. Also, these techniques are suitable only for server memory systems that use *strong* resilience schemes such as CPK, and not for the systems that use only error-detection or single-bit error correction schemes. Nonetheless, the techniques such as that of Nguyen et al. [79] propose a novel and interesting trade-off between ECC strength, storage/energy overhead and performance *benefit*. Fully exploring this design-space will be a worthy challenge for architects in the time to come.

6. Techniques based on in-DRAM ECC

Table 9 summarizes the salient features of techniques based on IECC. We now discuss these techniques.

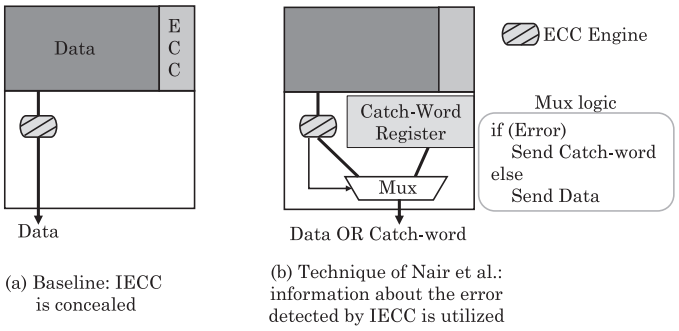
Son et al. [40] propose placing a small SRAM cache (termed "side-cache") in a DRAM device to which the accesses made to faulty DRAM locations are redirected. The tag array of side-cache stores the faulty-bit addresses and the data array stores their data value. Even though the SRAM density is much lower than DRAM density, the number of DRAM cells required for each faulty bit (in row sparing scheme) is much higher than the SRAM cells in the side-cache. Hence, use of side-cache incurs less overhead than the sparing scheme. Fig. 31 shows the architecture of their technique. The side-cache takes the command and address information in parallel with a DRAM bank access, and replaces it if the access is made to faulty DRAM cells.

For a fixed side-cache capacity, increasing its associativity reduces the probability of cache being full. Since DRAM banks and side-cache are accessed in parallel, and DRAM access is much slower, side-cache access latency is easily hidden. However, with increasing associativity, the energy to read the tags and compare them with a candidate address also increases. To reduce this, they use a Bloom filter which finds whether an incoming address matches the addresses of faulty DRAM cells. The Bloom filter shows false positives whereby some non-faulty cell addresses may be flagged as faulty, however, these are easily handled by the side-cache. Due to the low BER of DRAM, a small filter suffices. Thus, the size and access-energy of the filter are much lower than that of the side-cache. The Bloom filter uses multiple Hash functions to reduce the probability of false positives. If their latency is lower than a threshold, then they can be evaluated in two groups, such that the second group needs to be evaluated only if the first group indicates that the address is present. Further, the sets of side-cache are divided into multiple subsets. Each subset is sequentially evaluated until a hit is observed or all the subsets have been evaluated. This increases effective associativity without increasing the access energy in the case of a hit.

The limitation of the above design is that it requires dozens of SRAM cells for each permanent bit failure. To resolve this issue, they propose integrating the IECC with side-cache and Bloom filter. IECC corrects most of the errors while the side-cache handles the remaining infrequent errors. A DRAM bank is divided into fixed-size regions and IECC repairs one bit error in a region. In the rare cases of two or more bit errors, the side-cache is used for correcting the remaining errors. Since IECC is used for fixing most but not all bit errors and most regions are expected to have no more than one bit error, the area overhead of IECC remains low. To reduce the performance and energy overheads, they use the following scheme.

The conventional IECC scheme protects each word of a device, which requires accessing all message and parity bits of a CW for checking the error, even though only a single DRAM burst in a message with multiple bursts may be delivered externally. Instead, they group multiple DRAM bursts in a region and fix only one burst of the region with the parity bits. For associating the faulty locations with parity bits, fault-position bits are used. On each read, only one DRAM burst, position bits and parity bits are accessed and thus, their IECC requires reading much lower number of bits than the traditional IECC. Since a codeword is composed of one burst and its parity bits, the old value of the burst need not be read for updating the parity bits. This saves write bandwidth and brings it equal to the read bandwidth. By changing the region size, the number of correctable single-bit errors can be controlled.

They further note that IECC needs to work in sync with RECC, however, traditional IECC designs do not meet this requirement. Generally, RECC uses SECDED and IECC uses SEC (single-error correction). Now, if there is a faulty bit in a burst of a device, it can be corrected by the SEC code of IECC, as shown in case 1 of Fig. 32. If an extra fault occurs in the burst, RECC is expected to correct this fault. However, since the SEC code of IECC has a Hamming distance of 3, it sees the CW with two-bit errors as a different valid CW with one-bit error. Hence, it corrects the CW to that valid CW, which differs from the original valid CW by three bits. Although there is an overlap between the CWs of RECC and IECC, none of them includes the other. Hence, two of these three-bit errors may belong to a single RECC CW, and then, the RECC can de-



(a) Baseline: IECC is concealed

(b) Technique of Nair et al.: information about the error detected by IECC is utilized

Fig. 33. In the technique of Nair et al. [38], DRAM provides the catch-word or the data value, depending on whether the error is detected or corrected by IECC.

tect but not correct the error, as shown in case 2 of Fig. 32. If all three error bits are located in a single RECC CW, then RECC cannot even detect the error, as shown in case 3 of Fig. 32. To mitigate these limitations, they propose using either DEC or a code with a Hamming distance of 4 and leverage erasure or masking whereby the faulty bit-locations are detected by iterative reads and writes and then, corrected accordingly. They also discuss strategies to reduce the latency impact of IECC. Overall, for a given level of reliability, their technique incurs lower area and read energy overhead than row-sparing and traditional IECC schemes.

Cha et al. [4] present a technique to use IECC for tolerating DRAM scaling errors. Their technique targets randomly-scattered single-cell failures (SCFs). They explore multiple ECC types in IECC and find that double-bit EC and 8b-symbol EC are infeasible since they incur higher than 3ns latency. Hence, only (136,128) SEC and (72,64) SECDEC codes are suitable for implementation. For implementing (136,128) SEC code in x4 or x8 devices with burst length of 8, extra data bits are internally prefetched on read and write operations since (8*4b) and (8*8b) is smaller than the data size (128b) of the 136b CW. In (72,64) SECDED code, prefetching is required with x4 devices with burst length of 8. All the bits of a CW in a device are stored in the same row. On a write, only the incoming data is partially overwritten in a CW and the parity bits are re-computed for the whole 136b CW. Thus, prefetching requires a read-modify-write operation on every write. Multiple banks can concurrently perform read/write operations. They also adjust suitable DRAM timing parameters for ensuring correct operation with IECC.

They show that as fault rates increase, use of sparing scheme with IECC incurs much lower area overhead than use of sparing scheme alone. Thus, the former approach is more effective in handling randomly-scattered SCFs [4]. They provision a very small number of IECC CWs for correcting errors before device shipment and use the remaining CWs for correcting errors in the field. This improves the DRAM reliability significantly.

They further analyze the interaction of IECC and RECC and show that for IECC, (136,128) SEC is more suitable than (72,64) SECDED, since the former incurs much lower area overhead. To reduce the performance impact of using 136b CW, they propose long-burst write command, where the burst size equals to the codeword size of IECC. This avoids the need of read-modify-write operations. Overall, their technique improves DRAM reliability significantly.

Nair et al. [38] note that IECC allows correcting errors from faulty cells. However, some designs hide IECC information from the system to maintain compatibility with DDR standards and lower the bandwidth cost of sending IECC. However, due to this, a 9-chip ECC-DIMM does not offer any reliability advantage relative to a 8-chip non-ECC DIMM. This happens because memory failures are primarily due to large-granularity faults. However, on exposing the IECC information to MC, CPK-level protection can be obtained even with a 9-chip DIMM. They propose a technique for leveraging IECC information to provide stronger reliability.

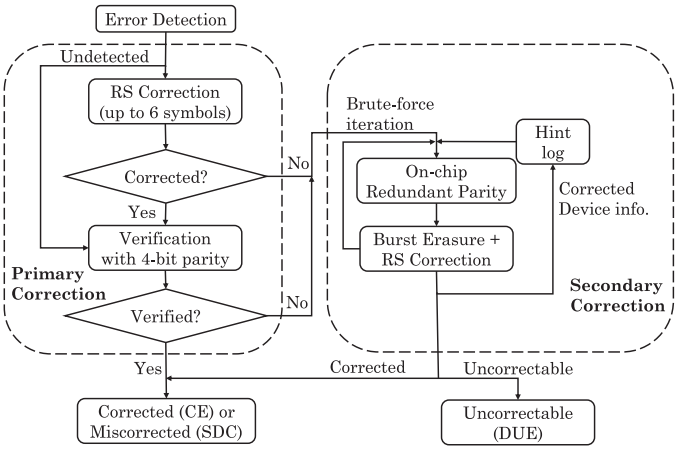


Fig. 34. The decoding procedure CPK-correct scheme of Gong et al. [9].

They note that the MC need not see IECC bits but it needs to only know whether it has detected an error. Then, this information can be used along with DIMM-level parity for correcting errors, as done in RAID-3. For communicating error information to MC, a catch-word is used which is a pre-determined random data value. When a chip gets an error, it transmits the catch-word instead of the data-value, which allows identifying the error location. Their technique stores the parity information of remaining chips in the 9th chip of the ECC DIMM. On encountering the catch-word, MC ignores the value of that chip and uses the parity from the 9th chip for recovering the data of the faulty chip.

Since a chip which provides 64-bit data-value cannot store all possible 2^{64} data-values, the probability of a chip storing the catch-word is extremely small. Their technique can also detect the rare case of a data-value matching the catch-word. In such a case, the catch-word can be changed and their technique can continue to function normally. Their technique improves performance compared to chipkill scheme. Also, with a chipkill system, their technique can achieve double-chipkill protection without incurring corresponding overheads.

Gong et al. [9] note that using IECC only for 1b EC or ED leads to weak protection. Instead, their technique bypasses IECC and combines the IECC storage space with RECC for achieving a single strong code. They add an “IECC bypass” mode which disables IECC and allows MC to directly access those bits. This leads to extra bus transfers but it avoids the overheads of IECC encoding/decoding. The bypass option resolves the size-mismatch issue between IECC CW granularity and data-transfer size which is responsible for data overfetch and associated penalties.

For low-reliability systems that seek to tolerate scaling errors (e.g., those due to VRT), their technique uses single-bit correction with BCH codes. Assume that a DDR4-like x16 non-ECC DIMM is used and thus, RECC is not used. IECC can correct at most 4 bit-errors as long as no two errors are present in the same CW. Instead, their technique uses triple-bit EC BCH code which, for 512b data, uses 30 redundant bits for correction and 2 bits for improving detection coverage, thus its 32b storage overhead is same as that of IECC. Although 3EC BCH corrects less number of errors than 4EC IECC, 3EC BCH can correct any 3b errors, which provides more flexibility. Also, 3EC BCH has much lower SDC rate.

For high-reliability systems seeking to tolerate FG scaling errors and CG operational errors, their technique provides CPK protection with symbol-based RS codes. This technique works with long ECC CW and has a primary and a secondary decoding scheme, as summarized in Fig. 34. Assuming 6.25% on-chip and 12.5% rank-level redundancy, a DDR4 x4 ECC DIMM can use an RS(76,64) 8b symbol code as the primary decoding. A single CW spans the entire transfer block. Every CW has 64

data symbols and 12 check symbols. The CW does not include 4b half-symbol redundancy which is used for improving the correction coverage (see below). RS(76,64) CW has 12 check symbols which can correct 6 erroneous symbols. The limitation of primary decoding is that it cannot directly handle a device failure in DDR4 x8 ECC DIMM since it cannot correct 9 erroneous symbols from a failed device.

For extending the correction capability of primary (RS) decoding, they use a secondary correction scheme which is activated when primary correction fails due to CG operational errors. As shown in Fig. 34, this scheme combines two strategies: (1) burst erasure decoding and (2) on-chip redundancy parity (ORP).

(1) Burst erasure decoding corrects both erasures and errors. When the location of the failed device is known, all these errors can be corrected as erasures. Erasure decoding requires only one symbol for correcting an erasure. In a DDR4 x8 ECC DIMM, failure of a device can be corrected with 9 check symbols. Of the remaining 3 symbols, 2 are used for correction of a scaling error and the third is used for improving detection coverage. For DDR4 x4 ECC DIMMs, 10 check symbols can correct all errors from two failed devices and 2 check symbols can additionally correct one error symbol.

On occurrence of an operational fault, the location of the erroneous symbol is not known. To deal with this issue, their technique serially performs burst erasure decoding in a "brute force search" manner, where each attempt assumes that a different device has failed. Since performing erasure RS decoding with an incorrect device is equivalent to trying to correct more errors than the correction capability of the RS code. Such cases are detected with very high probability and the decoder is mistakenly "successful" only when the code miscorrects. Such incorrect location results have negligible impact on reliability. Due to the rareness of operational faults and the efficacy of primary RS decoding, brute-force search is rarely required. To avoid repeated searches, the location of already-identified failed devices is logged in a hint log, which is useful for deciding where to begin the erasure decoding.

(2) For handling two scaling fault in a CW which also shows operational fault, remaining 4b redundancy in each codeword is utilized which is termed as ORP. ORP takes a checksum of the on-chip redundancy information across the entire memory block. For DDR4 x4 ECC DIMM, ORP calculates two 2-bit sums, each across the 4 2-bit blocks of an RS symbol stored in the in-DRAM redundancy bits. Before attempting burst erasure decoding, first the on-chip redundancy of the corresponding device is corrected with a RAID-like approach using the ORP checksum. In case of success, the amount of redundancy required for correcting the remaining failed chip(s) reduces from 5 check symbols to 4 for DDR4 x4 ECC DIMM. This frees an extra check symbol for each failed chip, which allows correction of at most two scaling errors in the typical case, in addition to CG operational faults. Compared to IECC, their technique reduces DRAM energy while maintaining or improving performance. Also, their technique improves reliability compared to IECC and other ECC-based techniques.

Future challenges: Given the rising memory-capacity demands placed on future memory systems, the storage overhead of IECC will become increasingly unacceptable. Given this, efficient techniques are required for optimizing IECC architectures and synergistically integrating it with RECC. This will also allow benefiting from a decade of research work done on RECC.

7. Techniques for mitigating retention failures

Khan et al. [3] discuss the effectiveness of three techniques for mitigating VRT-induced failures: online testing, guardbanding and ECC. (1) Testing: one testing round consists of writing DRAM module, waiting for the duration of refresh period and reading the data again. They note that although just 5 rounds can detect the majority of intermittent failures, a very small number of cells show new failures even after thousands of rounds. Thus, testing is not sufficient for detecting all the failures. (2) Guardbanding: a guardband K on the refresh period of 64ms implies that any DRAM chip which shows failure at (64*K)ms refresh period is discarded before shipping. This is because if cells with VRT show retention period less than the guardband, these cells will fail on using a refresh period of (64*K)ms. They note that although just K=2 value can avoid a large fraction (e.g., 90%) of failing cells, even K=5 is not sufficient for the remaining cells. Thus, guardbanding alone is not sufficient for mitigating all the failures.

(3) ECC: Use of SECDED ECC reduces the probability of failure by 10^2 times, but by combining ECC with testing and guardbanding, failure probability can be reduced by 10^{12} times. Thus, use of multiple techniques is significantly more efficacious than using a single technique. Further, the strength of the ECC required is dependent on the probability of the multi-bit failures. They also note that ECC based techniques can ensure reliable DRAM operation in presence of VRT-induced failures with only few hours of online testing, whereas bit repair schemes cannot ensure strong reliability even after months of testing.

Qureshi et al. [49] present a technique for mitigating errors due to VRT. They use two refresh rates: 64ms (fast refresh) and 320ms (slow refresh). Cells showing or not-showing error due to VRT are termed as strong or weak, respectively. A row with at least one weak cell is termed as weak row. They note that the initial testing identifies most of weak rows, and they are always refreshed with 64ms. Further, weak cells are uniformly distributed throughout the memory. However, even after multiple days, new bits continue to show errors due to VRT. They define a cell showing retention failure due to VRT in a time period as active-VRT cell. Also, the rate at which new cells become active-VRT is termed as active-VRT injection (AVI). The number of active-VRT cells varies dynamically over time and follows a lognormal distribution. Further, AVI reaches a steady value after few hours of testing.

They show that treating VRT-errors as soft-errors and using SECDED ECC to mitigate them is not acceptable since this solution leads to uncorrectable errors in 6–8 months. The reason for this is that while a soft error is removed immediately upon correction, a VRT-cell remains vulnerable for several hours. Hence, applying ECC to repeatedly correct these persisting errors makes it unavailable for correcting any new errors.

To address these limitations, they propose a VRT-aware multirate refresh scheme. Initially, retention periods are profiled. On an access, ECC DIMMs are used for detecting and correcting errors due to VRT. Their technique avoids accumulation of a large number of active-VRT cells by upgrading a row to fast refresh immediately when any word within the row shows an ECC error. This protects the row from further retention failures. Since ECC check is performed only on a memory access, memory regions with low activity may still accumulate VRT errors. To resolve this issue, their technique periodically (e.g., every 15 minute) performs scrub operations to identify VRT-induced errors during this period. These errors are corrected and the row is upgraded to fast refresh. Due to this, over time, the number of rows with fast refresh increases, although at a slow rate. To maintain the benefits of slow refresh, their technique infrequently (e.g., yearly) tests upgraded rows and if they do not show VRT, they are downgraded to slow refresh. Compared to the naive multirate refresh schemes, their technique improves reliability significantly with only small loss in performance and energy efficiency.

Lin et al. [48] note that retention-failures can be seen as hard errors and not soft errors. Also, very few DRAM cells have retention period smaller than 128ms. They propose a technique which performs offline profiling to identify cells with low retention period. Then, error-correction resources are allocated to only these cells and a large refresh period is used for saving refresh power. For error-correction, they use "error correcting pointer" (ECP) scheme [80] which incurs much lower overhead than ECC. ECP stores the address of a faulty cell and along with its remapping information. Their technique can also account for the impact of temperature on retention period and adjust the refresh rate accordingly. Overall, their technique provides large reduction in refresh power.

A DRAM cell is weak if it does not store data reliably in worst-case scenario, especially the highest temperature. When a large fraction of cells are weak, recording and reading their location information incurs high overhead. Further, due to small feature size, weak cells may not be precisely identified during offline testing and new weak cells may develop at runtime. Along with errors due to weak cells, the ECC also needs to protect against soft errors.

Wang et al. [65] present a technique for tolerating weak cells in 3D DRAM. They note that an ECC with the minimum Hamming distance of d can correct at most t errors and e erasures, where $d-1 \geq 2t+e$. For $e=0$, at most $t_{\max} = \lfloor \frac{d-1}{2} \rfloor$ errors can be corrected. If ECC CW length is K, then the K memory cells protected by one ECC CW are collectively referred to as CW-cell-group (CWG). Let t_R show the number of errors due to radiation and undetected weak cells, then, a CWG with at most $t_{\max} - t_R$ detected weak cells will never show more than $t_{\max}$ bit errors, and to protect against such cases, the ECC can be configured to operate in $t_{\max}$ mode by setting $e=0$ (i.e., no erasure correction). Most CWGs fulfill this requirment and for them, storing weak-cell location information is not required.

For CWGs with more than $t_{\max} - t_R$ deteted weak cells, erasure decoding needs to be used which requires storing information about the weak-cell location. To reduce this overhead, they observe that (1) under normal operating conditions, only few cells show errors. Also, since the probability of errors due to radiation and undetected weak cells (t_R) is low, a CWG with e weak cells generally sees much less than $t_R + e$ bit errors. (2) For an ECC with minimum Hamming distance of d which can correct and detect at most t and c errors, respectively, we have $d-1 \geq t+c$. If a CWG has at most $e_{\max} - j$ $(j \geq 0)$ weak cells, then this CWG sees at most $e_{\max} + t_R - i$ errors. Hence, the ECC can be used in $(t_R + i)$ EC and $(e_{\max} + t_R - i)$ ED mode. This works if the number of bit errors is at most $(t_R + i)$ and thus, the erasure decoding is not required. Here $e_{\max}$ shows the maximum number of detected weak cells that can be tolerated.

Based on these observations, their technique chooses an ECC with suitable d value. Also, the CWGs are divided into different sets based on the number of weak cells in them (N_{weak}). For the Set($N_{weak} > e_{\max}$), data reliability cannot be ensured and hence, the data is written to a separate small memory instead of the DRAM. For the Set($N_{weak} = e_{\max} + 1 - i$), where $1 \leq i \leq t_{\max} - t_R$, erasure decoding may be required and hence, weak-cell location should be stored. For the Set($N_{weak} \leq t_{\max} - t_R$), erasure decoding is not required and the ECC can work in $t_{\max}$ EC mode. For a read operation, their technique incurs lower latency than performing weak-cell information lookup.

Their technique is feasible only for 3D DRAM since the 3D DRAM can more easily tolerate the variable latency of a complex ECC technique and the ECC can be implemented in logic die which requires no modifications in host CPU. As for implementation in a 3D DRAM, they use one ECC encoding and ED engine in each vault. Note that in stacked DRAM, a DRAM die is divided into multiple partitions and all vertically adjacent DRAM partitions form a vault. Also, a group of vaults share an ECC error/erasure decoding engine. Their technique explicitly stores the location information of less than 0.1% of the weak cells and can also tolerate inaccuracy in detection of weak cells. With only 12.5% storage overhead, their technique can tolerate high-rate of weak cells if all or most (e.g., 90%) of the weak cells are known a-priori.

Future challenges: As researchers discover newer sources of error, the limitations of existing techniques continues to come to the fore. For example, it has been shown that due to the coupling between adjacent cells, the retention period of a cell depends on the data values stored both in the cell and its neighboring cells [3]. Hence, the techniques which do not account for the impact of data-pattern on error behavior (e.g., [49,65]) may not provide protection when a DRAM row stores unfavorable data pattern. Evidently, future reliability techniques need to be aware of data-values to also address data-pattern dependent errors.

Table 10
Location where faulty locations are remapped to.

Category	References
LLC	[55,57]
An SRAM-based side-cache in memory	[40]
A spare area in DRAM	[39,50]

8. Techniques based on repairing and hashing/checkpointing

In this section, we discuss techniques based on repairing (Section 8.1) and hashing/checkpointing (Section 8.2). Table 10 summarizes the salient features of these techniques.

8.1. Using repairing schemes

While ECCs are effective at correcting both soft and hard errors for low BER values, they become ineffective with increasing BER value. For example, to tolerate BER higher than 10^{-4}, 3b error correction for 64b word is required, which is quite costly and still requires extra ECC for tolerating soft errors. For such high BER values, repairing schemes can be used which obviate or relax the requirement of other resiliency schemes such as ECC. A repair scheme may either retire a faulty memory region or remap it to a spare area in the LLC [55,57], or to an SRAM-based side-cache in memory [40] or in a DRAM-based spare area [39]. We now discuss these repairing schemes.

Kim et al. [55] present a technique which uses parts of LLC for storing the retired memory regions. On each access to memory, first it is decided whether it should consult the memory or the alternative storage due to retirement of the target address. For retired addresses, a remapping table is queried for locating their data. These steps happen together as part of the regular LLC lookup. Requests from memory which indicate an error are tracked for identifying the candidates for repair. A memory location which is ascertained to have a permanent fault is retired and remapped to alternative storage. The LLC lines corresponding to physical addresses of the retired memory region are locked. A retired memory address always leads to LLC hit. By comparison, an unretired address may either hit in cache or miss in cache and access memory.

Their technique decides the memory regions to retire based on the number of already-retired memory addresses and the nature of error, viz., hard/soft-error. Hence, on detecting erroneous transfer from memory, their technique performs retirement aggressively since (1) the fault rates are low, (2) most faults impact only few cacheline-sized memory regions and (3) most hard errors cannot be easily differentiated from the soft errors. However, aggressive retirement incurs high overhead for only CG faults which impact several DRAM addresses spanning over multiple cachelines. For this, they note that since scrubbing repairs all soft-faults, retirement corresponding to soft-fault must happen within a single scrubbing interval. Hence, they record the number of retirements in a single scrubbing interval, as shown in Fig. 35(a). If this count exceeds a threshold, they further ascertain whether a CG soft-fault happened. For this, they start the scrubber immediately and if no errors are found by the scrubber, the fault was a soft-fault. Hence, the memory addresses retired in previous scrubbing interval are un-retired. Fig. 35(b) shows the working flow of the scrubber. CG faults are not mitigated by their technique but with another higher-level technique.

Their technique can perform retirement at fine-granularity and does not incur extra latency for accessing retired memory. Their technique reduces the strength of ECC required. By using a small region of LLC (e.g., 8KB or more), their technique can cover a major fraction of DRAM faults. Also, a small reduction in LLC capacity (specifically, LLC associativity) has negligible impact on performance of most applications.

Kim et al. [57] present a technique which repairs memory faults by remapping those locations to few locations in the LLC. LLC stores both regular blocks and remapped blocks and they are distinguished by a bit

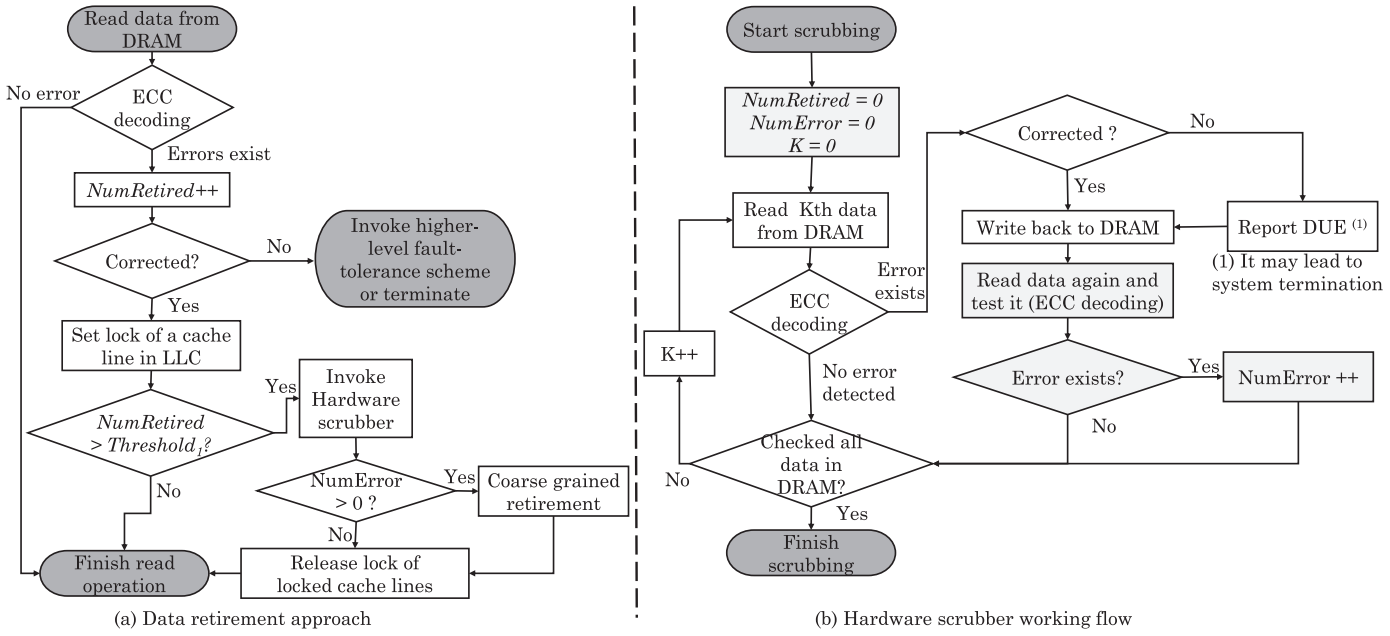


Fig. 35. Working flow of the technique of Kim et al. [55]. If an error is detected by the ECC, the cacheline associated with the error is marked as locked. If error events happen more than a threshold number of times in a single scrubbing interval, the hardware scrubber is invoked to detect if a CG fault had happened. After scrubbing, the locked lines may be unlocked if the fault was likely to be a soft-fault.

in the tag array. MC keeps a table which tracks repaired faulty banks. On occurrence of a fault, its address is computed based on the remapping logic. A cacheline in the LLC set corresponding to this address is evicted and then, this line is locked. The fault information is stored in the table. Since data of only the faulty memory device is remapped, every cacheline can store multiple remapped sub-blocks, e.g., 16 4B data. All the sub-blocks are filled on the first access to any data which should be remapped to the cache line. For this, memory requests are generated to those blocks and this needs to be done only once. After this, all accesses to these memory locations use their repair technique, even if every location may not be faulty. While accessing data from faulty DRAM device(s) which are already repaired, data from memory and remapped data need to be combined. This is ascertained by testing the table. If required, remapped data is obtained from LLC in parallel with the DRAM access. The combined data is filled back in LLC and further hits to that address do not require any intervention by their technique. LLC writebacks which include repaired addresses are handled in a similar manner.

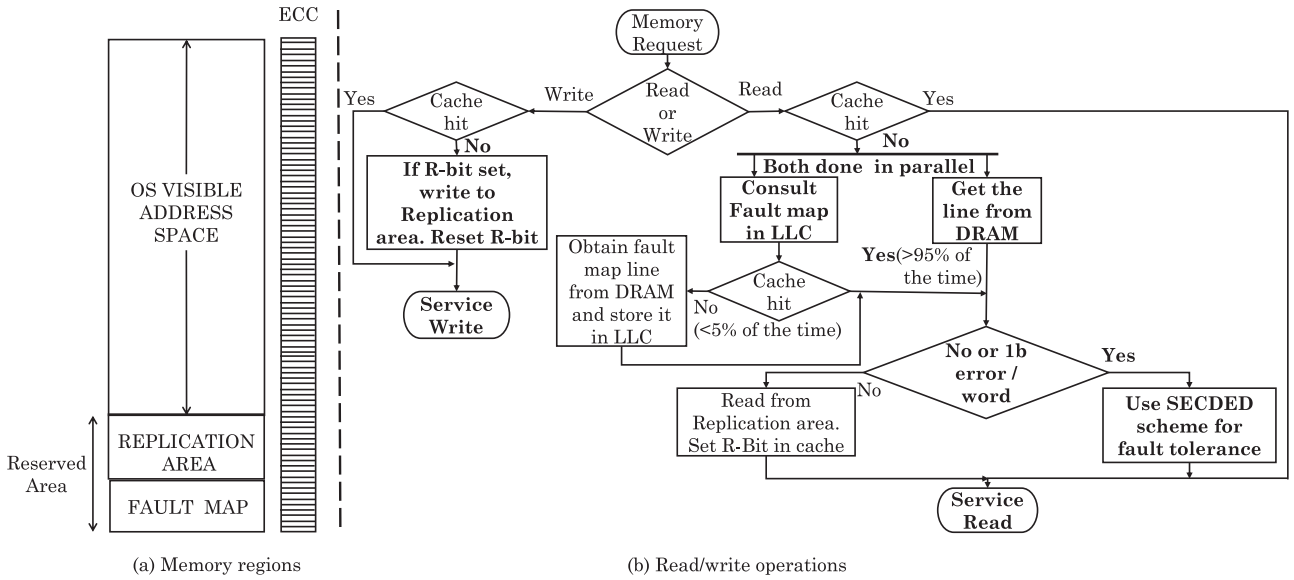
They note that most DRAM faults impact only a small number of bits in a single/few rows/columns of generally a single DRAM device. However, performance-oriented DRAM address mapping schemes spread these faulty bits to numerous cache lines. Due to this, previous repair techniques (e.g., [55]) lock many cache lines and incur high overhead. To address these limitations, they present a mapping strategy which leverages the correlations of fault locations to place as many bits as possible from a DRAM tile into a cacheline. For this, every column address is treated as referring to the data from a single memory device. Every LLC set stores multiple remapped cachelines which are distinguished by their tag value. Their strategy for coalescing remapped faulty bits decreases the number of cachelines used for repair and improves repair coverage. They also add XOR-based cache set hashing which reduces set conflicts and boosts coverage. Their technique reduces DUE and SDC rate significantly and improves system availability. For a similar repair coverage, their technique requires much less LLC capacity than the previous technique [55] and also limits the number of ways lost by each set.

Nair et al. [39] propose a technique for tolerating high BER (e.g., 10^{-4}) in DRAM. Their technique tailors resilience of a word based on its error-count. At the time of plugging a DIMM, their technique finds whether the 64b word has zero, one or more than one error(s). The words with single and multiple faulty cells are distinguished since the former can be tackled with ECC in case of no soft-error. The information about faulty words is stored in a structure called 'FaultMap' on per line basis. To lower the storage cost of FaultMap, the information about faulty words is maintained at cache line granularity (64B) which is the granularity of memory access. In a line, the word with the highest number of errors decides the fault-level of this line. Words with one or more errors are replicated in a spare region, called replication area, as shown in Fig. 36(a). Then, the words with multi-bit error can be retired and soft-error protection can be provided to those with one-bit error.

To reduce the latency of accessing FaultMap, it is cached in LLC. Fig. 36(b) shows the read/write operations in their technique. On a read miss in LLC, the request is sent to DRAM. At the same time, the LLC is accessed with the FaultMap address. Most of the time, LLC access leads to a hit, and the FaultMap entry is obtained. Otherwise, a second access is made to memory to fetch the FaultMap entry and then, this entry is stored in LLC. If the FaultMap entry shows that the line has no faulty cell, the data provided by DRAM is used. If the line has a word with more than 1b error, the replicated words for the line are obtained from the spare area and are incorporated in the line. Although this incurs extra latency, this case happens rarely. For lines having a word with only 1b error, the replicated copy is retrieved only when an uncorrectable fault is seen at the original place. This reduces the latency is common case.

They add a bit termed "R-bit" ("replication bit") in the tag-store of LLC which shows whether the line needs replication on writeback. The R-bit is set if, at the time of a read, the line had single or multiple faulty cells. For a word with single fault, writing to both the good location and the replicated location ensures protection from soft error. On eviction of a dirty line whose R-bit is set, the replication area is also updated. Their technique can tolerate high BER with only small performance and memory capacity overhead, while ensuring a soft-error protection of 1-bit error per ECC word.

Future challenges: A limitation of the techniques that remap faulty DRAM locations to LLC is that they make DRAM operation dependent on LLC. This may increase design complexity since LLC and DRAM may be designed by different companies. Also, it may affect the operation of



(a) Memory regions (b) Read/write operations

Fig. 36. (a) Memory regions in the technique of Nair et al. [39] (b) Read/write operations. The decisions in boldface show the most frequent path for requests on an LLC miss.

LLC. Moving forward, a thorough evaluation of remapping schemes is required on the operation and efficiency of both LLC and DRAM.

8.2. Using hashing and checkpointing

Chen et al. [53] note that "incremental hash functions" have the property that the overhead of updating the hash-result on an update to the original message is proportional to the update. They use multi-set hash functions which are incremental hash functions and their hash-result does not depend on the ordering of its elements. Based on this property, a fingerprint of memory access can be created. Their memory reliability technique detects error by keeping and cross-checking a read-log hash (RHash) and a write-log hash (WHash), which log the (address, data) pairs of memory reads and memory writes, respectively. On a memory read due to an LLC miss, the memory block is logged into RHash. On a cache replacement, the evicted block, whether clean or dirty, is logged in WHash. Although no memory write happens on eviction of a clean block, logging it ensures that the same group of (address, data) pairs are logged in RHash and WHash. Also, when the evicted block is again fetched in cache, the read will match the logged write at the time of eviction regardless of whether an actual memory write occurred at the time of eviction.

These hashes are updated at runtime and are synchronized periodically and at the end of program-execution. This operation is termed as "integrity-check". For every physical memory block used by the application, the OS checks whether the block is cached. If it is not cached, the block is loaded and its (address,data) pair is logged in RHash for ensuring that the same group of (address,data) pairs are logged in RHash and WHash since for each un-cached memory block, there is an (address, data) pair which has been logged in WHash but not in RHash. Thus, the above step synchronizes both the hashes. In case of no error, both the hashes match. Otherwise, an error is flagged and recovery (correction) is performed by rolling to the last checkpoint.

In their technique, a hash function is required for converting the (address, data) pair of a memory request to a hash value before using the multi-set hash function. Since memory errors are completely different from malicious attacks, cryptographic hash functions are not required. Hence, they use non-cryptographic hash functions which provide higher throughput and incur lower complexity than the cryptographic hash functions. Among the possible non-cryptographic hash functions, they choose SpookyHash function [81], since it has low implementation overhead, has avalanche property, is non-linear and is resistant to collisions which ensures strong detection capability.

In their technique, integrity-check operation is required before each checkpointing. This operation scans the whole memory region allocated to the application process for loading any block absent from the LLC. However, in case the allocated memory region is large, the overhead becomes large. To avoid this, they propose loading only those pages which were accessed during an integrity-checking period. Since the memory region allocation happens at beginning and only a small part of it is accessed in a period, this scheme reduces the memory traffic significantly.

Future challenges: A limitation of hashing-based approach is that it assumes that pages not accessed in a period are correct and hence, it cannot correct those pages. Due to this, for applications with only few memory accesses, this technique will leave a large portion of the memory unprotected. To address these challenges, their technique can be combined with the ECC for providing stronger protection.

9. Conclusion and future outlook

In this paper, we presented a survey of techniques for improving DRAM reliability. We organized the works on several axes to highlight their similarities and differences. We close this paper by pointing some interesting research directions that merit further investigation.

Since error is anathema to exact execution, they cannot co-exist in the conventional *precise computing* approach. However, the *approximate computing* approach [82] offers a paradigm shift in this mindset and embraces errors as the opportunity for enhancing efficiency. For example, for error-resilient data, ECC can be relaxed/skipped and scrub frequency can be reduced. While these initial intuitions are promising, fully benefiting from the approximate-computing approach will require rigorous evaluation of entire computing stack, beginning from DRAM-microarchitecture, ECC-designs, error-resilience of applications and users, etc.

In large-scale and mission-critical systems, ECCs alone may not provide desired level of reliability and hence, a suite of reliability techniques may be necessary, such as sparing, memory scrubbing, checkpointing, memory retirement, etc. Efficient integration of ECC with these techniques is vital to provide just right level of protection without over-provisioning of EC resources. Also, integration of DRAM with emerging non-volatile memories such as domain wall memory, phase change

memory and resistive memory [83,84] is essential to achieve higher memory capacity.

Since even a single error can lead to failure of an entire system [85,86] and can even be exploited by an adversary for launching attacks [87], achieving the tall reliability targets of next-generation systems demands the architects to be always on vigil. Evidently, the goals of memory reliability are likely to keep the designers challenged for more time to come.

Conflict of interest statement

The authors declare that they have no conflict of interest.

Acknowledgment

Support for this work was provided by Science and Engineering Research Board (SERB), India, award number ECR/2017/000622.

References

[1] V. Sridharan, N. DeBardeleben, S. Blanchard, K.B. Ferreira, J. Stearley, J. Shalf, S. Gurumurthi, Memory errors in modern systems: The good, the bad, and the ugly, in: ASPLOS, 2015, pp. 297–310.
[2] A.A. Hwang, I.A. Stefanovici, B. Schroeder, Cosmic rays Don't strike twice: understanding the nature of DRAM errors and the implications for system design, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2012) 111–122.
[3] S. Khan, D. Lee, Y. Kim, A.R. Alameldeen, C. Wilkerson, O. Mutlu, The efficacy of error mitigation techniques for dram retention failures: A comparative experimental study, in: ACM SIGMETRICS Performance Evaluation Review, vol. 42, ACM, 2014, pp. 519–532.
[4] S. Cha, O. Seongil, H. Shin, S. Hwang, K. Park, S.J. Jang, J.S. Choi, G.Y. Jin, Y.H. Son, H. Cho, et al., Defect Analysis and Cost-Effective Resilience Architecture for Future DRAM Devices, in: International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 61–72.
[5] S. Mittal, J. Vetter, A survey of techniques for modeling and improving reliability of computing systems, IEEE Trans. Parallel Distrib. Syst. 27 (4) (2015) 1226–1238.
[6] M. Scrbak, M. Islam, K.M. Kavi, M. Ignatowski, N. Jayasena, Exploring the processing-in-memory design space, J. Syst. Archit. 75 (2017) 59–67.
[7] S. Mittal, J. Vetter, A survey of techniques for architecting DRAM caches, IEEE Trans. Parallel Distributed Syst. (TPDS) 27 (6) (2016) 1852–1863.
[8] X. Jian, V. Sridharan, R. Kumar, Parity helix: Efficient protection for single-dimensional faults in multi-dimensional memory systems, in: High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on, 2016, pp. 555–567.
[9] S.-L. Gong, J. Kim, S. Lym, M. Sullivan, H. David, M. Erez, DUO: Exposing On-chip Redundancy to Rank-Level ECC for High Reliability, HPCA (2018).
[10] S. Mittal, A survey of soft-error mitigation techniques for non-volatile memories, Computers 6 (8) (2017).
[11] S. Mittal, A survey of architectural techniques for near-Threshold computing, ACM J. Emerg. Technol. Comput. Syst. 12 (4) (2015) 46:1–46:26.
[12] B. Jacob, S. Ng, D. Wang, Memory systems: cache, DRAM, disk, Morgan Kaufmann, 2010.
[13] J. Kim, et al., Strong, thorough, and efficient memory protection against existing and emerging DRAM errors, Ph.D. thesis, 2016.
[14] D.H. Yoon, M. Erez, Virtualized and flexible ECC for main memory, in: ASPLOS, 2010, pp. 397–408.
[15] A.N. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, N.P. Jouppi, LOT-ECC: Localized and tiered reliability mechanisms for commodity memory systems, in: ISCA, vol. 40, 2012, pp. 285–296.
[16] S. Li, K. Chen, M.-Y. Hsieh, N. Muralimanohar, C.D. Kersey, J.B. Brockman, A.F. Rodrigues, N.P. Jouppi, System implications of memory reliability in exascale computing, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011, p. 46.
[17] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, Z. Zhu, Mini-rank: Adaptive dram architecture for improving memory power efficiency, in: Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on, IEEE, 2008, pp. 210–221.
[18] JEDEC, HIGH BANDWIDTH MEMORY (HBM) DRAM (2015).
[19] H. M. C. Consortium, Hybrid memory cube specification 1.0 (2013).
[20] Tezzaron Semiconductor, Octopus 8-Port DRAM for Die-Stack Applications: TSC100801/2/4 (2010).
[21] S. Mittal, J.S. Vetter, A survey of software techniques for using non-volatile memories for storage and main memory systems, IEEE Trans. Parallel Distrib. Syst. 27 (5) (2016) 1537–1550.
[22] S. Mittal, A Survey Of Architectural Techniques for Managing Process Variation, ACM Comput. Surv. 48 (4) (2016) 54:1–54:29.
[23] A. Hocquenghem, Codes Correcteurs d'Erreurs, Chiffres (Paris) (1959) 2:147–156.
[24] R.C. Bose, D.K. Ray-Chaudhuri, On a class of error correcting binary group codes, Inf. Control 3 (1) (1960) 68–79.
[25] I. Reed, G. Solomon, Polynomial codes over certain finite fields, J. Soc. Industr. Appl. Math. 8 (2) (1960) 300–304.
[26] S.-H. Kim, W.-O. Lee, J.-H. Kim, S.-S. Lee, S.-Y. Hwang, C.-I. Kim, T.-W. Kwon, B.-S. Han, S.-K. Cho, D.-H. Kim, et al., A low power and highly reliable 400mbps mobile ddr sdram with on-chip distributed ecc, in: Solid-State Circuits Conference, 2007. ASSCC'07. IEEE Asian, IEEE, 2007, pp. 34–37.
[27] T.Y. Oh, H. Chung, Y.-C. Cho, J.-W. Ryu, K. Lee, C. Lee, J.-I. Lee, H.-J. Kim, M.S. Jang, G.-H. Han, K. Kim, D. Moon, S. Bae, J.-Y. Park, K.-S. Ha, J. Lee, S.-Y. Doo, J.-B. Shin, C.-H. Shin, K. Oh, D. Hwang, T. Jang, C. Park, K. Park, J.-B. Lee, J.S. Choi, 25.1 A 3.2Gb/s/pin 8Gb 1.0V LPDDR4 SDRAM with integrated ECC engine for sub-1V DRAM core operation, 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC) (2014) 430–431.
[28] Chipkill Memory. (https://goo.gl/Nfhrgg).
[29] Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability, (https://goo.gl/mJCGBQ).
[30] Oracle SPARC Server RAS Comparison, (https://goo.gl/MuxJbv).
[31] V. Sridharan, D. Liberty, A Study of DRAM Failures in the Field, in: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012, pp. 1–11.
[32] B. Schroeder, E. Pinheiro, W.-D. Weber, DRAM errors in the wild: a large-scale field study, in: ACM SIGMETRICS Performance Evaluation Review, 37, 2009, pp. 193–204.
[33] T.J. Dell, A white paper on the benefits of chipkill-correct ECC for PC server main memory, IBM Microelectronics Division 11 (1997).
[34] Enhancing IBM Netfinity server reliability.
[35] C.-L. Chen, M. Hsiao, Error-correcting codes for semiconductor memory applications: a state-of-the-art review, IBM J. Res. Dev. 28 (2) (1984) 124–134.
[36] N. El-Sayed, I.A. Stefanovici, G. Amvrosiadis, A.A. Hwang, B. Schroeder, Temperature management in data centers: why some (might) like it hot, ACM SIGMETRICS Perform. Eval. Rev. 40 (1) (2012) 163–174.
[37] X. Jian, N. DeBardeleben, S. Blanchard, V. Sridharan, R. Kumar, Analyzing reliability of memory sub-systems with double-chipkill detect/correct, in: Pacific Rim International Symposium on Dependable Computing (PRDC), 2013, pp. 88–97.
[38] P.J. Nair, V. Sridharan, M.K. Qureshi, XED: exposing on-die error detection information for strong memory reliability, in: International Symposium on Computer Architecture (ISCA), 2016, pp. 341–353.
[39] P.J. Nair, D.-H. Kim, M.K. Qureshi, ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates, in: ISCA, 2013, pp. 72–83.
[40] Y.H. Son, S. Lee, O. Seongil, S. Kwon, N.S. Kim, J.H. Ahn, CiDRA: A cache-inspired DRAM resilience architecture, in: International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 502–513.
[41] P. Nikolaou, Y. Sazeides, L. Ndreu, M. Kleanthous, Modeling the implications of DRAM failures and protection techniques on datacenter TCO, in: International Symposium on Microarchitecture, 2015, pp. 572–584.
[42] U. Kang, H.-s. Yu, C. Park, H. Zheng, J. Halbert, K. Bains, S. Jang, J.S. Choi, Co-architecting controllers and dram to enhance dram process scaling, in: The memory forum, 2014, pp. 1–4.
[43] H. Jeon, G.H. Loh, M. Annavaram, Efficient RAS support for die-stacked DRAM, in: Test Conference (ITC), 2014 IEEE International, IEEE, 2014, pp. 1–10.
[44] Tezzaron, Our Technology 101, (http://www.tezzaron.com/about-us/our-technology-101/).
[45] H.-M. Chen, C.-J. Wu, T. Mudge, C. Chakrabarti, Ratt-ecc: Rate adaptive two-tiered error correction codes for reliable 3d die-stacked memory, ACM Trans. Archit. Code Optimization (TACO) 13 (3) (2016) 24.
[46] X. Jian, H. Duwe, J. Sartori, V. Sridharan, R. Kumar, Low-power, low-storage-overhead chipkill correct via multi-line error correction, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013, p. 24.
[47] C. Chou, P. Nair, M.K. Qureshi, Reducing refresh power in mobile devices with morphable ECC, in: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2015, pp. 355–366.
[48] C.-H. Lin, D.-Y. Shen, Y.-J. Chen, C.-L. Yang, M. Wang, Secret: Selective error correction for refresh energy reduction in drams, in: Computer Design (ICCD), 2012 IEEE 30th International Conference on, 2012, pp. 67–74.
[49] M.K. Qureshi, D.-H. Kim, S. Khan, P.J. Nair, O. Mutlu, Avatar: A variable-retention-time (vrt) aware refresh for dram systems, in: Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, IEEE, 2015, pp. 427–437.
[50] P.J. Nair, D.A. Roberts, M.K. Qureshi, Citadel: Efficiently protecting stacked memory from large granularity failures, in: International Symposium on Microarchitecture, IEEE Computer Society, 2014, pp. 51–62.
[51] J. Kim, M. Sullivan, M. Erez, Bamboo ECC: Strong, safe, and flexible codes for reliable computer memory, in: International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 101–112.
[52] B. Giridhar, M. Cieslak, D. Duggal, R. Dreslinski, H.M. Chen, R. Patti, B. Hold, C. Chakrabarti, T. Mudge, D. Blaauw, Exploring dram organizations for energy-efficient and resilient exascale memories, in: International Conference on High Performance Computing, Networking, Storage and Analysis, 2013, p. 23.
[53] L. Chen, Z. Zhang, Memguard: A low cost and energy efficient design to support and enhance memory system reliability, in: ISCA, 2014, pp. 49–60.
[54] S. Li, D.H. Yoon, K. Chen, J. Zhao, J.H. Ahn, J.B. Brockman, Y. Xie, N.P. Jouppi, MAGE: adaptive granularity and ECC for resilient and power efficient memory systems, in: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, 2012, pp. 1–11.
[55] D.W. Kim, M. Erez, Balancing reliability, cost, and performance tradeoffs with freefault, in: High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on, IEEE, 2015, pp. 439–450.
[56] X. Jian, R. Kumar, Adaptive reliability chipkill correct (ARCC), in: High Performance

Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on, IEEE, 2013, pp. 270–281.
[57] D.W. Kim, M. Erez, Relaxfault memory repair, ISCA (2016) 645–657.
[58] D.W. Kim, M. Erez, Stay Alive, Don't Give Up: DUE and SDC Reduction with Memory Repair, in: International Conference on Cluster Computing (CLUSTER), 2015, pp. 588–594.
[59] A. Malek, E. Vasilakis, V. Papaefstathiou, P. Trancoso, I. Sourdis, Odd-ECC: on-demand DRAM error correcting codes, in: International Symposium on Memory Systems, 2017, pp. 96–111.
[60] Y. Luo, S. Ghose, T. Li, S. Govindan, B. Sharma, B. Kelly, A. Boroumand, O. Mutlu, Using ECC DRAM to adaptively increase memory capacity, arXiv:1706.08870 (2017).
[61] S.-L. Gong, M. Rhu, J. Kim, J. Chung, M. Erez, Clean-ECC: High reliability ECC for adaptive granularity memory system, in: International Symposium on Microarchitecture, 2015, pp. 611–622.
[62] A.N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, N.P. Jouppi, Rethinking DRAM design and organization for energy-constrained multi-cores, in: ISCA, 2010, pp. 175–186.
[63] H.-M. Chen, A. Arunkumar, C.-J. Wu, T. Mudge, C. Chakrabarti, E-ECC: Low power erasure and error correction schemes for increasing reliability of commodity DRAM systems, in: International Symposium on Memory Systems, ACM, 2015, pp. 60–70.
[64] X. Jian, J. Sartori, H. Duwe, R. Kumar, High performance, energy efficient chipkill correct memory with multidimensional parity, IEEE Comput. Archit. Lett. 12 (2) (2013) 39–42.
[65] H. Wang, K. Zhao, M. Lv, X. Zhang, H. Sun, T. Zhang, Improving 3D DRAM fault tolerance through weak cell aware error correction, IEEE Trans. Comput. 66 (5) (2017) 820–833.
[66] X. Jian, R. Kumar, ECC Parity: A technique for efficient memory error resilience for multi-channel memory systems, in: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2014, pp. 1035–1046.
[67] D.H. Yoon, M.K. Jeong, M. Sullivan, M. Erez, The dynamic granularity memory system, in: ISCA, 2012, pp. 548–559.
[68] G. Mappouras, A. Vahid, R. Calderbank, D.R. Hower, D.J. Sorin, Jenga: Efficient fault tolerance for stacked dram, in: International Conference on Computer Design (ICCD), IEEE, 2017, pp. 361–368.
[69] J. Sim, G.H. Loh, V. Sridharan, M. O'Connor, Resilient die-stacked DRAM caches, in: International Symposium on Computer Architecture, 2013, pp. 416–427.
[70] L. Chen, Y. Cao, Z. Zhang, E3CC: A memory error protection scheme with novel address mapping for subranked and low-power memories, ACM Trans. Archit. Code Optim. (TACO) 10 (4) (2013) 32.
[71] D.H. Yoon, M.K. Jeong, M. Erez, Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput, in: ISCA, 2011, pp. 295–306.
[72] Y. Cao, L. Chen, Z. Zhang, Memory design for selective error protection, in: International Conference on Computer Design (ICCD), 2015, pp. 133–140.
[73] J. Kim, M. Sullivan, S.-L. Gong, M. Erez, Frugal ECC: Efficient and versatile memory error protection through fine-grained compression, in: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2015, pp. 1–12.
[74] S. Mittal, A survey of architectural techniques for DRAM power management, Int. J. High Perform. Syst. Archit. 4 (2) (2012) 110–119.
[75] A. Seznec, Decoupled sectored caches: conciliating low tag implementation cost, in: ISCA, 1994, pp. 384–393.
[76] H.-M. Chen, S. Jeloka, A. Arunkumar, D. Blaauw, C.-J. Wu, T. Mudge, C. Chakrabarti, Using low cost erasure and error correction schemes to improve reliability of commodity DRAM systems, IEEE Trans. Comput. 65 (12) (2016) 3766–3779.
[77] D.J. Palframan, N.S. Kim, M.H. Lipasti, COP: To compress and protect main memory, in: ISCA, 2015, pp. 682–693.
[78] S. Mittal, J. Vetter, A survey of architectural approaches for data compression in cache and main memory systems, IEEE Trans. Parallel Distributed Systems (TPDS) 27 (5) (2015) 1524–1536.
[79] K. Nguyen, K. Lyu, X. Meng, V. Sridharan, X. Jian, Nonblocking memory refresh, International Symposium on Computer Architecture (ISCA) (2018).
[80] S. Schechter, G.H. Loh, K. Straus, D. Burger, Use ECP, Not ECC, for Hard Failures in Resistive Memories, in: International Symposium on Computer Architecture (ISCA), 2010, pp. 141–152.
[81] B. Jenkins, Spookyhash: a 128-bit noncryptographic hash, 2011, (http://www.burtleburtle.net/bob/hash/spooky.html).
[82] S. Mittal, A survey of techniques for approximate computing, ACM Comput. Surv. 48 (4) (2016) 62:1–62:33.
[83] S. Mittal, A survey of techniques for architecting processor components using domain wall memory, ACM J. Emerg. Technol. Comput. Syst. 13 (2) (2016) 29:1–29:25.
[84] S. Mittal, A survey of reRAM-based architectures for processing-in-memory and neural networks, Mach. Learn. Knowl. Extr. 1 (2018) 5.
[85] M. Stoicescu, J.-C. Fabre, M. Roy, Architecting resilient computing systems: a component-based approach for adaptive fault tolerance, J. Syst. Archit. 73 (2017) 6–16.
[86] J. Zhan, Y. Zhang, W. Jiang, J. Yang, L. Li, Y. Li, Energy-aware page replacement and consistency guarantee for hybrid NVM–DRAM memory systems, J. Syst. Archit. 89 (2018) 60–72.
[87] S. Mittal, A.I. Alsalibi, A survey of techniques for improving security of non-volatile memories, J. Hardware Syst. Secur. (2018) 1–22.

Sparsh Mittal received the B.Tech. degree in electronics and communications engineering from IIT, Roorkee, India and the Ph.D. degree in computer engineering from Iowa State University (ISU), USA. He worked as a Post-Doctoral Research Associate at Oak Ridge National Lab (ORNL), USA for 3 years. He is currently working as an assistant professor at IIT Hyderabad, India. He was the graduating topper of his batch in B.Tech and has received fellowship from ISU and performance award from ORNL. Sparsh has published more than 75 papers in top conferences and journals. My research has been covered by several technical news websites, e.g. Phys.org, InsideHPC, Primeur Magazine, StorageSearch, Data-Compression.info, TechEnablement, ScientificComputing, SemiEngineering, ReRAM forum and HPCWire. His research interests include accelerators for neural networks, architectures for machine learning, non-volatile memory, and GPU architectures. His webpage is http://www.iith.ac.in/~sparsh/.

Maruthi Inukonda is a 1st year Ph.D student in department of computer science engineering, IIT Hyderabad in the CANDLE Lab. Maruthi holds M.Tech in CSE from IIT Roorkee. His interests include computer architecture, Linux Internals, storage software, IAAS cloud computing, software engineering, software architecture. He has about 13 years systems programming experience in storage software companies (Veritas, NetApp, EMC). He has one journal publication and a patent in storage software on Linux.