RDIS: Tolerating Many Stuck-At Faults in Resistive Memory

Rakan Maddah, *Member, IEEE,* Rami Melhem, *Fellow, IEEE,* and Sangyeun Cho, *Senior Member, IEEE*

Abstract—With their potential for high scalability and density, resistive memories are foreseen as a promising technology that overcomes the physical limitations confronted by charge-based DRAM and flash memory. Yet, a main burden towards the successful adoption and commercialization of resistive memories is their low cell reliability caused by process variation and limited write endurance. Typically, faulty and worn-out cells are permanently stuck at either '0' or '1'. To overcome the challenge, a robust error correction scheme that can recover from many hard faults is required.
In this paper, we propose and evaluate *RDIS*, a novel scheme to efficiently tolerate memory stuck-at faults. RDIS allows for the correct retrieval of data by recursively determining and efficiently keeping track of the positions of the bits that are stuck at a value different from the ones that are written, and then, at read time, by inverting the values read from those positions. RDIS is characterized by a very low probability of failure that increases slowly with the relative increase in the number of faults. Moreover, RDIS tolerates many more faults than the best existing scheme—by up to 95% on average at the same overhead level.

Index Terms—Hard Faults; Phase Change Memory; Fault Tolerance; Reliability;

1 Introduction

Phase Change Memory (PCM) is receiving due attention as an alternative memory technology to DRAM and flash memory. The latter are hindered by physical limitations putting their scalablity into jeopardy [1], [2], [3]. Instead of representing information as the presence or absence of electrical charges, PCM encodes bits in different physical states of chalcogenide alloy that consists of Ge, Sb and Te. Data is stored in PCM devices in the form of either a low resistance crystalline state (SET) or a high resistance amorphous state (RESET). Early evaluations (e.g., [4], [5], [6]) demonstrate that PCM can compete favorably with DRAM (main memory) in terms of performance and can beat DRAM in terms of power consumption.

Unfortunately, each PCM cell can endure only a limited number of SET/RESET cycles [7], [8], [9]. There are two major factors in this matter: imperfect process control with a very deep sub-micron technology and repeated writes to a cell (i.e., write cycling). Prior architecture and systems research focus on the latter because manufacturers will ship chips with a minimum guaranteed write endurance. As a cell gets written, the heating and cooling process to program the cell at a desired resistance level leads to frequent expansions and contractions of the material. Consequently, the heating element detaches from the chalcogenide material after sustaining 10^6 to 10^8 [10], [11], [12], [13] writes on average, which results in a stuck-at hard fault at either '0' or '1 that can be subsequently read but not reprogrammed [8], [14].

In order to make PCM a viable memory technology for high volume manufacturing, mitigating the write endurance problem is essential. It is believed that both aggressive wear leveling and proactive error masking techniques are necessary. Wear leveling spreads writes to the entire memory capacity to evenly wear memory cells (e.g., through periodic, pseudo-randomization of write addresses) [5], [6], [15], [16]. Techniques to suppress unnecessary bit-level writes were proposed [4], [6], [17], [18]. However, due to process variation, memory cells are expected to wear out at different rates, which compromises the chip's lifetime. Therefore, error masking techniques are required to overcome cell failures.

Error correction code (ECC) such as SEC-DED (single error correction, double error detection) has been successfully used to protect main memory. However, a traditional hamming code based ECC is designed for a general fault model and its overhead is unnecessarily large for the stuck-at fault model. This is especially true when the probability of having multiple bit errors is high, as is the case with resistive memories. For example, imagine that many cells in a memory block have reached their write endurance limit simultaneously. To cope with many faults, we must employ a correspondingly stronger ECC, which would incur excessively large space and computation overheads. Moreover, the change in the data pattern written within a block requires a recomputation of the parity information for the ECC. In fact, a change for a single bit in the data could induce a change of several parity bits. Accordingly, the parity bits are vulnerable to wear out earlier than the data bits. Subsequently, recently proposed error masking techniques for resistive memories [7], [8], [9] combine clever microarchitectural and coding ideas to reduce overhead.

In this paper, we propose *RDIS* (recursively defined invertible set), a novel low-overhead error correction scheme to recover from hard errors.[1] RDIS allows for the correct retrieval of data in the presence of stuck-at faults by keeping track of the bits that are stuck at a value different from the ones that are written, and then, at read time, by inverting the values read for those bits. For a write operation, each cell in a data block is either: "non-faulty" (NF), stuck at the opposite of the value being written ("stuck-at-wrong" or SA-W), or stuck at the same value written ("stuck-at-right" or SA-R).

- *R. Maddah and R. Melhem are with the Department of Computer Science, University of Pittsburgh, 210 S. Bouquet, Pittsburgh, PA 15260.*
E-mail: {rmaddah,melhem@cs.pitt.edu}
- *S. Cho is with the Memory Division of Samsung Electronics Co. while on leave of absence from the University of Pittsburgh.*
E-mail: {cho@cs.pitt.edu}

1. The principles of RDIS are not limited to resistive memories. RDIS is particularly relevant for resistive memories because it can correct many errors with high probability.

For example, trying to write '0' in a cell stuck at '1' makes the cell SA-W. The underlying idea of RDIS is to identify and encode a subset S—out of all cells forming a data block to be updated—containing all the SA-W cells. Later, the members of S are read inverted, which retrieves the data as it was intended to be written originally. RDIS initiates the computation of S after detecting write failure through applying a read-after-write verification operation.

Although it can only guarantee the recovery from three faults, RDIS has a desirable property of effectively recovering from many more faults beyond what it guarantees. Intrinsically, RDIS enjoys a low probability of failure that increases at a very slow rate with the relative increase in the number of fault occurrences. By comparison, current state-of-the-art schemes either cannot recover from a single fault beyond a guaranteed number of faults (e.g., ECC [19] and ECP [7]) or can recover additional faults but with a low probability (e.g., SAFER [8]). Our evaluation shows that RDIS can tolerate 95% more faults on average than SAFER when the protected block size is 1 KB. Given its ability to recover many faults with high probability, RDIS is a very good fit for resistive memories that will experience a growing number of faults over the course of use.

We formally prove the fault tolerance properties of RDIS by stating all necessary and sufficient conditions that make RDIS holds. It is worth mentioning that RDIS error correction capabilities are not limited to main memory. RDIS is capable of tolerating faults within block sizes ranging from cache line size to secondary storage block sector size, while incurring a low overhead. Accordingly, we present a study of RDIS error correction capability at different block sizes. Furthermore, we present techniques that allow RDIS to recover from fault patterns that it cannot mask. Our evaluation shows that those techniques significantly increase the error correction capability of RDIS.

The remainder of this paper is organized as follows. Section 2 first summarizes the related work. Section 3, Section 4 and Section 5 then give the details of the proposed RDIS scheme by formally describing the concepts and the coverage of the scheme as well as discuss hardware implementation implications. Section 6 presents a technique to reduce the space overhead for RDIS. Section 7 discusses the application of RDIS in the context of multilevel cells. Section 8 presents techniques that allow RDIS to squeeze more lifetime for memory devices. Experimental evaluation of RDIS is presented in Section 9, and finally, Section 10 concludes the paper.

2 Prior Related Work

The exploration of ECC can be traced many years back [19]. Among many ECC schemes, SEC-DED is widely used to protect DRAM in main memory. Since DRAM errors are typically transient and occur infrequently, SEC-DED is adequate in most situations. On the other hand, resistive memories have different failure mechanisms and are subject to multiple bit faults that occur gradually with the lifetime of a chip. Consequently, it is necessary to deploy a multi-bit error correction scheme. Hamming code based BCH code [20] is one such scheme. Yet, codes based on BCH are complex and expensive to implement [21], [22]. As a matter of fact, the complexity increases linearly with the number of faults to be tolerated [21].

There are three recent proposals that target specifically masking errors in resistive memories with higher auxiliary storage efficiency than traditional ECC techniques. First, ECP (Error Correcting Pointer) [7] provides a limited number of programmable "correction entries". A correction entry holds a pointer (address) to a faulty cell within the protected block and a "patch" cell that replaces the faulty one. When a faulty cell is detected, a new correction entry is allocated to cover the cell. A memory block is de-commissioned when the number of faulty cells exceeds that of the correction entries. In essence, ECP provides cell-level spares to each block.

SAFER (Stuck-at-Fault Error Recovery) [8] dynamically partitions a protected data block into a number of groups so that each group contains at most one faulty cell. When the value of the faulty cell is different from the intended value to be written, all cells in the the group are written and read inverted. If the data block is to be partitioned into n groups, then SAFER allows $\log_2 n$ "repartitions". Repartitioning is done whenever a new fault is detected. Therefore, SAFER guarantees the recovery from $\log_2 n+1$ faults. Any additional fault is tolerated only if it occurs in a fault-free group. Otherwise, the block has to be retired. SAFER was shown to provide stronger error correction than ECC or ECP at the same overhead level.

Free-p (Fine-grained Remapping with ECC and Embedded-Pointers) [9] combines error correction and redundancy, and as such, has two protection layers. First, it uses an ECC to mask faults within a data block. Second, when a block becomes defective, Free-p embeds a pointer within the defective block so that a redundant, non-faulty block can be quickly identified without having to access a separate remapping table. Free-p employs ECC to correct up to four hard errors in a data block of cache line size and relies on the OS to perform block remapping. We note that the block remapping idea of Free-p is orthogonal to RDIS. Hence, RDIS could be used to replace ECC in Free-p.

PAYG (Pay-As-You-Go) [23] is a resilient architecture proposed to decrease the storage overhead of auxiliary bits information required by error correction schemes (e.g. ECP and SAFER) targeting the recovery from stuck-at faults. Essentially, PAYG moves from a uniform allocation of auxiliary bits across the protected memory blocks to a dynamic on demand allocation. PAYG exploits the variability in lifetime that the memory blocks exhibit and assigns additional auxiliary bits to weaker blocks. We note that RDIS is compatible with PAYG. The auxiliary bits of RDIS could be allocated in a similar way to what is proposed in PAYG.

3 RDIS

This section describe RDIS intuitively using Set Theory. We begin with the idea of invertible sets and how to specify an invertible set given a set of faulty memory cells in a block. We then focus on an algorithm that computes the necessary auxiliary information to correctly store and retrieve user information. Finally, we discuss a hardware embodiment of RDIS before we close this section.

3.1 Basic idea

RDIS applies to a block of memory/storage cells. Let's assume that the block has N cells, $c(0)$, …, $c(N-1)$ that store binary information $b(0)$, …, $b(N-1)$. Each cell $c(i)$ is either non-faulty (NF), stuck at '0' (SA-0), or stuck at '1' (SA-1). Furthermore, RDIS uses a different classification of the faulty cells, depending on the value that is to be written in those cells. Specifically, when bit $b(i)$ is to be stored in a faulty cell $c(i)$, then $c(i)$ is stuck at the right value (SA-R) if it is SA-0

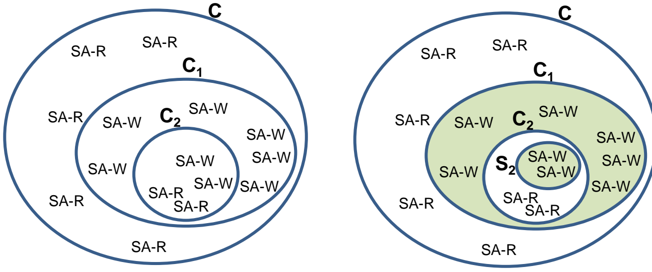


Fig. 1: The invertible set $S = (C_1 - C_2) \cup S_2$.

and $b(i) = 0$ or it is SA-1 and $b(i) = 1$. Similarly, $c(i)$ is stuck at the wrong value (SA-W) if it is SA-0 and $b(i) = 1$ or it is SA-1 and $b(i) = 0$. Using this classification, each cell $c(i)$ can be in one of three classes: NF, SA-R, or SA-W.

H-bit auxiliary information is used to allow the correct retrieval of the N stored bits. The value of H will be specified later. For clarity of discussion, we assume that the auxiliary information is maintained in a separate fault-free storage. Alternatively, the auxiliary information can be stored in the same faulty medium as the data but adequately protected by some other technique (see Section 9.3 for further discussions).

Denoting the set of memory cells $c(0)$, ..., $c(N-1)$ by C, the main idea of RDIS is to use the auxiliary H bits to identify a subset $S \subset C$ such that every SA-W cell is in S and every SA-R cell is in $C - S$. In other words, S contains all the SA-W cells of C and none of its SA-R cells. We call S an "invertible" subset of C. When the N bits of information are stored, any cell $c(i)$ in $C - S$ will store $b(i)$ intact, while any cell in S will store the complement of $b(i)$. Subsequently, when the information is read, the content of any cell in S is complemented, thus allowing the correct retrieval of all N bits.

A simple way of expressing S is to keep a list of pointers to the SA-W cells. This requires $\log_2 N$ bits of auxiliary information for each cell and hence, to tolerate a maximum of F faults, $H = F \times \log_2 N$ bits of auxiliary information is needed. RDIS introduces a different, yet systematic method for constructing and representing S by allowing it to include NF (not faulty) cells in addition to SA-W cells. Clearly, if a cell $c(i)$ is not faulty, then it is possible to store (and correctly retrieve) the complement of $b(i)$ in $c(i)$. Conceptually, the set S is constructed by computing the two subsets $C_2 \subset C_1 \subset C$ such that:

- All the SA-W cells that are in C, are included in C_1.
- All the SA-R cells that are in C_1, are included in C_2.

Fig. 1 illustrates the idea of the construction of C_1 and C_2. Note that any of C, C_1, and C_2 can contain NF cells as well. However, by definition, $C_1 - C_2$ does not contain any SA-R cells. Clearly, if C_1 does not contain any SA-R cells, then the construction of C_2 is not needed since we can set $S = C_1$.

Next, we consider two cases. First, if C_2 does not contain any SA-W cells, then the invertible set S that we are looking for is $S = C_1 - C_2$ since we are sure that $C_1 - C_2$ contains all the SA-W cells of C and none of its SA-R cells. The second case occurs if C_2 contains some SA-W cells. In this case, we recursively apply the same process to find an invertible set S_2 of C_2 which includes all its SA-W cells and none of its SA-R cells. Therefore, $S = C_1 - (C_2 - S_2) = (C_1 - C_2) \cup S_2$. Fig. 1 shows the invertible set S of C as a shaded area.

3.2 Specifying an invertible subset

One way to identify S is to arrange the N bits/cells into a logical two-dimensional array of n rows and m columns,[2] and accordingly, re-label the information bits as $b(i,j)$ and the storage cells as $c(i,j)$, where $i = 0, \ldots, n-1$ and $j = 0, \ldots, m-1$. In this section, we will use the example of the 8×8 array shown in Fig. 2(a) to illustrate the process of specifying the invertible set. As depicted, C contains 7 SA-W and 7 SA-R faults.

RDIS maintains $n + m$ auxiliary binary flags $\mathrm{VX}_1(i)$, $i = 0, \ldots, n-1$ and $\mathrm{VY}_1(j)$, $j = 0, \ldots, m-1$. These flags are set such that:

- $\mathrm{VX}_1(i) = 1$ if row i of C contains at least one SA-W cell (otherwise $\mathrm{VX}_1(i) = 0$); and
- $\mathrm{VY}_1(j) = 1$ if column j of C contains at least one SA-W cell (otherwise $\mathrm{VY}_1(j) = 0$).

Let n_1 be the number of rows in the $n \times m$ array C that have $\mathrm{VX}_1 = 1$ and let m_1 be the number of columns of C that have $\mathrm{VY}_1 = 1$. Moreover, define C_1 as the subset of cells $\{c(i,j) | (\mathrm{VX}_1(i) = 1) \text{ and } (\mathrm{VY}_1(j) = 1)\}$. In other words, C_1 is the $n_1 \times m_1$ subarray of C that contains: (1) SA-W cells and (2) cells that lie at the intersection of a row that contains a SA-W cell and a column that contains a SA-W cell (these can be either NF or SA-R). In our example, the values of VX_1 and VY_1 are shown in Fig. 2(a). The SA-W cells of C are confined to rows 2, 4, 5, 7 and columns 1, 3, 4, 6, and hence, these rows and columns form the subarray C_1 shown in Fig. 2(b).

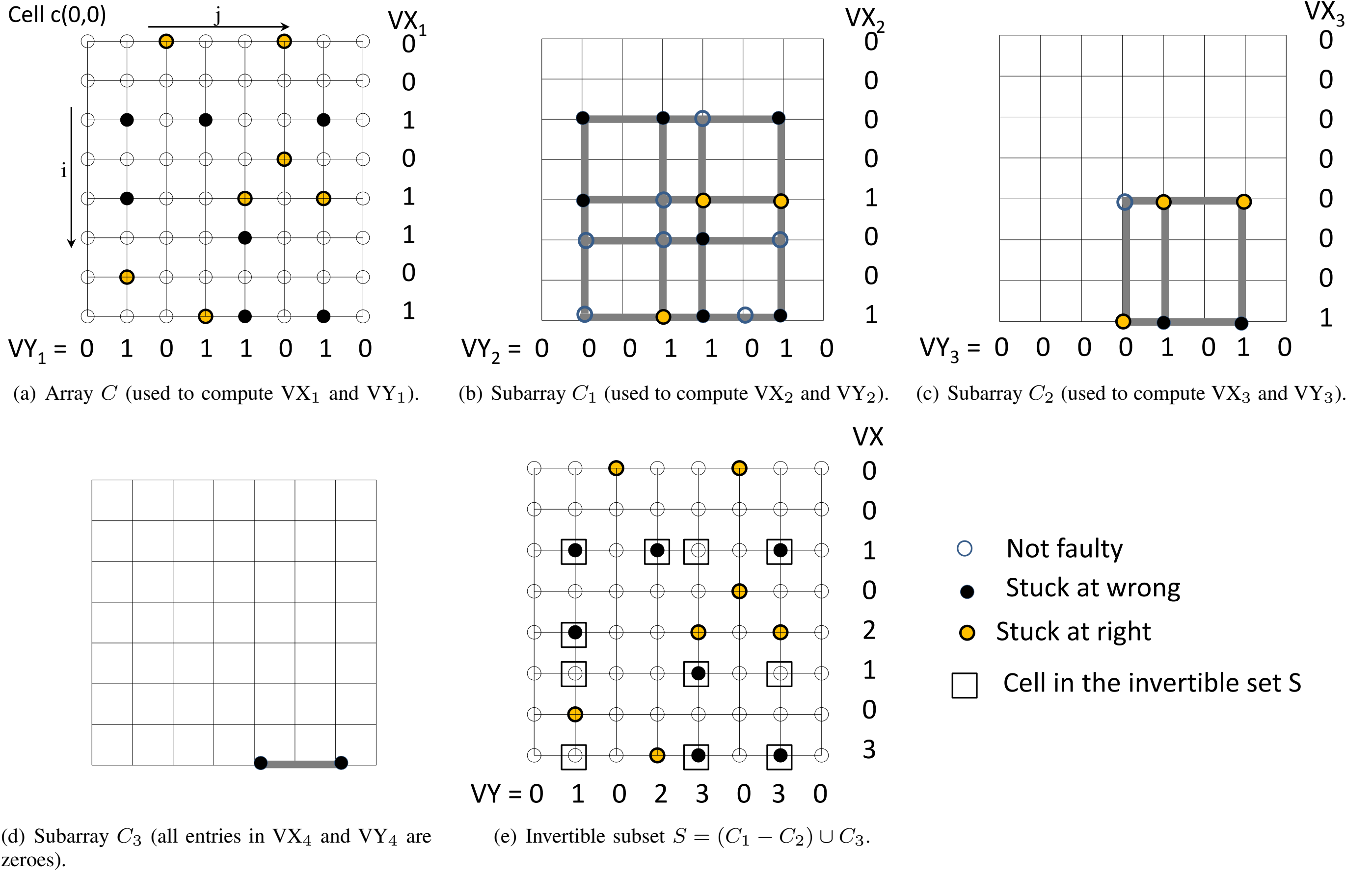
Since C_1 is defined to include all the SA-W cells of C, any cell that is in $C - C_1$ is either NF or SA-R, and thus can hold the correct value of the corresponding information bit. However, the cells that are in C_1 may be NF, SA-W, or SA-R. If C_1 does not contain any SA-R cell (i.e., C_1 contains only NF or SA-W cells), then $S = C_1$. If, however, C_1 contains some SA-R cells (as is the case in Fig. 2(b)), then, we need to find a subset, S_1 of C_1, which includes all its SA-R cells and none of its SA-W cells. This will allow us to specify an invertible subset of C as $S = C_1 - S_1$. To obtain S_1, we apply the same procedure used to extract C_1 from C, but after reversing the roles of SA-R and SA-W. Specifically, we define the binary flags

- $\mathrm{VX}_2(i) = 1$ if row i of C_1 contains at least one SA-R cell (otherwise $\mathrm{VX}_2(i) = 0$); and
- $\mathrm{VY}_2(j) = 1$ if column j of C_1 contains at least one SA-R cell (otherwise $\mathrm{VY}_2(j) = 0$).

Let n_2 be the number of rows of C_1 that have $\mathrm{VX}_2 = 1$ and let m_2 be the number of columns of C_1 that have $\mathrm{VY}_2 = 1$. Moreover, define C_2 as the subset of cells $\{c(i,j) | (\mathrm{VX}_2(i) = 1) \text{ and } (\mathrm{VY}_2(j) = 1)\}$. In other words, C_2 is the $n_2 \times m_2$ subarray of C_1 that contains: (1) SA-R cells and (2) cells that lie at the intersection of a row that contains a SA-R cell and a column that contains a SA-R cell. In the example of Fig. 2, we form subarray C_2 to include all the SA-R cells that are in C_1. In Fig. 2(c), C_2 is composed of rows 4, 7 and columns 3, 4, 6. By construction, any cell that is in $C_1 - C_2$ is either NF or SA-W. Moreover, if C_2 does not contain any SA-W cell, then $S_1 = C_2$ and we can form the invertible set $S = C_1 - C_2$.

Unfortunately, we cannot guarantee that C_2 does not contain any SA-W cell. Fortunately, however, if $C_2 \neq C_1$ (i.e., C_2 is a proper subset of C_1), then we can apply the same procedure used to extract C_1 from C to compute the subset S_2 of C_2

2. Introducing more than two dimensions is certainly possible, but is beyond the scope of this paper.



(a) Array C (used to compute VX_1 and VY_1). (b) Subarray C_1 (used to compute VX_2 and VY_2). (c) Subarray C_2 (used to compute VX_3 and VY_3).

(d) Subarray C_3 (all entries in VX_4 and VY_4 are zeroes). (e) Invertible subset $S = (C_1 - C_2) \cup C_3$.

Fig. 2: An example for constructing the invertible set.

that contains all its SA-W cells and then set $S = (C_1 - C_2) \cup S_2$. The iterative process can continue to compute consecutive subarrays $C_3, \ldots, C_k$. After k iterations, we can have one of three cases:

1. k is odd and C_k contains only SA-W cells. In this case, the invertible set S is defined as

$$S = (C_1 - C_2) \cup (C_3 - C_4) \cup \ldots \cup (C_{k-2} - C_{k-1}) \cup C_k.$$

2. k is even and C_k contains only SA-R cells. In this case the invertible set S is defined as

$$S = (C_1 - C_2) \cup (C_3 - C_4) \cup \ldots \cup (C_{k-1} - C_k).$$

3. The progress stalls because $C_k = C_{k-1}$, in which case the set of faults cannot be masked.

Back to the example of Fig. 2, the array C_2 shown in Fig. 2(c) includes all the SA-R cells that are in C_1 but also contains two SA-W cells. Hence, we form subarray C_3 to include all the SA-W cells that are in C_2 (see Fig. 2(d)). The process terminates with $k = 3$ because C_3 does not include any SA-R cells, and thus, $S = (C_1 - C_2) \cup C_3$ contains all the SA-W cells that are in C and none of its SA-R cells (see Fig. 2(e)).

3.3 Overhead of auxiliary information

The subarrays C_1, C_2, ... are completely specified by the binary flags $\mathrm{VX}_1(i)$, $\mathrm{VX}_2(i)$, ..., $i = 0, \ldots, n-1$ and $\mathrm{VY}_1(j)$, $\mathrm{VY}_2(j)$, ..., $j = 0, \ldots, m-1$. In other words, these flags form the auxiliary information that has to be maintained to retrieve the correct values stored in the N cells. Note that if $\mathrm{VX}_u(i) = 0$ for some u, then $\mathrm{VX}_k(i) = 0$ for any $k > u$. Similarly, if $\mathrm{VY}_w(i) = 0$ for some w, then $\mathrm{VY}_k(i) = 0$ for any $k > w$. Hence, the flags can be compressed into two sets of counters (see Fig. 2(e)): $\mathrm{VX}(i) = \sum_{k=1}^{u} \mathrm{VX}_k(i)$ for $i = 0, \ldots, n-1$; and $\mathrm{VY}(j) = \sum_{k=1}^{w} \mathrm{VY}_k(j)$ for $j = 0, \ldots, m-1$.

The auxiliary information needed to reconstruct S, thus, consists of the $(n+m)$ counters $\mathrm{VX}(i)$ and $\mathrm{VY}(j)$. If each of these counters can count up to K, then the number of bits, H, needed to keep the auxiliary information is $H = (n+m) \times \lceil \log_2(K+1) \rceil$. Note that by limiting the maximum value of each counter to K, we assume that the recursive construction of S will terminate in K steps. If that is not the case, then the process will fail and the given faults cannot be tolerated.

3.4 Storing and retrieving information

In order to store and retrieve user data, VX and VY must be computed first. Let us present an algorithm to do that assuming that the locations and nature of faults are known. This information can be kept in a separate storage (e.g., SRAM cache) or discovered on line by a write-read-check process (as described later). Given the fault information and the data that is to be written, we can associate with each cell, $c(i,j)$, a state that is represented by two bits $\phi(i,j)$ and $\sigma(i,j)$ as follows:

- $\phi(i,j) = 1$ and $\sigma(i,j) = 0 \rightarrow$ cell $c(i,j)$ is SA-R.
- $\phi(i,j) = 1$ and $\sigma(i,j) = 1 \rightarrow$ cell $c(i,j)$ is SA-W.
- $\phi(i,j) = 0$ and $\sigma(i,j) = 0 \rightarrow$ cell $c(i,j)$ is NF or the fault was successfully handled.

To compute the values of the counters VX_i for $i = 0, \ldots, n-1$ and VY_j $j = 0, \ldots, m-1$;, Algorithm 1 is

Algorithm 1: Computing VX and VY.

```
1  begin
2    for k ← 1 to K do
3      for i ← 0 to n − 1 do
4        VX_k(i) ← σ(i, 0) + ... + σ(i, m − 1); // Boolean OR
5        if VX_k(i) = 1 then
6          VX(i) ← VX(i) + 1;
7      for j ← 0 to m − 1 do
8        VY_k(j) ← σ(0, j) + ... + σ(n − 1, j); // Boolean OR
9        if VY_k(j) = 1 then
10         VY(j) ← VY(j) + 1;
11     if ∀i, j VX_k(i) = 0 and VY_k(j) = 0 then
12       EXIT; // successful completion
13     /* prepare for next iteration */
14     for i ← 0 to n − 1 do
15       for j ← 0 to m − 1 do
16         if VX_k(i) = 0 or VY_k(j) = 0 then
17           set φ(i, j) ← 0; σ(i, j) ← 0;
18         else if φ(i, j) = 1 then
19           set σ(i, j) ← ¬σ(i, j); // Bit complement
20   if ∃i, j VX_k(i) > 0 or VY_k(j) > 0 then
21     FAIL; // Given faults can't be masked
```

applied. In each iteration, k, of the algorithm (line 2), the subarray which contains SA-W cells is formed (by computing the flags VX_k and VY_k - lines 3 to 10). Then, the state of every cell that is not in this subarray is set to (ϕ = 0 and σ = 0) since it is either NF or is SA-R (lines 16 and 17). In preparation for the next iteration, the algorithm then changes the states of every faulty cell in the identified subarray such that SA-W cells become SA-R and SA-R cells become SA-W (lines 18 and 19). The algorithm assumes that the counters $\mathrm{VX}(i)$ and $\mathrm{VY}(j)$are initially set to zero.

The way the counters $\mathrm{VX}(i)$ and $\mathrm{VY}(j)$ are computed implies that if cell $c(i, j)$ is in C_k and not in C_{k+1} then at least one of the two counters $\mathrm{VX}(i)$ or $VY(j)$ is equal to k while the other one is equal to $k + 1$. Given this observation, Algorithm 2 can be used to store the data bits.

Algorithm 2: Storing data bits.

```
1  begin
2    for i ← 0 to n − 1 do
3      for j ← 0 to m − 1 do
4        if min(VX(i), VY(j)) is even then
5          Store b(i, j) in c(i, j);
6        else
7          Store ¬b(i, j) in c(i, j);
```

Similarly, when retrieving the data, the bit read from cell $c(i, j)$ is complemented if the minimum of $\mathrm{VX}(i)$ and $\mathrm{VY}(j)$ is an odd number. Building a hardware circuit to perform this operation is straightforward, especially when the maximum value of VX and VY is small.

4 Putting it All Together

In this section, we sum up the overall flow of execution that RDIS follows to detect and mask errors. After a write operation is executed, a read operation is performed to verify the correctness of the written data. In case the read verification step did not detect any error, then the write request completed successfully and no further action is required. RDIS is different in this regard from ECC where the auxiliary information is always computed irrespective of the presence of errors. As a matter of fact, RDIS is designed specifically to recover from stuck-at faults where errors are permanent once manifested after a write operation. On the other hand, ECC is designed for a general fault model where latent errors are possible. Hence, RDIS exploits the characteristics of the stuck-at fault model and saves the overhead of computing the auxiliary information when no errors are manifested. While the read verfication operation is not exclusive to RDIS and has been used by many schemes that dealt with the PCM endurance problem such as in [7] and [8], it could affect the memory bandwidth. Nevertheless, PCM reads are fast and the read verification operation is off the critical path and should not be affecting the bandwidth severely. Moreover, dealing with errors at encoding time makes data retrieval faster and simpler than dealing with error at decoding time.

In the other case where the read verification operation discovers errors, RDIS initiates the computation of the auxiliary information. As noted previously, the read verification step reveals only the SA-W cells. However, the encoding process of RDIS requires the identification of SA-R cells that can be identified by testing storage cells on the intersection of a row and column both containing a SA-W cell. Specifically, to test a cell $c(i, j)$, we first read the value, v, stored in that cell, write the complement of v into the cell and read it again. If the value read is not the complement of v, then the cell is SA-R. Otherwise, the cell is NF. One way of avoiding the overhead of determining the fault information after each write operation is to keep a cache to store the faults positions along with their stuck-at values. Such a cache was proposed in [8], where it was shown that a 128K-entry cache is enough to capture most of the fault information in an 8 Gbit memory. The same cache design can be used in RDIS.

Clearly, RDIS could require an extra write operation to reveal the fault information. An extra write could have a detrimental effect on endurance as it exacerbates the rate at which non-faulty cells wear-out. However, a non-faulty cell undergoes an extra write only if it lies at the intersection of a row and a column both containing SA-W cells. Thus, non-faulty cells do not get written twice on every write request due to the data-dependent nature of errors. We quantify the effect of of extra writes on the lifetime of a memory chip in Section 9.5.

Once the fault information is determined, the computation of the invertible set is initiated (refer to Algorithm 1 and Algorithm 2).This computation involves setting the auxiliary counters with the appropriate values in order the retrieve the data correctly. Clearly, the auxiliary counters are subject to wear-out if stored in the PCM medium. To protect the auxiliary counters, two approaches could be followed. The first would be to store the counters in a more reliable medium such as DRAM. The second would be to dedicate an error correction scheme to recover from errors within the auxiliary counters. As a matter of fact, the auxiliary counters are written at a lower rate than the actual data cells. Specifically, none of the auxiliary counters starts to be written before the first stuck-at fault appears in the protected block. In addition, writing the counters depends on whether their associated rows or columns contain a stuck-at faults that happen to be SA-W.

Therefore, protecting the auxiliary counters with a low order error correction scheme should be enough as their initial endurance should sustain their infrequent writes. In Section 9, we show that RDIS is capable of tolerating a significantly large number of stuck-at faults and we consider the case of protecting the auxiliary counters with a dedicated error correction scheme.

5 Coverage of RDIS

The previous section described the basic idea of RDIS as well as the necessary algorithms.This section will delve further into the properties of RDIS by studying specific conditions under which RDIS fails to cover a given set of faults. There are two such conditions: (1) the progress stops because for some k, $C_k = C_{k-1}$; and (2) the capacities of the counters VX and VY are exceeded before the recursion terminates. Each of these two situations is caused by specific fault patterns as described next.

5.1 Coverage failure caused by a loop of faulty cells

In this section, let us first consider the case where the progress of the construction of the invertible set stops because $C_k = C_{k-1}$, for some k. We start with some preliminary definitions.

Definition. *A faulty cell, $c(i, j)$ in C_k is row and column connected (RC-connected) if row i in C_k contains at least one other faulty cell, $c(i, j')$, $j \neq j'$ and column j in C_k contains at least one other faulty cell $c(i', j)$, $i \neq i'$.*

For example, cells $c(7, 3)$ in the array of Fig. 2(a) is RC-connected while cell $c(0, 2)$ is not RC-connected.

Definition. *A loop of faulty cells (or "loop of faults") is a sequence of $2q$ faults ($q > 1$) where every two consecutive faults in the sequence are, alternatively, in the same row or in the same column. More specifically, a loop of faulty cells is of the form $c(i_1, j_1)$, $c(i_2, j_1)$, $c(i_2, j_2)$, $c(i_3, j_2)$, ..., $c(i_q, j_q)$, $c(i_1, j_q)$.*

Definition. *A loop of faults $c(i_1, j_1)$, $c(i_2, j_1)$, $c(i_2, j_2)$, $c(i_3, j_2)$, ..., $c(i_q, j_q)$, $c(i_1, j_q)$ is alternatively-stuck (or "A-stuck") if the faults in the loop alternate between SA-R and SA-W. That is, faulty cells $c(i_1, j_1)$, $c(i_2, j_2)$, ..., $c(i_q, j_q)$, are stuck at a value, while faulty cells $c(i_2, j_1)$, $c(i_3, j_2)$, ..., $c(i_1, j_q)$, are stuck at the opposite value.*

For example, the loop in Fig. 3(a) includes the sequence of faulty cells $c(2, 6)$, $c(4, 6)$, $c(4, 4)$, $c(6, 4)$, $c(6, 0)$, $c(3, 0)$, $c(3, 1)$, $c(2, 1)$. Moreover, this loop is A-stuck since cells $c(2, 6)$, $c(4, 4)$, $c(6, 0)$, $c(3, 1)$ are SA-W while cells $c(4, 6)$, $c(6, 4)$, $c(3, 0)$, $c(2, 1)$ are SA-R.

Theorem 1. *The process of constructing the invertible set stops with $C_k = C_{k-1}$ for some k, if the original array of cells, C, contains a loop of faults that is A-stuck.*

Proof. Assume that C contains the A-stuck loop of faults, $c(i_1, j_1)$, $c(i_2, j_1)$, $c(i_2, j_2)$, $c(i_3, j_2)$, ..., $c(i_q, j_q)$, $c(i_1, j_q)$. By definition, each of rows $i_1, \ldots, i_q$ contains two faults, one SA-R and one SA-W, and each of columns $j_1, \ldots, j_q$ contains two faults, one SA-R and one SA-W. Hence, C_1 will include rows i_1, ..., i_q and columns j_1, ..., j_q, meaning that it will include the loop of faults. Similarly, we argue that C_2 and any subsequent subarray will include the same loop of faults. Given that the number of faulty cells in C is finite, then the construction of $C_k \subset C_{k-1}$ will eventually terminate with $C_k = C_{k-1}$ for some k. ■

Theorem 2. *The process of constructing the invertible set terminates with C_K being empty for some K if the original array of cells, C, does not contain a loop of faults.*

Proof. First, we observe that if k is odd (a similar argument applies if k is even) and array, C_k, contains some faulty cells but does not contain a loop of faults, then at least one of the faulty cells in C_k, say $c(i, j)$, is not RC-connected. Second, we observe that if $c(i, j)$ is SA-R then during the construction of C_{k+1}, either $\mathrm{VX}_{k+1}(i) = 0$ or $\mathrm{VY}_{k+1}(j) = 0$. This is because either row i does not have a faulty cell besides $c(i, j)$ or column j does not have a faulty cell besides $c(i, j)$. This leads to the exclusion of $c(i, j)$ from C_{k+1}. If, on the other hand, $c(i, j)$ is SA-W then it will be included in C_{k+1} but will lead to $\mathrm{VX}_{k+2}(i) = 0$ or $\mathrm{VY}_{k+2}(j) = 0$ and thus excluded from C_{k+2}. That is, C_{k+2} is a strict subset of C_k. Moreover, given that C_k does not contain a loop of faults, then C_{k+2} does not contain a loop of faults either and the process of excluding faults from consecutive subarray continues until an empty C_K is reached. ■

5.2 Coverage failure caused by limited counter capacity

Theorem 2 implies that the process of constructing the invertible set eventually terminates successfully if the fault pattern does not include a loop of faults. However, even in the absence of a loop of faults, the process of constructing the invertible set may fail because of the limited capacity of the counters VX and VY. Specifically, if the maximum capacity of the counters is K and C_K contains both SA-W and SA-R cells, then the construction of the invertible set will fail. We explore the fault configuration that leads to this failure next.

Definition. *A row-column alternating sequence ("RCA sequence") of $2q - 1$ faulty cells ($q > 1$) is a loop of $2q$ faulty cells after excluding one node.*

The above definition implies that every two consecutive faults in an RCA sequence are, alternatively, in the same row or in the same column. If the two first cells in the sequence are in the same column, then the sequence is of the form $c(i_1, j_1)$, $c(i_2, j_1)$, $c(i_2, j_2)$, $c(i_3, j_2)$, ..., $c(i_{q-1}, j_q)$, $c(i_q, j_q)$, while if the first two cells are in the same row, the sequence is of the form $c(i_1, j_1)$, $c(i_1, j_2)$, $c(i_2, j_2)$, $c(i_2, j_3)$, ..., $c(i_q, j_{q-1})$, $c(i_q, j_q)$. The notation in the following definition encompasses both cases.

Definition. *an RCA sequence of $2q - 1$ faulty cells, c_1, c_2, ..., c_{2q-1}, is said to be alternatively-stuck (or "A-stuck") if the first fault in the sequence is SA-W and subsequent faults alternate between SA-R and SA-W. That is, cells c_1, c_3, ..., c_{2q-1} are SA-W, while cells c_2, c_4, ..., c_{2q-2} are SA-R.*

For example, Fig. 3(b) shows an RCA sequence of 7 faults which is obtained by removing cell $c(2, 1)$ from the loop of faults shown in Fig. 3(a). This RCA sequence is A-stuck. The step-like RCA sequence in 3(b) is isomorphic to the RCA sequence and is obtained by interchanging columns 0 and 2, rows 4 and 5, rows 4 and 6 and rows 2 and 7. The proofs of the following theorems are more intuitive if RCA sequences are envisioned as step-like. In general, any RCA sequence can be transformed to a step-like one by a series of row/column interchanges.

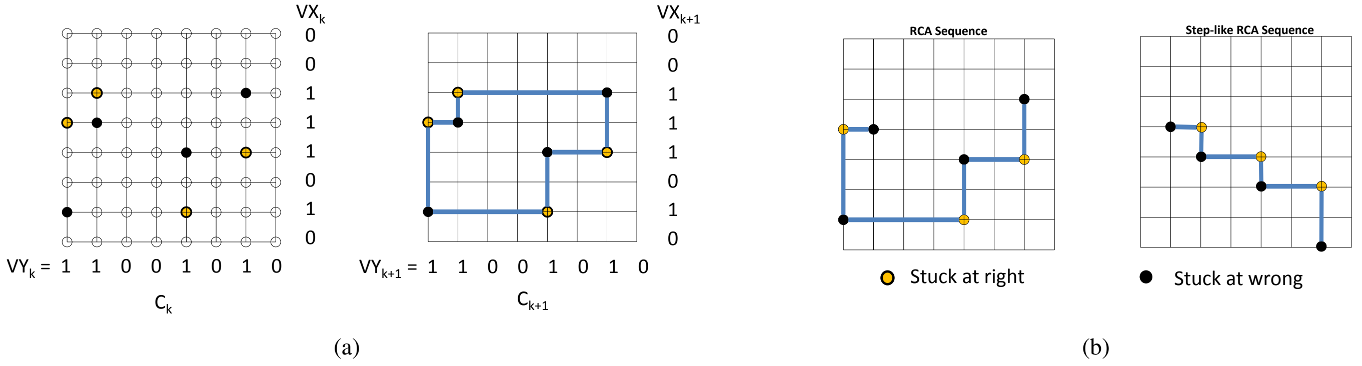


Fig. 3: (a) A loop of faults which cannot be masked ($\mathrm{VX}_k = \mathrm{VX}_{k-1}$ and $\mathrm{VY}_k = \mathrm{VY}_{k-1}$). (b) Two isomorphic RCA sequences of faults that cannot be masked in three iterations.

Theorem 3. *The process of constructing the invertible set fails to terminate after K iterations (with C_K containing only SA-R cells or only SA-W cells) if the original array of cells, C, contains an RCA sequence of at least 2K+1 faults and this sequence is A-stuck.*

Proof. Assume that C contains an RCA sequence c_1, c_2, ..., c_{2q+1} which is A-Stuck. By construction, C_1 contains all the cells in that sequence. However, consider any of the cells c_i, *i = 2,...,2q*. If cell c_i is SA-R, then it is located in a row that contains a SA-W cell and in a column that contains a SA-W cell. Hence, this cell will be included in the subarray C_2. A similar argument applies if c_i is SA-W and consequently C_2 will contain the RCA sequence c_2, c_3, ..., c_{2q}. Applying this argument recursively leads to the conclusion that if $q \geq K$, then the subarray C_K will contain the RCA sequence c_K, ..., $c_{2q+1-(K-1)}$. In other words, if the RCA sequence contains at least *2K + 1* cells, then C_K will contain at least the three cells c_K, c_{K+1} and c_{K+2}. Being three consecutive cells in an RCA sequence, at least one of the cells is SA-R and another is SA-W, which proves the theorem. ∎

Theorem 4. *The invertible set can be constructed in at most K iterations if the longest RCA sequence of faults in the original array of cells, C, contains at most $2K-1$ faults.*

Proof. We prove the theorem by induction. Specifically, we prove three Lemmas: the first establishes the base of the induction, while the other two deal with the induction steps. The proofs of the lemmas are based on the observation that the first and last cells in an RCA sequence are not RC-connected.
Lemma 1: If the longest RCA sequence in *C* is c_1, c_2, ..., c_q, then the longest RCA sequence in C_1 is c_{1+u}, ..., c_{q-v}, where *u, v* $\geq$ *0* and both c_{1+u} and c_{q-v} are SA-W. This is because, by construction, any faulty cell that is not RC-connected in C_1 should be SA-W.
Lemma 2: For *k = 1, 3,...,* if the longest RCA sequence in C_k is $c_1, c_2, \ldots, c_q$, where c_1 and c_q are SA-W, then the longest RCA sequence in C_{k+1} is $c_{1+u}, \ldots, c_{q-v}$, where $u, v > 0$ and both c_{1+u} and c_{q-v} are SA-R. This is because, by construction, any faulty cell in C_{k+1} that is not RC-connected should be SA-R.
Lemma 3: For *k = 2, 4,...,* if the longest RCA sequence in C_k is $c_1, c_2, \ldots, c_q$ where c_1 and c_q are SA-R, then the longest RCA sequence in C_{k+1} is $c_{1+u}, \ldots, c_{q-v}$, where $u, v > 0$ and both c_{1+u} and c_{q-v} are SA-W. This is because, by construction, any faulty cell that is not RC-connected in C_{k+1} should be SA-W.

The above three lemmas prove that for $k = 1, 2, \ldots$, if the longest RCA sequence in C_k includes q cells, then the longest RCA sequence in C_{k+1} includes $q-2$ cells. Therefore, if the longest RCA sequence in C has $2K-1$ cells then the longest RCA sequence in C_K has one cell (SA-W if K is even and SA-R if K is odd). This proves that C_K includes only one type of faulty cells (SA-R or SA-W). ∎

5.3 Defective blocks of storage cells

Consider a storage block of $n \times m$ cells of which F cells are faulty and assume that RDIS is used for masking the faults with the maximum counter capacity of K. Theorem 1-4 identify the only two types of fault patterns that can cause the failure of RDIS to mask the faults: loops of faults and RCA sequence of length $2K+1$. Hence, we call a block of cells *defective* if it contains a loop of faults or an RCA sequence of at least $2K+1$ faults.

If a block of cells with F faults is not defective, then it can be used to write/read any combination of information bits. For a small number of faults, it is possible to compute the probability of having a defective block analytically. For example, three faults cannot form a loop of faults. With four faults, the probability of having a loop of faults in an $n \times m$ block is given by $\binom{n}{2} \cdot \binom{m}{2} / \binom{n \cdot m}{4}$. Applying this formula, we find that the probability of having a defective fault pattern given four faults is 0.0012 when $n = m = 8$ and 0.00008 when $n = m = 16$. Section 9 gives a detailed evaluation of the probability of a block being defective in the presence of F faults.

6 Multidimensional RDIS

In order to enable the determination and retrievel of the invertible set, RDIS adds a number of auxiliary counters. These counters form the major space overhead of RDIS. In Section 3, we have chosen to represent an N bits memory block as a logical two-dimensional array of n rows and m columns. In addition, we have introduced $n+m$ counters that hold the auxiliary information. Accordingly, the physical overhead of RDIS is equal to $\frac{(n+m)\times k}{N}$ where k is the number of bits used for to each auxiliary counter. Interestingly, the concepts of RDIS are not limited to a logical two-dimensional grouping of memory cells. As a matter of fact, the grouping of cells can be extended up to d-dimensional, where d is $log_2(N)$. For example, a 512-bit memory block can be represented as a logical 9-dimensional array i.e. 2^9.

Extending the array dimensions of the logical representation of a memory block reduces the space overhead incurred by RDIS. For example, consider a 512-bit memory block. Assuming each auxiliary counter is 2 bits, the space overhead

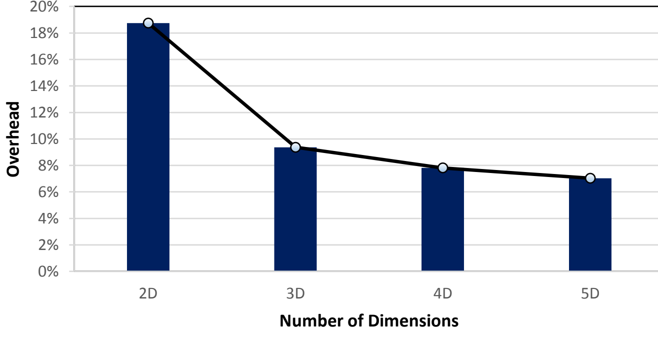


Fig. 4: Space overhead of RDIS for a 512-bit block with various dimensional arrangements.

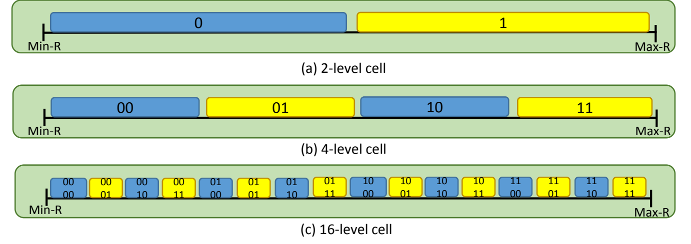


Fig. 5: The concept of multilevel PCM cell [27].

incurred by RDIS for a logical 2-D arrangement of 16×32 is 18.75%. If the number of dimension is extended to 3 i.e. $8 \times 8 \times 8$ logical block, the space overhead of RDIS is reduced to 9.37%. In this sequel, an additional increase in the number of dimensions further reduces the overhead of RDIS. Nevertheless, increasing the number of dimension can have a diminishing return. In fact, the overhead cannot be further reduced when the sum of the dimensions of the dth-D arrangement is equal to that of the d^{th+1}-D arrangement. Going back to our 512-bit memory block example, the space overhead of a 5-D logical arrangement, i.e. $4\times4\times4\times4\times2$, is 7.03%. Moving to a 6-D arrangement, i.e. $4\times4\times4\times2\times2\times2$, keeps the space overhead of RDIS at 7.03%. Thus, increasing the number of dimensions past 5 for a 512-bit block does not bring any merit in terms of space overhead reduction. Fig. 4 shows the decrease in overhead for RDIS as the number of dimension for a logical array arrangement of a 512-bit block increases from 2D to 5D.

On the flip side, RDIS loses some of its correction capability with each move to a higher dimension. However, RDIS still guarantees the recovery from 3 stuck-at faults irrespective of the dimensional representation of the protected block. In Section 9, we show that the likelihood of cycles and RCA sequences increases with the increase in the number of dimensions used to represent a block. Nevertheless, multidimensional RDIS still tolerates more faults that any other scheme at the same level of space overhead.

7 Multilevel Cell PCM

Instead of representing information as the presence or absence of electrical charges, PCM encodes bits in different physical states of chalcogenide alloy that consists of Ge,Sb and Te. Data is stored in PCM devices in the form of either a low resistance crystalline state (SET) or a high resistance amorphous state (RESET). Switching between the states happens through the application of different programming currents that melt and then re-solidify the material into one of the SET/RESET states. As a matter of fact, resistance in the fully crystalline and amorphous states differs by 3 to 4 orders of magnitude [26]. Accordingly, the difference in resistance can be exploited to store multi-bits per cells which results in higher density chips. Instead of representing the data in a cell using two levels, the resistance range can be broken down into multiple levels where each level represents a particular data pattern. Fig. 5 shows a PCM cell arranged in two levels (a), 4 levels (b) and 16 levels (c).

With the stuck-at fault model in binary storage cells, it was straight forward to identify SA-W and SA-R cells and compute the invertible set which contains both SA-W and NF cells and write/read the complement of the data into the cells of the invertible set. In multi-level cells (MLC), each cell can store one of $q > 2$ possible values, with q usually being a power of 2. If a cell can be stuck at one of the possible q values, then a data value, v, to be written on a faulty cell can still be used to differentiate between SA-W and SA-R cells [26]. However, the SA-W cell to which v is to be written may be stuck at a value, u, that is different than the complement of v, and hence the concept of invertible sets would not suffice to mask the faults.

Nevertheless, RDIS can be used to mask faults for MLC. One straight forward way to apply RDIS to MLC is to envision an $n\times m$ array of q level cells as a $n\times m\times log_2(q)$ logical array of binary cells and apply RDIS to the logical array. Hence, RDIS would logically deal with binary cells only. As such, the concepts of RDIS can be extended to MLC.

8 Repairing Defective Blocks

In Section 5, we have proved that a cycle or an RCA sequence of faults are the only patterns that cause RDIS to halt. In this section, we propose techniques that could be adopted to recover from defective patterns. We classify our techniques into two categories. The first encompasses techniques to fix a defective block through breaking cycles or RCA sequences. The second encompasses techniques that try to squeeze more lifetime from defective blocks through a smart usage of block sparing.

8.1 Block Fix

In this section, we present two techniques that could be used to recover from the detrimental effect of RCA sequences and cycles. The first technique, *Pointer Break*, consists of allocating a pointer entry in addition to a patch cell that are to be used to break a cycle or an RCA sequence of faults. The pointer entry specifies the location of a stuck-at cell that is substituted by the patch cell. Choosing any faulty cell in an RCA sequence or cycle is enough to break the defective pattern. However, for an RCA sequence picking the middle cell is recommended as it reduces the probability of forming a new RCA sequence. As a matter of fact, half the total number of faults in the original defective pattern is required for a new RCA sequence to form.

The second technique, *Shift Break*, consists of changing the mapping of the cells into the logical 2D $n \times m$ structure after a defective pattern is formed. One possible implementation would be to shift the position of a cell by its row number modulo m. Though shifting the cells in the block is not masking any of the stuck-at faults, it is likely to cause the faulty cells to form a pattern that is not anomalistic to RDIS. Both techniques are depicted in Fig. 6.

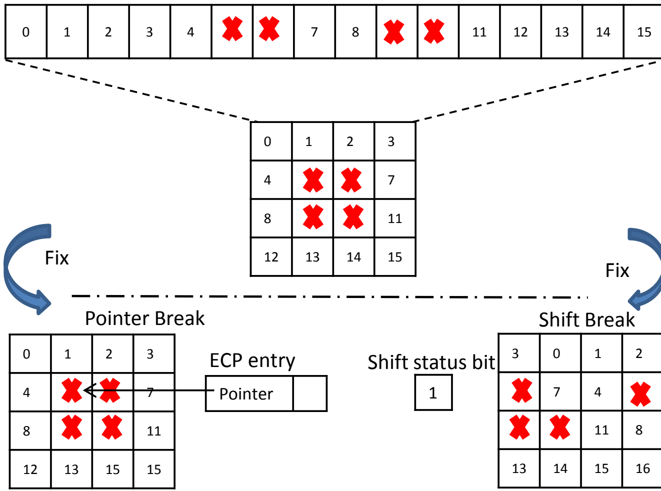


Fig. 6: Defective block fixing techniques.

It is worth mentioning that both techniques are complementary to RDIS. Accordingly, it is a design choice to adopt either of them. *Pointer Break* guarantees the recovery from defective patterns. On the other hand, *Shift Break* cannot guarantee the recovery from defective patterns. While shifting the cells in the protected block breaks the existing defective pattern, other stuck-at faults within the block could form a new defective pattern. Nevertheless, the likelihood of this event is low.

To implement both techniques, *Shift Break* requires a simple remapping function and one additional auxiliary bit that serves as a flag to be set when a shift is applied. On the other hand, *Pointer Break* necessitates $lg(n \times m) + 1$ auxiliary bits. In addition, it requires determining the location of the stuck at cells that formed a defective pattern in order to pick one of them to break the pattern. We evaluate both techniques in Section 9.6.

8.2 Dynamic Sparing

After an error correction scheme fails to recover from errors manifested within a memory block, the common practice is to map-out the memory page where the block resides from the address space. Instead of retiring a whole memory page, Free-p [9] proposed to degrade the memory gracefully through only retiring the block in which errors were failed to be masked. To achieve this goal, a pointer is embedded within the failed block to permanently remap it into a fault free spare block. Recently, Data-dependent sparing [25] was proposed. The data-dependent technique builds on the fact that failures are data-dependent within the context of the stuck-at fault model. Thus, a write operation to a block fails only if the manifested errors form a defective pattern. This said, data-dependent sparing proposed to assign spares temporally and dynamically after a write failure. A later write request to the same location is attempted on the original block. Due to the data dependent nature of errors, the later write is likely to be successful. Subsequently, the previously assigned spare is reclaimed in order to be used as a temporal replacement for other failing blocks. The concept of data dependent-sparing is illustrated in Fig. 7, where blocks 1 and 3 are defective. Fig. 7 (left) shows that writing to block 1 has failed while writing to block 3 has not. This has caused block 1 to be mapped to the spare block. However, a later write to block 1 was successful. Accordingly, the spare block was reclaimed and assigned to block 3 after a write request failed on it as depicted by Fig. 7 (right).

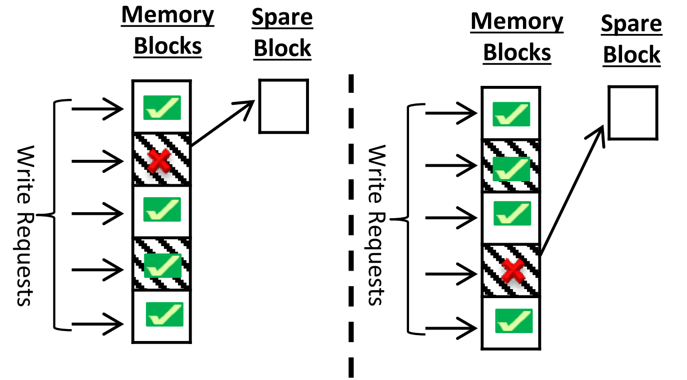


Fig. 7: Data-dependent Sparing. Shaded cells represent defective blocks.

In section 5, we have proved that it is not enough to have a cycle or an RCA sequence of faults for RDIS to halt. However, the cycle or RCA sequence must be alternatively stuck. Thus, failures in RDIS are data dependent. Accordingly, coupling RDIS with the data-dependent sparing technique could have a synergistic effect on the lifetime of a memory device. We study the effect on lifetime in section 9.7.

9 Evaluation

In this section, we rely on Monte Carlo simulation to study the various parameters that affect RDIS as well as to compare it to other schemes. We assume that all cells within a storage block have equal probability of failure. To test if a $n \times m$ storage block having F faulty cells is defective, this block is modeled as a bipartite graph of $(n + m)$ nodes, one for each row and one for each column. If a cell $c(i, j)$ is faulty, then an edge connects the nodes representing row i and column j. A simple variation of depth first search algorithm (DFS) is used to detect the occurrence of a loop. To detect RCA sequences, we keep track of the longest recursion depth executed by DFS while attempting to detect a loop. In other words, our algorithm either detects the existence of a loop, if any, or returns the length of the longest RCA sequence.

9.1 Sensitivity to RDIS Parameters

The block size to be protected, and the overhead of auxiliary counters are the main parameters that affect RDIS. In this section, we study the performance of RDIS in light of these parameters. We simulate RDIS with 5 different block sizes of varying overhead. In addition, each block size is simulated with three variations of RDIS. The first limits the capacity of the auxiliary counters to 3, the second to 7 and the last to the number required to tolerate the maximum possible RCA sequence. Hereafter, we denote these three variations as RDIS-3, RDIS-7 and RDIS-max. For each block size, we report the average number of faults that can be tolerated as well as the probability of failure with F faults for $F = 1, 2, \ldots$. Given a block of size $n \times m$, the corresponding overhead is $(n \cdot s + m \cdot s)/(n \cdot m)$, where s is the size of each auxiliary counter in bits. For example, a 128-byte data block arranged as a 32×32 bit array incurs a 12.5% overhead for $s = 2$. It is to be noted that for a fixed s, the overhead percentage decreases with the increase in the size of the protected storage block.

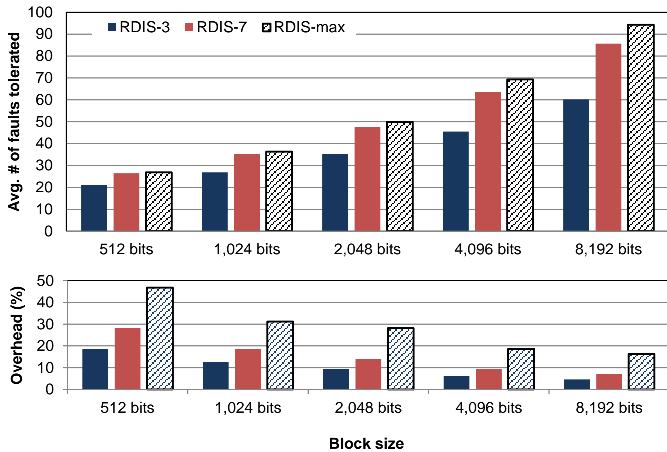


Fig. 8: Average number of faults tolerated within a block and the corresponding overhead.

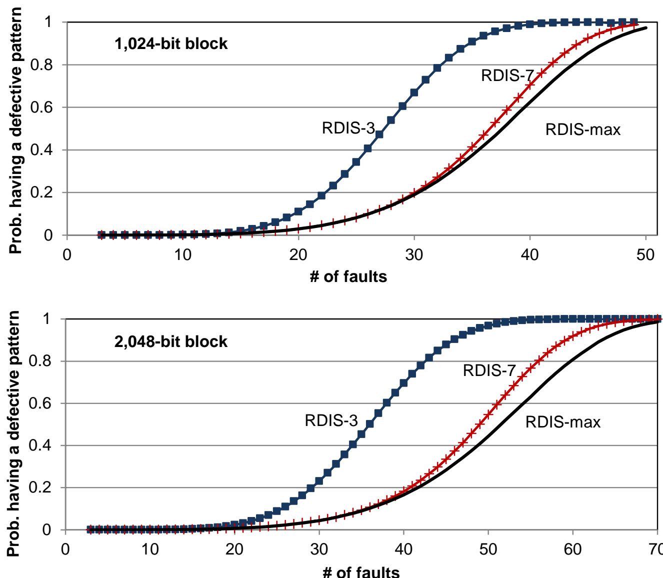


Fig. 9: Probability of failure with F faults.

Fig. 8 shows the average number of faults that can be tolerated for various block sizes and the overhead of the three RDIS configurations. The overhead for RDIS-max is calculated based on the maximum length of an RCA sequence that can occur within a block. Specifically, for $n \times m$ blocks, the maximum length of an RCA sequence is $n + m - 1$ (imagine a step-like RCA sequence starting at $c(0, 0)$ and ending at $c(n - 1, m - 1)$), and thus a counter of size $s = \lceil \log_2((n + m - 1)/2)\rceil$ bits is sufficient for recovery according to Theorem 4. From the results shown in Fig. 8, it can be inferred that the average number of faults tolerated within each block increases with the overhead. Hence, the choice of auxiliary counters capacity for RDIS represents a trade-off between the number of faults that can be tolerated and the overhead. RDIS-3 is shown to correct many errors robustly at the smallest overhead.

The main advantage of RDIS is the large probability to tolerate a relatively large number of faults. Fig. 9 shows that given F faults, the probability of forming a loop or RCA sequence increases at low pace with the increase of F. Accordingly, RDIS is capable of tolerating a high number of faults beyond what it guarantees. Even though we show the results for blocks of 1,024 bits and 2,048 bits, other block sizes exhibit the same trend and are omitted for brevity. These results show that RDIS-3 is capable of tolerating a notable number of faults while incurring an affordable overhead. As a matter of fact, the relative increase in the number of faults tolerated by increasing the counters capacity beyond three is not proportional to the increase in the overhead. Consequently, we consider only RDIS-3 in the rest of our evaluation.

9.2 Comparison with existing schemes

In this section, we evaluate the performance of RDIS against other schemes. Specifically, we compare RDIS-3 with SAFER which was shown in [8] to be superior to ECP and ECC. The overhead of SAFER depends on the number of groups that a block is partitioned into.[3] Hence, RDIS-3 is compared with two SAFER configurations that have an overhead just smaller and just larger than RDIS-3. Two metrics are used for comparison: (1) the probability of failure with F faults; and (2) the average number of faults that can be tolerated in a storage block.

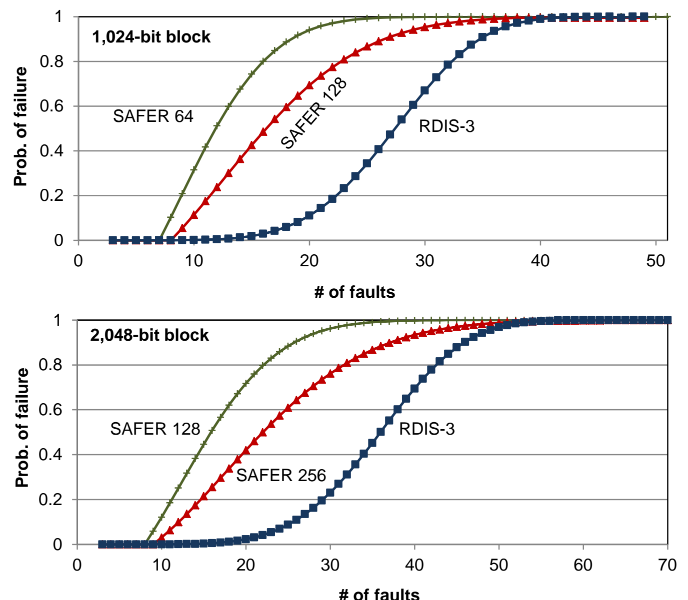


Fig. 10: RDIS vs. SAFER: Probability of failure with F faults.

Both RDIS and SAFER can probabilistically tolerate more faults than what they guarantee. With n groups, SAFER (denoted by SAFER n) guarantees the tolerance of $\log_2 n + 1$ faults while RDIS can always tolerate three faults. Any additional fault is tolerated by both schemes with a certain probability. Fig. 10 shows the probability of failure of a storage block after F faults. Though SAFER guarantees the tolerance of more faults than RDIS, the probability of failure after what it guarantees increases at a high rate. On the contrary, the probability of failure for RDIS increases at a substantially low rate. In addition, the probability of failure for RDIS in the interval of faults that SAFER guarantees is remarkably low as depicted in Table 1, even when compared with the higher overhead version of SAFER. Though we show the results for two different block sizes for brevity, the same trend is manifested with other block sizes.

3. The overhead of SAFER when used to protect a block of N bits using n groups is: $(\lceil \log_2 n\rceil \times \lceil \log_2 \lceil \log_2 N\rceil\rceil) + \lceil (\log_2 \lceil \log_2 n\rceil + 1)\rceil + n$.

	Faults #	4	5	6	7	8	9	10	11	12	13
1 Kbits	SAFER 128	0	0	0	0	0	0.055	0.11	0.17	0.23	0.30
	RDIS-3	6×10^{-6}	3×10^{-5}	8×10^{-5}	2×10^{-4}	4×10^{-4}	0.00008	0.0015	0.0025	0.0045	0.0074
2 Kbits	SAFER 256	0	0	0	0	0	0	0.03	0.06	0.09	0.13
	RDIS-3	2×10^{-6}	5×10^{-6}	1×10^{-5}	4×10^{-5}	1×10^{-4}	2×10^{-4}	0.00033	0.00057	0.00093	0.0015

TABLE 1: RDIS vs. SAFER: Probability of failure.

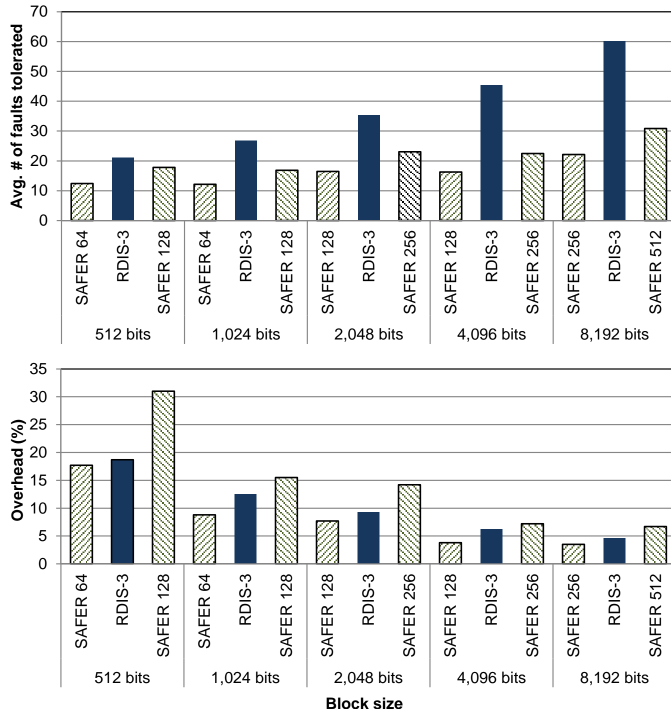


Fig. 11: RDIS vs. SAFER: Average number of tolerated faults and the corresponding overhead.

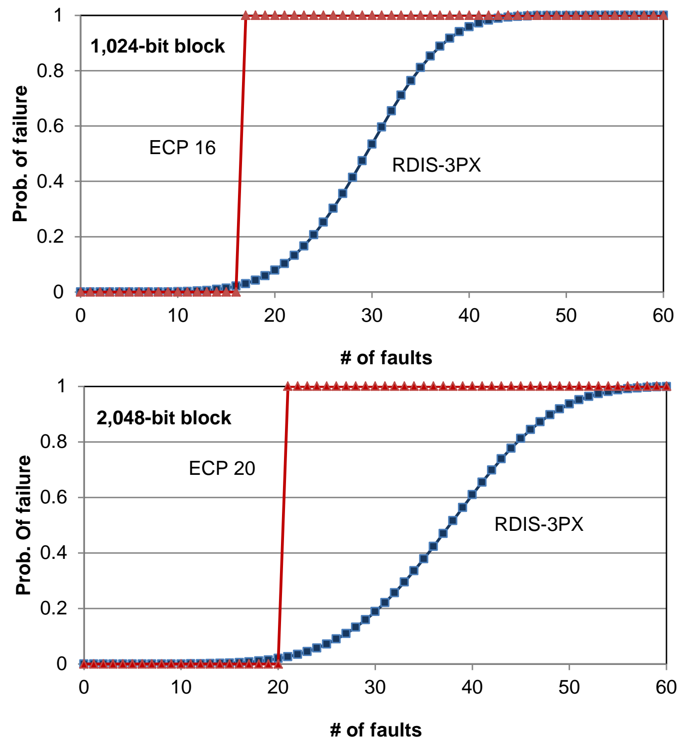


Fig. 12: RDIS-3PX vs. ECP: Probability of failure with F faults.

The advantage of RDIS over SAFER, when it comes to the low probability of failure, is manifested by the average number of faults that each scheme can tolerate as shown in Fig. 11. The results show a significant advantage for RDIS over SAFER. For example, RDIS-3 is capable of tolerating 18% more faults than SAFER 128 with a 512-bit block size and 95% more faults than SAFER 512 with a 8,192-bit block. Note that this increase in the average number of faults tolerated is realized with lower overhead.

9.3 Protecting auxiliary data

Similarly to SAFER, RDIS cannot recover from faults in the auxiliary bits. Specifically, it is assumed that the storage of those bits is error free. The ECP scheme [7] is different in that regard in the sense that it can protect the cells that replace faulty cells. To this end, we can use ECP to protect the auxiliary counters of RDIS-3 against faults. For this, we can allocate π pointers to protect the auxiliary bits. We simulated RDIS-3 with various values of π and concluded that $\pi = 5$ is a suitable value since it maintains the high number of faults tolerated when counters are assumed to be fault-free. Hereafter, we denote the scheme that protects the auxiliary bits of RDIS-3 (can be applied to any version of RDIS) as RDIS-3PX.

Subsequently, we compare RDIS-3PX against ECP itself. We assign to ECP the minimum number of pointers, n, that makes its overhead larger than RDIS-3PX and denote the scheme by ECP n.[4] For various block sizes, we study the probability of failure with F faults as well as the average number of tolerated faults achieved by each scheme. When it comes to the probability of failure with F faults, Fig. 12 shows that ECP cannot recover from faults beyond the provided number of correction pointers. To the contrary, RDIS is capable of remarkably tolerating faults beyond what it guarantees. Furthermore, RDIS exhibits a notably low probability of failure within the error free window of ECP. Again, these results are manifested in the average number of faults that both schemes can tolerate as depicted in Fig. 13. For example, RDIS tolerates up to 81% more faults with block size of 8,192 bits. It is to be noted that RDIS' average number of faults tolerated corresponds to faults occurring both in the protected block and the auxiliary bits.

The presented results make it clear that RDIS can tolerate more faults with higher probability than previously proposed schemes using the same assumptions and fault model. It is particularly suited for large blocks of 128 bytes or more.

4. The overhead of ECP n when used to protect a block of N bits using n pointers is: $n(\lceil \log_2 N \rceil + 1) + 1$.

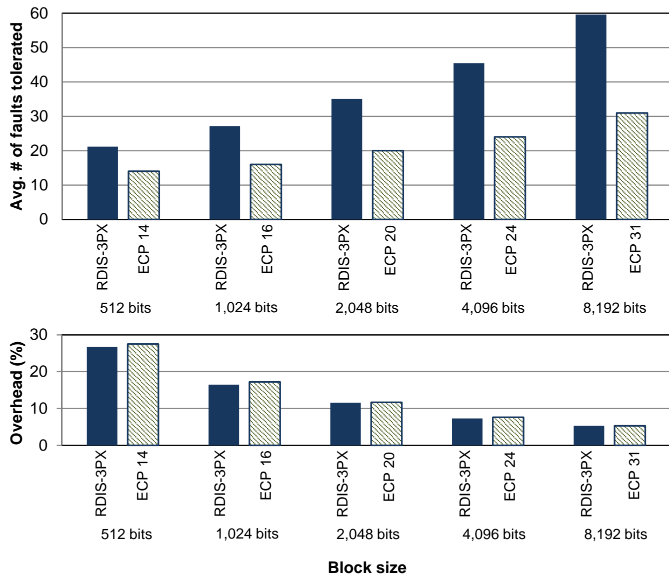


Fig. 13: RDIS-3PX vs. ECP: Average number of tolerated faults and the corresponding overhead.

9.4 Multidimensional RDIS

In Section 6, we have presented multidimensional RDIS that consists of extending the number of dimensions of the logical array used to envision a protected memory block. We have shown that extending the number of dimensions reduces the space overhead incurred by RDIS. In this section, we study the impact of extending the number of dimension on the error correction capability of RDIS. To this end, we report the average number of faults that can be tolerated by RDIS-3 with a 2D, 4D and 6D arrangement of memory cells for 4096-bit block. We limit the maximum number of dimensions to 6 as no further reduction in space overhead can be achieved past a 6D arrangement. In addition, we compare multidimensional RDIS to SAFER where each configuration of RDIS is compared to that of SAFER k that is either at the same level of space overhead or slightly larger where k is the number of groups a block is arranged into and $log_2(k)$ is the number of faults that can be tolerated. Moreover, we compare RDIS to BCH. The latter is an industry standard multi-bit error correction scheme. A major difference between RDIS and BCH is the probabilistic nature of the error correction capability. While RDIS can tolerate a large number of faults beyond the threshold it guarantees with high probability, the capability of a deployed BCH code predetermines the maximum number of faults that can be tolerated. Similarly to SAFER, we compare RDIS to a BCH t code that is either at the same level of space overhead or slightly larger where t indicates the maximum number of faults that BCH can tolerate.

Fig. 14 shows the average number of faults that can be tolerated by each of the three schemes. The results reveal that RDIS maintains its superiority to other schemes even with higher dimensions. Nevertheless, RDIS loses some of its correction capability with the increase in the number of dimensions used to logically arrange a protected block. Our findings indicate that the probability of block defectiveness for RDIS, i.e. the probability of forming a cycle or an RCA-sequence, increases with the increase in the number of dimensions. In fact, a new plane is added with each additional dimension. Each plane can be envisioned as a sub-block. With the increase in the number of dimensions, the size of each plane decreases. In accordance with our results from previous sections, the error correction capability of RDIS depends of the size of the protected block. Hence, a smaller plane size implies a higher probability of defectiveness. The probability of defectiveness of RDIS for 4096-bit block with 2D, 4D and 6D arrangement is ploted in Fig. 15. At last, other block sizes show the same pattern when it comes to the average number of tolerated faults and the probability of defectiveness. Accordingly, their results were omitted for brevity.

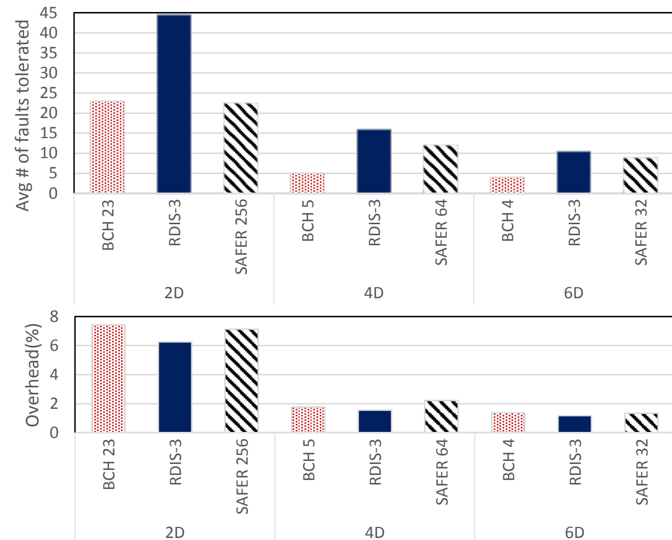


Fig. 14: Average number of faults that can be tolerate by RDIS with various dimensional arrangement compare to SAFER and BCH for 4096-bit block.

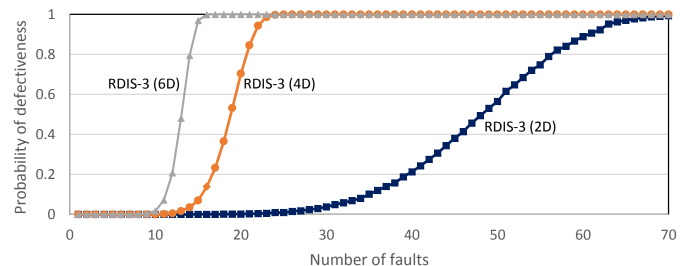


Fig. 15: Probability of defectiveness with different dimensional arrangements for a 4096-bit block.

9.5 Extra Write Effect

As indicated in section 4, RDIS requires an extra write operation to reveal the fault information and mask erroneous cells. This extra write could exacerbate the wear-out rate of non-faulty cells that happen to be on the intersection of a row and column both containing stuck-at faults (i.e. row-column connected). In this section, we study the effect of extra writes on the lifetime of a memory block. To this end, we compare the number of writes that can be executed on a memory block in two settings. The first assumes that the fault information is cached; thus only one write operation is required. The second assumes no knowledge about the fault information; thus additional write operations are required. We lay down 2000 PCM blocks of various sizes and assign a lifetime to each cell drawn from a Gaussian distribution with a mean of

10^8 and a standard deviation of 25×10^6 [7]. Fig. 16 plots the lifetime decrease in terms of the total number of writes executed when extra writes could occur relative to one write.

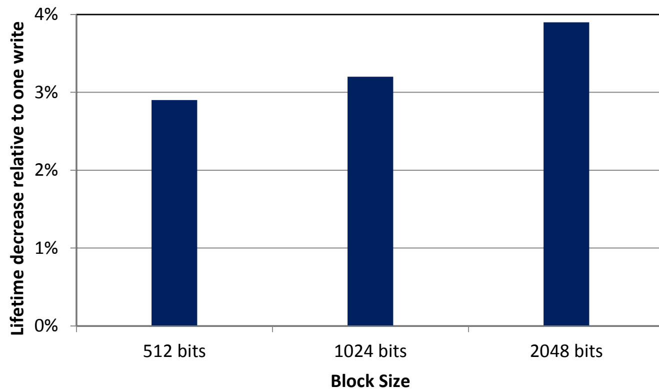


Fig. 16: Lifetime decrease due to extra writes.

Fig. 16 shows that the decrease in lifetime due to extra writes is notably low. This result is attributed to the fact that row-connected cells are not always part of the initial mesh that RDIS forms unless both row and column contain stuck-at wrong cells. Thus, extra writes to healthy row-connected cells happens occasionally due to the data dependent nature of errors i.e. stuck-at wrong. In the event that a row-connected cell wears-out earlier than expected due to extra writes, this cell is harmful only when it leads to the formation of a cycle or an RCA sequence of stuck-at cells. Therefore, the extra writes incurred by RDIS harm the lifetime marginally. Nevertheless, a cache is still beneficial to eliminate the performance overhead of the extra writes.

9.6 Block Fix

In section 8.1, we have presented two techniques to break defective patterns that cause RDIS to halt. In this section, we evaluate these two techniques in terms of the average number of additional faults that can be tolerated after breaking a defective pattern in a memory block that suffers from k stuck-at faults.To this end, we resort to Monte-Carlo simulation. We start with a block that already has k faults and is defective whether because of an RCA sequence or a cycle.Subsequently, we break the defective pattern with both techniques and record the additional faults that could be tolerated in the block until a new defective pattern is formed. We ran the experiment for million times and thes results are depicted in Fig. 17.

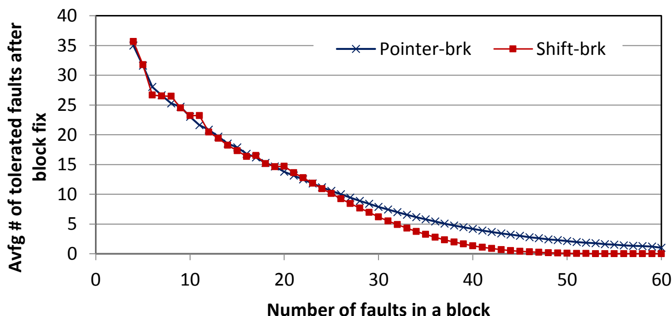


Fig. 17: Avg. number of additional tolerated faults after breaking a defective pattern in a 2048-bits block size.

It is notable that both techniques are capable of significantly tolerating a large number of faults after fixing a block in which a defective pattern occurred with a relatively small number of faults. This finding is a direct consequence of the low probability of defectiveness that RDIS exhibits with a small number of faults in the block. Hence, fixing a block that got defective with a small number of faults yields into a greater number of faults that can be tolerated after the fix.

In addition, it is notable that fixing a defective block with a pointer performs better when the number of faults in the protected block is high. By shifting the cells in a block, the defective pattern is broken. However, a new defective pattern could form due to the large number of faults already existing in the block. On the other hand, fixing a block with a pointer is guaranteed to break the defective pattern. Nevertheless, implementing the shifting technique is simple and easy. It only requires one additional bit of overhead to indicate whether the data was written shifted or not and tolerates a significant number of additional faults.

9.7 Graceful Degradation

In section 8.2, we have proposed to couple RDIS with data-dependent sparing. In this section, we study the effect on lifetime of coupling data-dependent sparing with RDIS as compared to coupling the static sparing technique with RDIS. To this end, we laid down 2000 physical pages each composed 512-bits memory blocks. We followed the Free-p [9] approach in assuming that the OS is responsible of dispatching a memory page that serves as a set of spares for defective blocks. we assign to each cell a lifetime drawn from a normal distribution with mean 10^8 and standard deviation of 25×10^6. We ran our simulation until all memory pages have been retired i.e. all memory blocks became defective and record the total number of writes executed. The results are shown in Fig. 18.

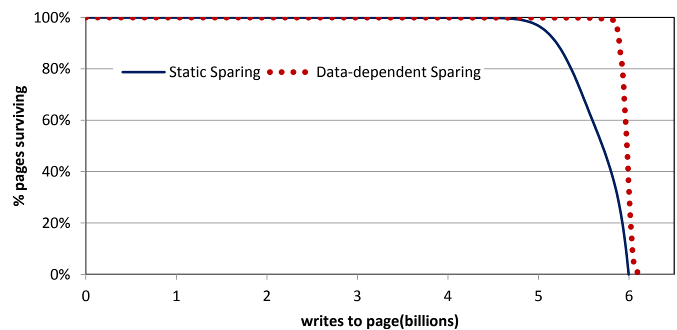


Fig. 18: Data-dependent sparing vs. static sparing effect of lifetime when couple with RDIS.

The results show that coupling RDIS with data-dependent sparing is capable of extending the life significantly. When RDIS + data-dependent sparing still enjoy 100% of the memory blocks, RDIS + static sparing retires around 50% of the blocks. This finding is a direct consequence of the data-dependent nature of failures exhibited by RDIS where defective blocks can still be written reliably except with few specific data patterns.

10 Conclusions

The limited write endurance is the major weakness of emerging resistive memories. Accordingly, robust error recovery

schemes are required to mask off hard errors and prolong the lifetime of a resistive memory chip. In this paper, we have presented and evaluated RDIS, a recursively defined invertible set scheme to tolerate multiple stuck-at hard faults. Our extensive evaluation shows that RDIS achieves a very low probability of failure on hard fault occurrences, which increases slowly with the relative increase in the number of faults. This characteristic allows RDIS to effectively recover from a large number of faults. For example, RDIS can recover from 46 hard faults on average when the block size is 512 bytes (storage sector size) while incurring a low overhead of 6.2%.

Given its high error tolerance potential, RDIS fits the need to recover from many faults in emerging resistive memories. We believe that RDIS provides a very robust memory substrate to a system and allows system designers to focus their efforts on effective integration and management of resistive memory capacity at higher levels, for better overall system performance and reliability.

References

[1] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," 2005, pp. 323–326.
[2] ITRS, http://public.itrs.net, 2011.
[3] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 439–447, July 2008.
[4] B. C. Lee *et al.*, "Architecting phase change memory as a scalable dram alternative," *SIGARCH Comput. Archit. News*, vol. 37, pp. 2–13, June 2009.
[5] M. K. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *MICRO Conference*, dec. 2009, pp. 14–23.
[6] P. Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," *SIGARCH Comput. Archit. News*, vol. 37, pp. 14–23, June 2009.
[7] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," *SIGARCH Comput. Archit. News*, vol. 38, pp. 141–152, June 2010.
[8] N. H. Seong *et al.*, "SAFER: Stuck-At-Fault Error Recovery for Memories," in *MICRO Conference*, dec. 2010, pp. 115–124.
[9] D. H. Yoon and Others, "FREE-p: Protecting non-volatile memory against both hard and soft errors," in *HPCA conference*, feb. 2011, pp. 466–477.
[10] S. Kang *et al.*, "A 0.1 μm 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) With 66-MHz Synchronous Burst-Read Operation," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 1, pp. 210–218, jan. 2007.
[11] K.-J. Lee *et al.*, "A 90 nm 1.8 V 512 Mb Diode-Switch PRAM With 266 MB/s Read Throughput," *IEEE JSSC*, vol. 43, pp. 150–162, January 2008.
[12] K. Bourzac, "Memristor memory readied for production," http://www.technologyreview.com/computing/25018/, April 2010.
[13] I. Micron Technology, "Phase change memory (pcm)," http://www.micron.com/products/pcm, 2011.
[14] S. Lee *et al.*, "A Study on the Failure Mechanism of a Phase-Change Memory in Write-Erase Cycling," *IEEE Electron Device Letters*, vol. 30, no. 5, pp. 449–450, May 2009.
[15] M. Qureshi *et al.*, "Practical and secure PCM systems by online detection of malicious write streams," in *HPCA Conference*, feb. 2011, pp. 478–489.
[16] N. H. Seong *et al.*, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *SIGARCH Comput. Archit. News*, vol. 38, pp. 383–394, June 2010.
[17] B.-D. Yang *et al.*, "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme," ser. ISCAS, May 2007, pp. 3014–3017.
[18] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," in *MICRO Conference*, Dec. 2009, pp. 347–357.
[19] R. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, vol. 26, no. 2, pp. 147 – 160, 1950.
[20] R. Bose and D. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 1, pp. 68–79, 1960.
[21] D. Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in *ACSSC Conference*, nov. 2006, pp. 1183–1187.
[22] W. Wong, "A chat about micron's clearnand technology," *electronic design*, December 2010.
[23] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *MICRO-44*, 2011.
[24] H. Chung *et al.*, "A 58nm 1.8V 1Gb PRAM with 6.4MB/s Program BW," in *IEEE ISSCC*, February 2011, pp. 500–502.
[25] R. Maddah *et al.*, "Data dependent sparing to manage better-than-bad blocks," *Computer Architecture Letters*, 2012.
[26] G. Burr *et al.*, "Phase change memory technology," *Journal of Vacuum Science Technology*, vol. 28, no. 2, pp. 223–262, 2010.
[27] M. Qureshi *et al.*, "Morphable memory system: a robust architecture for exploiting multi-level phase change memories," in *ISCA*, 2010.

Rakan Maddah received his B.S. and M.S. degree in Computer Science from the Lebanese American University in 2007 and 2009 respectively. He joined the Computer Science Department at the University of Pittsburgh as a Ph.D. student since 2010. His research interests are in computer architecture, storage devices, systems and fault tolerance. He is a student member of IEEE.

Rami Melhem received a B.E. in Electrical Engineering from Cairo University in 1976, an M.A. degree in Mathematics and an M.S. degree in Computer Science from the University of Pittsburgh in 1981, and a Ph.D. degree in Computer Science from the University of Pittsburgh in 1983. He was an Assistant Professor at Purdue University prior to joining the faculty of the University of Pittsburgh in 1986, where he is currently a Professor in the Computer Science Department which he chaired from 2000 to 2009. His research interests include Power Management, Parallel Computer Architectures, Real-Time and Fault-Tolerant Systems, Optical Networks and High Performance. Dr. Melhem served and is serving on program committees of numerous conferences and workshops and on the editorial boards of the IEEE Transactions on Computers (1991-1996, 2011-), the IEEE Transactions on Parallel and Distributed systems (1998-2002), the Computer Architecture Letters (2001-2010), the Journal of Parallel and Distributed Computing (20032011) and The Journal of Sustainable Computing, Informatics and Systems (2010 -). Dr. Melhem is a fellow of IEEE and a member of the ACM.

Sangyeun Cho received the BS degree in computer engineering from Seoul National University in 1994 and the PhD degree in computer science from the University of Minnesota in 2002. In 1999, he joined the System LSI Division of Samsung Electronics Co., Giheung, Korea, and contributed to the development of Samsung's flagship embedded processor core family CalmRISC(TM). He was a lead architect of CalmRISC-32, a 32-bit microprocessor core, and designed its memory hierarchy including caches, DMA, and stream buffers. Since 2004, he has been with the Computer Science Department at the University of Pittsburgh, where he is currently an associate professor. His research interests are in the area of computer architecture and system software with particular focus on performance, power and reliability of memory and storage hierarchy design for next-generation multicore systems.