SAFER: Stuck-At-Fault Error Recovery for Memories

Nak Hee Seong[†] nhseong@ece.gatech.edu
Dong Hyuk Woo[†*] dhwoo@ece.gatech.edu
Vijayalakshmi Srinivasan[‡] viji@us.ibm.com
Jude A. Rivers[‡] jarivers@us.ibm.com
Hsien-Hsin S. Lee[†] leehs@gatech.edu

[†]School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332

[‡]IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT

As technology scaling poses a threat to DRAM scaling due to physical limitations such as limited charge, alternative memory technologies including several emerging non-volatile memories are being explored as possible DRAM replacements. One main roadblock for wider adoption of these new memories is the limited write endurance, which leads to wear-out related permanent failures. Furthermore, technology scaling increases the variation in cell lifetime resulting in early failures of many cells. Existing error correcting techniques are primarily devised for recovering from transient faults and are not suitable for recovering from permanent stuck-at faults, which tend to increase gradually with repeated write cycles.

In this paper, we propose SAFER, a novel hardware-efficient multi-bit stuck-at fault error recovery scheme for resistive memories, which can function in conjunction with existing wear-leveling techniques. SAFER exploits the key attribute that a failed cell with a stuck-at value is still readable, making it possible to continue to use the failed cell to store data; thereby reducing the hardware overhead for error recovery. SAFER partitions a data block dynamically while ensuring that there is at most one fail bit per partition and uses single error correction techniques per partition for fail recovery. SAFER increases the number of recoverable fails and achieves better lifetime improvement with smaller hardware overhead relative to recently proposed Error Correcting Pointers and even ideal hamming coding scheme.

Keywords

multi-bit error correction, stuck-at fault recovery, reliability, write endurance, resistive memory, phase-change memory

1. INTRODUCTION

Process technology scaling poses a threat to DRAM scaling due to physical limitations, such as limited charge, which leads to lower retention time and unreliable sensing of charge on the trench capacitor [1, 9]. Emerging memories such as phase-change memories that use different states of atomic structure to store data, and the resistive drop through the atomic arrangement to sense the stored data are promising alternatives to overcome the DRAM scalability wall. Additionally, some of these emerging memory technologies enable high density by representing multiple bits per cell using fine adjustment of atomic arrangements. Furthermore, these memories are free of charge leakage, and they are projected to have more than ten years of retention time [1]. Among the emerging memories, such as Phase-change memory (PCM), magneto-resistive RAM (MRAM), spin-torque transfer memory (STT-RAM), and ferroelectric RAM (FeRAM), PCM appears to be the forerunner being closest to mass production.

[*]Now with Platform Architecture Research, Intel Labs

A PCM cell typically uses chalcogenide alloy composed of **Ge**, **Sb**, and **Te**. The material has two distinct states, namely, a low resistive crystalline state and a high resistive amorphous state. Using fine grain partitioning of the resistance range between the two states, it is possible to store multiple bits per PCM cell. Although PCM is slower than DRAM to read and much slower to write, architecture-level solutions have been explored to mitigate these high latencies, and to effectively use PCM as a DRAM replacement for main memory [10, 15]. One of the roadblocks to wider adoption of PCM as main memory is its limited write endurance, which is around 10^8 writes [7]. As writes result in repeated expansion and contraction of the chalcogenide alloy due to state change of the cell, there is a higher probability of the material physically detaching from the heating element resulting in the cell being permanently stuck-at a value. Prior work addresses the write-related failures of PCM by limiting the write frequency [3, 10, 21, 22], or by using wear-leveling to evenly spread the writes [16, 18, 22]. However, as technology scales, the lifetime variation of cells increases, resulting in early failures of many cells. Additionally, there is no spatial correlation of failures among neighboring cells, which implies that, in the absence of error recovery techniques, the lifetime of the PCM memory is dictated by the weakest cell. Recent studies use operating system page remapping [6] and error correcting pointers [17] to recover from stuck-at faults of PCM.

In this paper, we propose SAFER, an alternative approach to recover from stuck-at faults, which exploits the fact that a failed cell with a stuck-at value is still readable and uses the failed bit to continue to store data. We propose a dynamic data block partition technique that ensures each partition has at most one failed bit; thereby, enabling recovery per partition using single bit error correction techniques. SAFER is complementary to previously proposed wear-leveling techniques and can be used in conjunction with them.

The SAFER concept can virtually be applied to any emerging memory technology which suffers from permanent stuck-at faults. Depending on the underlying memory technology and the cost-effectiveness of the implementation, SAFER would incur varying overheads and result in different design trade-offs for the corresponding memory technologies. In this paper, we have chosen to demonstrate the effectiveness of SAFER with PCM as the memory technology example.

The rest of this paper is organized as follows. Section 2 describes background and motivation. Section 3 introduces Stuck-At-Fault Error Recovery (SAFER). Section 4 discusses efficient implementation options for SAFER. We present our evaluation methodology in Section 5, and results in Section 6. We conclude in Section 7.

2. BACKGROUND AND MOTIVATION

Emerging resistive memories such as PCM are typically endurance limited. In the case of PCM, repeated writes, which cause the cell to change state (amorphous or crystalline) also leads to mechanical stress, which in turn increases the chances of cells having permanent stuck-at faults. Since PCM uses atomic arrangement to store data, it is not susceptible to alpha particle induced transient errors. Possible reasons for transient errors may be thermal drift in multi-level cells or proximity disturbance due to writes to neighboring cells. However, transient errors are not a problem for PCM because thermal insulation between cells [8] and periodic refresh can be used to mitigate them. Hence, our focus is on recovery from permanent faults for PCM.

The existing error correcting code (ECC) schemes can be applied to recover from permanent stuck-at faults even though they are primarily devised for recovering from transient faults. For example, among the existing ECC schemes, the (72,64) Hamming Coding scheme is the most popular one and is currently adopted to correct single transient errors in DRAM memories. The (72,64) code uses eight more bits per 64 bits of data to recover from at most one fail. This is usually sufficient because typically alpha particle induced transient errors are rare events, and the probability of two failures within 64 bits is very low [12]. However, unlike transient errors, the number of stuck-at faults gradually grows with time (with repeated write cycles), making it necessary to provide efficient multi-bit error correction capability. Furthermore, an ECC scheme does not even provide efficient mechanisms to identify the position of the fail bits. More importantly, due to its nature, ECC bits wear out much faster than their corresponding data cells, thus ECC is not appropriate for emerging memory technologies that suffer from the limited write endurance.

One of the simplest ways to maintain memory integrity in the presence of more than one stuck-at fault per data block is for the Operating System (OS) to exclude the page containing the failed data block from being allocated. However, this might lead to many pages becoming unusable even when there are only two fails per data block.

Recently, architectural techniques have been proposed to overcome multiple stuck-at faults in PCM [6, 17]. Ipek *et al.* [6] proposed Dynamic Pairing scheme to reuse faulty pages. In the Dynamic Pairing scheme, each byte has its own fail indication bit. If a new fail occurs, the indication bit of the corresponding byte is set, and the OS adds the corresponding page to a waiting list of faulty pages. On a page allocation, the OS selects a pair of faulty pages such that their fail bits are not at the same offset within the page. One of the pages of the pair is maintained as the primary copy, and the other as a backup copy. Dynamic Pairing provides the ability to reuse faulty pages with more than one fail bit per data block. However, it does not support wear-leveling techniques that manipulate memory block addresses to uniformly spread the writes. This makes the memory system vulnerable to malicious attacks, especially when the OS is compromised [18].

Error-Correcting Pointer (ECP) scheme [17] stores six fail pointers for each 512 bits of data block and replaces the fail cells with extra/spare cells. This ECP scheme is more efficient than the (72,64) code from the standpoint of both hardware overhead and fail recovery because it can recover six fails per 512 bits with 61-bit overhead. Furthermore, this technique operates in the presence of existing wear-leveling algorithms.

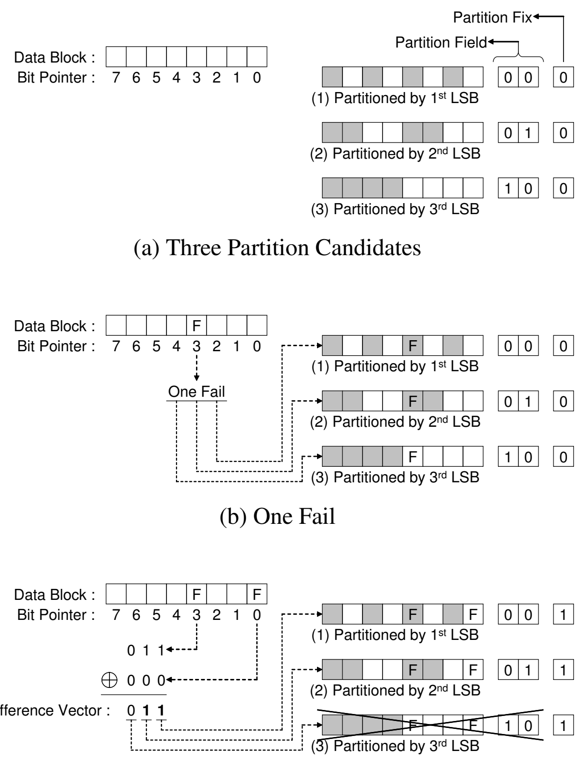
The main motivation for our work is to reduce the hardware overhead for multi-bit stuck-at fault error recovery. One of the key attributes of stuck-at faults is that the cell with a stuck-at value is still readable. We exploit this property to reuse the faulty cell with the stuck-at value to provide hardware efficient multi-bit stuck-at fault error recovery. This becomes necessary because, with technology scaling of resistive memories, the non-uniform distribution of lifetime variations may be exacerbated leading to more frequent occurrences of multiple permanent stuck-at faults per data block.

3. SAFER: STUCK-AT-FAULT ERROR RECOVERY

We now describe our stuck-at-fault error recovery (SAFER) technique, which enables a hardware-efficient multi-bit error recovery by dynamically partitioning the data blocks to ensure that each partition has at most one fail bit. We begin with a discussion of how to partition a data block such that each partition has at most one fail bit, and then describe how to recover from those fail bits.

3.1 Partition Technique of SAFER for Double Error Correction

We first explain how we partition a data block for double error correction (DEC). The key idea of SAFER for DEC is to partition a data block into two groups ensuring that the two fail bits belongs to different groups and to use single error correction (SEC) technique per group.


Partition Fix
Partition Field
Data Block :
Bit Pointer : 7 6 5 4 3 2 1 0
(1) Partitioned by 1st LSB
(2) Partitioned by 2nd LSB
(3) Partitioned by 3rd LSB
One Fail
0 1 1
⊕ 0 0 0
Difference Vector : 0 1 1

(a) Three Partition Candidates

(b) One Fail

(c) Two Fails

Figure 1: Example of Partitioning Two Fails

If we assume an n bit data block, there are $\frac{C^n_{n/2}}{2}$ possible ways to partition the block into two $n/2$ bit groups. However, if the goal is to only ensure that the two fail bits are not in the same group, the number of ways to partition them into two groups is reduced to only $\lceil log_2 n \rceil$. We now describe the partition technique to handle DEC using an example shown in Figure 1.

The partition technique of SAFER identifies the location of each data bit in a block using a bit pointer. Each data bit is assigned a bit pointer using $\lceil log_2 n \rceil$ bits. Figure 1(a) shows an example of partitioning an eight bit data block into two groups. Three bits are required to represent each bit position in this eight bit block. In this figure, each box indicates one data bit cell and gray and white boxes are used to indicate two different groups, say G and W. The data block can be partitioned into two groups in three different ways, namely, GWGWGWGW, GGWWGGWW, and GGGGWWWW, based on whether the least significant bit (LSB), the second LSB, or the most significant bit (MSB) of the three-bit bit pointer is used, respectively. In other words, all possible 28 fail bit-pairs that are selected in the eight-bit block can be separated into two groups by using one of these three patterns.

Thus a block with at most one fail bit can be partitioned by using any arbitrary bit of a bit pointer. Figure 1(b) shows that if the first bit to fail is at bit position 3, any of the three ways of partitioning discussed above can be used.

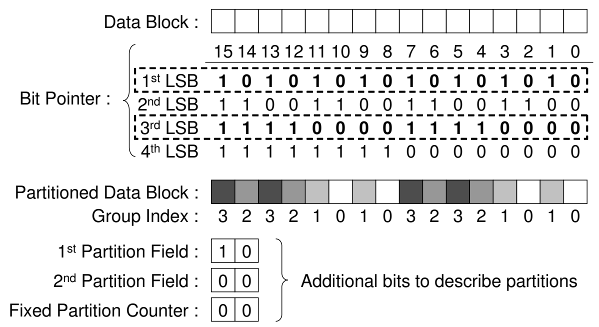
Now, if the second bit to fail is at bit position 0 as shown in Figure 1(c), the partition should be fixed to separate the two fail bits into different groups. The partition technique uses XOR operation to determine the difference vector of the two fail pointers ($000 \oplus 011 = 011$). The number of 1s in the difference vector indicates the possible choices for partitioning the data. With two bits being 1 in the difference vector there are two ways to partition the data block. If we choose the first LSB of the difference vector, the resulting partition is shown in Figure 1(c)(1), and instead if we choose the second LSB of the difference vector, the resulting partition is shown in Figure 1(c)(2).

The "partition field" identifies which bit of the difference vector was used to partition the data block. For a n bit data block, the "partition field" uses $\lceil log_2(\lceil log_2 n \rceil) \rceil$ additional bits to identify how a block is partitioned. For our example in Figure 1, with an eight bit data block, the partition field is $\lceil log_2(\lceil log_2 8 \rceil) \rceil (= 2)$ bits with a value of either "00","01", or "10" depending on the bit position (the first, second or third LSB) of the difference vector chosen for partitioning the data. Furthermore, the partition is not fixed unless there are two fail bits. Hence, a "partition fix" bit is used to indicate whether the partition is fixed, or not. In Figure 1(b) the "partition fix" bit is set to 0, and it is set to 1 only in Figure 1(c) as soon as there is a second fail bit.

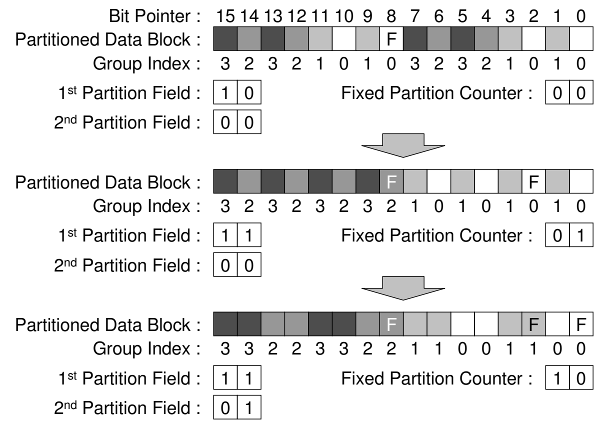
To summarize, the partition technique of SAFER identifies the two fail positions using their $\lceil log_2 n \rceil$ bit pointers. An XOR operation on the two fail pointers determines a bitwise difference vector between the two fail pointers. Finally, the technique selects a bit position with a value 1 from the difference vector, and resets all the other bits to 0. For example, if the selected bit position is the k^{th} LSB in the difference vector, the partition technique splits the n bit data block into two groups according to the k^{th} LSB of the pointer for each bit inside the block. Thus, $\lceil log_2 n \rceil$ group patterns exist and only $\lceil log_2(\lceil log_2 n \rceil) \rceil$ additional bits are needed to identify how a block is partitioned.

Furthermore, for blocks with two fail bits, the partitions have to be fixed in order to ensure that the fail bits are in different groups. Therefore, one additional bit is required to indicate whether a partition is fixed, or not. Thus, the total storage overhead for a n bit data block is $(1 + \lceil log_2(\lceil log_2 n \rceil) \rceil)$ bits.

As shown in the above example, the partition technique of SAFER for DEC is successfully able to partition the data block such that the two fail bits are not in the same group; thereby, enabling the use of SEC per group.


Data Block :
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Bit Pointer :
1st LSB 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
2nd LSB 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
3rd LSB 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
4th LSB 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
Partitioned Data Block :
Group Index : 3 2 3 2 1 0 1 0 3 2 3 2 1 0 1 0
1st Partition Field : 1 0
2nd Partition Field : 0 0
Fixed Partition Counter : 0 0
Additional bits to describe partitions

(a) Initial State


Bit Pointer : 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Partitioned Data Block :
F
Group Index : 3 2 3 2 1 0 1 0 3 2 3 2 1 0 1 0
1st Partition Field : 1 0
Fixed Partition Counter : 0 0
2nd Partition Field : 0 0
Partitioned Data Block :
F
F
Group Index : 3 2 3 2 3 2 3 2 1 0 1 0 1 0 1 0
1st Partition Field : 1 1
Fixed Partition Counter : 0 1
2nd Partition Field : 0 0
Partitioned Data Block :
F
F
F
Group Index : 3 3 2 2 3 3 2 2 1 1 0 0 1 1 0 0
1st Partition Field : 1 1
Fixed Partition Counter : 1 0
2nd Partition Field : 0 1

(b) Dynamic Partition

Figure 2: An Example of Four-Group Partition

3.2 Partition Technique of SAFER for Multi-Bit Error Correction

To be able to handle more than two bit fails in a data block, the partition technique is extended to dynamically partition the data block into multiple (> 2) groups by selecting multiple bits in the difference vector. We describe the extensions of the partition technique to handle multi-bit errors using an example shown in Figure 2.

Figure 2 shows an example data block of 16 bits with four fail bits to be partitioned into four groups, which are depicted with four different gray-levels. The partition technique associates a group index for each bit in the data block using two bits from the bit pointer. When a data block is composed of 16 bits, the bit pointer is ($log_2 16 = 4$) bits, which implies that there are $C_2^4 = 6$ possible ways to choose two bits out of them. Based on the fail bit locations, one of these six possible ways is chosen to determine the four groups.

In Figure 2(a), the initial partition arbitrarily uses the third and the first LSBs. For each data bit, the concatenation of these two bits in its bit pointer represents its group index. For example, the 12^{th} data bit has a bit pointer of "1100" and concatenating the third and the first LSBs results in a group index of "10"(2).

The "partition field" is extended to record which bit positions are used for partitioning the data. In this example, the two partition fields indicate that the third and the first LSBs are used from the bit pointer. The "partition fix" field is extended to a counter, "fixed partition counter", to keep track of the number of partitions that are fixed. In Figure 2(a), there are no fails, hence the value of the fixed partition counter is zero.

Figure 2(b) shows how a partition can be changed dynamically to account for a new fail bit. If the first fail bit occurs at bit position 8, the initial partition is still valid because none of groups has more than one fail bit. Now, if the second fail occurs at bit position 2, there are two fail bits in group 0. Thus, a new partition should be derived so that the two fail bits are in different groups. Using the partition technique described in Section 3.1, the difference vector of the two fails is ($1000 \oplus 0010 = 1010$), which implies that the second and the fourth LSBs are candidates for the first partition field. Correspondingly, the first partition field is set to "11". The fixed partition counter increases by one to account for fixing the first partition field. After this partition, groups 0 and 2 each have one fail bit. Note that even if the second fail bit was not located in group 0, the fixed partition counter would have to be incremented although the first partition field does not need to be changed.

At this point, if a third fail happens, the second partition field should be fixed with a proper value. If the third bit fails in position 0, the current partition has two fail bits in group 0. Applying the same partition technique as above, the difference vector of the two fails is ($0010 \oplus 0000 = 0010$), which implies that the second LSB position is the candidate for the second partition field. After this re-partitioning, groups 0, 1, and 2 each have one fail bit. Furthermore, the fixed partition counter is incremented and reaches its maximum value of two, making it impossible to re-partition further. Thus, in this example, we can recover from a fourth bit failure only if the failure occurs in bits belonging to group 3.

Based on the above discussion, it is clear that the hardware requirement is proportional to the number of groups required to partition the data to ensure one fail bit per group. For a n bit data block and a k group partition, the number of additional bits required is $\lceil log_2 k \rceil \times \lceil log_2 \lceil log_2 n \rceil \rceil + \lceil log_2(\lceil log_2 k \rceil + 1) \rceil$, where, $\lceil log_2 k \rceil$ is the number of partition fields, $\lceil log_2 \lceil log_2 n \rceil \rceil$ is the size of each partition field, and $\lceil log_2(\lceil log_2 k \rceil + 1) \rceil$ is the size of the fixed partition counter. For 512 bits of data block to be partitioned into 32 groups, additional 23 bits are required to represent the partition, which is still only 4.50% overhead compared to the data size.

3.3 Using Data Block Inversion for SAFER

The partitioning technique described in the previous section exploited the fact that stuck-at faults are permanent (not transient) to ensure that at most one stuck-at fault bit is present in each partition. In this section, we propose a recovery scheme by exploiting the fact that if a resistive memory's cell wears out resulting in a stuck-at fault, then it is still possible to read the cell content as the permanent stuck-at value. By exploiting this readability of failed cells, the recovery scheme reduces the number of additional bits required to recover data written to a stuck-at cell.

If a data block has only one fail bit and the data being written at the fail bit position is the opposite of the stuck-at value, then the data can be stored in an inverted form with a marked flip-bit. When reading the data, the original data can be recovered by inverting the stored data if the corresponding flip-bit is marked. The idea of inverting a data block is similar to bus-inverting coding [19] and Flip-N-Write [3]. However, our objective in inverting a data block is to recover a stuck-at fail while the bus-inverting coding [19] inverts a data block to reduce I/O power and the Flip-N-Write [3] utilizes it for removing redundant writes to PCM.

The proposed technique to invert the data can be used by SAFER only after verifying that the data write has failed to store the intended value. This write verification can be performed by reading the data written and comparing it with the original data. When the verification fails, the positions of fails and its stuck value can be revealed from the comparison result. Note that iterative write techniques which require a write verification phase are already needed for resistive memories using multi-level cells [14].

The proposed data inversion technique uses only one additional bit per partition to indicate that the data value has to be inverted prior to a read. However, the drawback is that the decision to invert and store the data can be made only after a first write fails the verification, resulting in two writes to store the data, thereby affecting the endurance of the cell.

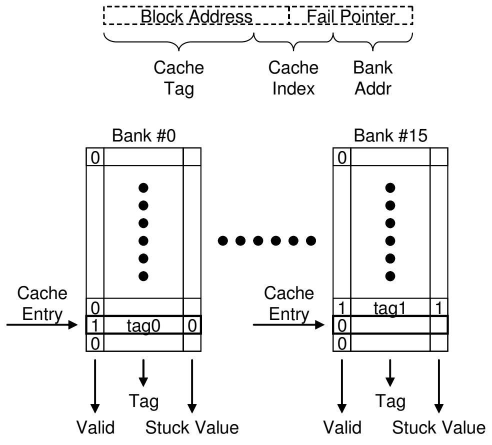


Figure 3: Fail Cache Organization

To alleviate this problem, we propose a relatively small direct-mapped cache called "fail cache", to keep track of data blocks with recent stuck-at fails. For these blocks recent fail positions and their stuck-at values are maintained in the cache. Figure 3 shows the fail cache organization that is composed of 16 banks. When storing new fail information, its block address and fail pointer are used to calculate the corresponding cache entry by separating into a cache tag, an index and a bank address. If new fail information is detected during the write verification phase, the fail position and the stuck-at value are known. Therefore, they can be stored with its tag portion in the corresponding cache entry. The tag, the cache index, and the bank address can also be calculated from its block address and the fail pointer. On every write request from the memory controller, all fail information for the corresponding n-bit data block should be extracted from the fail cache. To do so, the 16 banks are simultaneously accessed for $n/16$ iterations. For example, 32 iterations are required for a 512-bit data block. As a result, two n-bit vectors are generated – a fail indication vector and a stuck-at value vector. These two vectors for each write request can be exploited to avoid the additional write. If a fail indication vector indicates errors, the corresponding bits to be written are suitably inverted and stored according to their stuck-at values and partition information. Note that a read request to the same data block precedes a write request to eliminate redundant writes, and the partition information is collected during the read. Thus, if all fail information for a write data block is found in fail cache, the second write can be avoided. Also, since the preceding read can be used to gain enough time to access fail cache for $n/16$ iterations, the performance impact of the fail cache will be insignificant.

3.4 Putting It All Together

SAFER comprises two techniques, namely, the dynamic multi-group partition and the data block inversion. The dynamic multi-group partition ensures that each group includes at most one fail bit by partitioning the data into different groups. With each group now including at most one failed cell, the data block inversion scheme can be applied to recover from the stuck-at fault for that group. The total hardware bit budget of SAFER, to recover from a maximum of k failures, is $\lceil log_2 k \rceil \times \lceil log_2 \lceil log_2 n \rceil \rceil + \lceil log_2 (\lceil log_2 k \rceil + 1) \rceil + k$, where n is the size of a data block and k is the number of partitioned groups.

Another hardware overhead is bit manipulation logic for data block partition and data inversion. As our partition technique is based on the knowledge of fail positions, detecting a new fail position in a write verification phase is important. It can be implemented with simple combinational logic, an n-to-$\lceil log_2 n \rceil$ priority encoder for each partition group. If a priority encoder generates a valid fail pointer in the first verification phase, the corresponding group will be re-written in an inverted form. If a priority encoder still generates a valid fail pointer after the inversion write, it indicates the occurrence of a new fault in the corresponding group. Then, the data block is re-partitioned with the two fail pointers revealed at the two verification phases. That is, re-partition can be performed with the priority encoders and a simple FSM described in Figure 4. For both read and write data inversion, a partition decoder is required to select corresponding bits to be inverted.

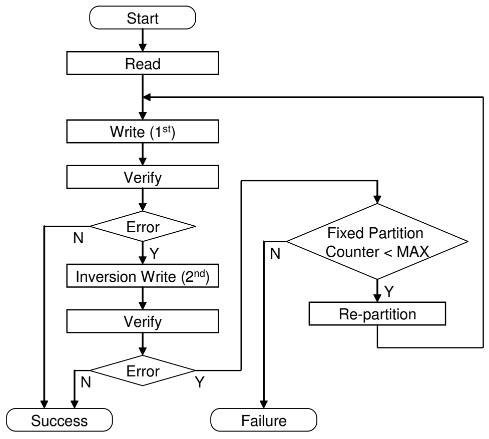


Figure 4: A Sequence of a Write Request in SAFER

Figure 5 shows an example of SAFER for a 16 bit data block and a four group partition. Additional six bits are required for the four-group partition and four flip bits are used to indicate whether the data in the corresponding groups is stored in an inverted form or not. Note that the six bits used to describe the partition are updated only when a new fail bit occurs. On the other hand, the four flip bits will be updated on every write that tries to store to the fail bit a value that is the opposite of the stuck-at value. In this example, fail bits are present in group 0, group 1, and group 2. Thus, the flip bits for those groups may be changed on every write. However, the flip bit for group 3 will still be zero until a new fail happens in group 3.

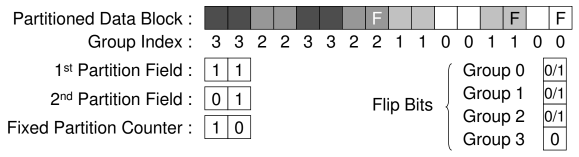

Partitioned Data Block :
F
F
F
Group Index : 3 3 2 2 3 3 2 2 1 1 0 0 1 1 0 0
1st Partition Field : 1 1
2nd Partition Field : 0 1
Fixed Partition Counter : 1 0
Flip Bits
Group 0 0/1
Group 1 0/1
Group 2 0/1
Group 3 0

Figure 5: An Example of SAFER

4. EFFICIENT IMPLEMENTATION OF SAFER

In this section, we address three key issues necessary for efficient implementation and use of SAFER, namely, where to place SAFER logic, what is the ideal data block size to maximize SAFER effectiveness, and how to limit the overhead of the "fail cache".

4.1 Where to place SAFER logic?

Fail recovery schemes are mainly used to prolong the lifetime of resistive memory especially after fails occur, but they are not geared towards decreasing the number of writes to improve the lifetime. Hence, fail recovery schemes must be used in concert with other schemes that delay the occurrences of failures, such as redundant write reduction schemes [10, 15, 21, 22, 3] and wear-leveling schemes [22, 16, 18]. These schemes are typically implemented in the memory controller or in the memory chip itself. For example, the wear-leveling schemes maintain their own address translation layer to evenly wear out the entire memory space, and the *Security Refresh* [18] logic is located inside the memory chip to protect against malicious attacks. In order to use fail recovery schemes in conjunction with these other schemes, it is necessary that they be embedded in the memory chips. Thus, we propose to locate the SAFER logic inside the memory chip.

4.2 Ideal Data Size for SAFER Effectiveness

SAFER dynamically partitions a data block into multiple groups according to fail locations and supports one bit correction for each group. Therefore, the larger the data block, the more efficient the fail recovery. For example, a double error correction per 16 bytes is more efficient than a single error correction per eight bytes. Similarly, four bit error correction per 32 bytes is more efficient than the two bit error correction. However, the upper bound of the size of a data block will be decided by the memory chip design, which is optimized to increase the density of the memory cell.

Figure 6 shows an example of a typical 4Gb 8 bank DDR3 DRAM architecture which is highly optimized for density. We expect the new resistive memory architecture to be similar to that of the DRAM because of the density issue. In the example, each bank is composed of 2048 sub-arrays whose size is 512×512 bits [4, 13, 2].

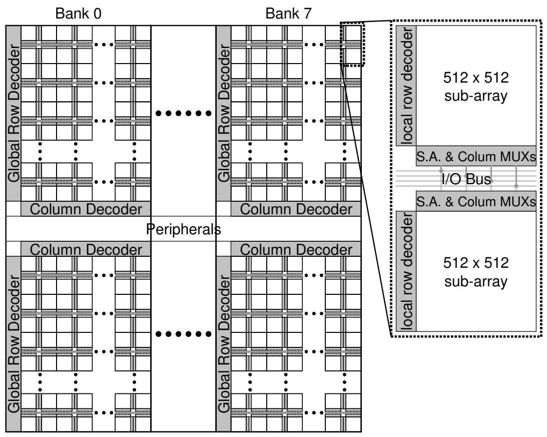


Figure 6: DRAM Architecture

Here, the column decoder generates column selection signals to sub-arrays, and pass-transistors, which act as column multiplexers, are located near by each sub-array. This is important so as to minimize area for long wires from the sense amplifiers to interface pe-

Table 1: SRAM Fail Cache Overhead Compared with a 8 Gbit PCM chip

Number of Entries	Tag Size (bits)	Entry Size (bits)	Cache Size (bits)	Area Overhead
1K	23	25	25.6K	0.01%
2K	22	24	49.2K	0.02%
4K	21	23	94.2K	0.04%
8K	20	22	0.18M	0.08%
16K	19	21	0.33M	0.15%
32K	18	20	0.63M	0.28%
64K	17	19	1.19M	0.53%
128K	16	18	2.25M	1.00%

ripherals by multiplexing them. Thus, the size of data bits that are transferred to interface peripherals at any time is equal to the minimum burst length that the chip supports. For instance, the DDR3 interface has a fixed burst length of eight. If the I/O data bus width is 16, 128 bits can reach to the peripherals.

To minimize the area overhead SAFER is best located in the peripherals, which implies that the size of a data block can be at most 128 bits in this case. However, historically the interface size continues on an upward trend from SDR to DDR, to DDR2, and to DDR3. Thus, we can safely assume that the size of data reaching the peripherals may be 512 bits in the near future. We evaluate the effectiveness of SAFER varying the block size from 64 to 512 bits in Section 5.

4.3 Area Overhead of Fail Cache

Since we decided to embed SAFER logic inside the memory chip, the fail cache should be located inside the memory chip with other peripherals. Fortunately, a PCM process is CMOS compatible, and it poses no process technology hurdles to implementing an SRAM cache. Hence, one of the major concerns is the area overhead of the "fail cache". According to ITRS projection [1], the cell sizes of SRAM and PCM, in 2024 will be $140F^2$ at 10 nm and $6F^2$ at 8 nm, respectively, implying that 36.46 times cell area difference may exist between SRAM and PCM. Table 1 shows the area overheads of a direct-mapped SRAM "fail cache" considering the 36.46 times cell area difference. For example, if we assume an 8Gbit PCM chip with a fail cache with 128K entries, in which each entry is composed of 16 bits of tag, a valid bit and a stuck-at value, then the total size of the cache is 2.25M bits which is only about 1.00% area overhead relative to the 8Gbit PCM. In Section 6, we show the effectiveness of the "fail cache" varying the number of entries from 1K to 128K.

5. METHODOLOGY

In this section, we present the methodology for evaluating SAFER and for comparing it against two existing techniques, namely the ideal *Hamming Coding* [5] and the *ECP* [17] technique. We compare against *Hamming Coding* because it represents a theoretical limit of memory lifetime for existing ECC schemes designed to correct transient errors. The number of bits required for the Hamming Coding implementation is provided by the Hamming Bound: $l \leq n - \lceil log_2 \Sigma_{k=0}^{t} C_k^n \rceil$, where l is the size of data, n is the size of the hamming code including meta-data for correction, and t is the number of correctable bits [20]. For example, a 512 bit data block needs 58 additional bits to be able to correct eight fails. Again, these 58 bits may serve only as a lower bound and a practical implementation may require more bits. In addition, Hamming Coding has a high toggle rate for the meta-data. Hence, an additional bit is needed to determine if the meta-data is valid. The indication bit also helps avoid cells for meta-data from failing earlier than data cells. In our evaluation, the Hamming Coding scheme is referred to as *IdealECC*.

Since our focus is to implement SAFER inside the memory chip, limiting the area overhead is important. We define area overhead as $\frac{\text{the size of meta-data}}{\text{the size of data}}$. For instance, the area overhead of the (72,64) hamming code is 12.5% ($= \frac{72-64}{64}$). Throughout this paper, we use the area overhead of the (72,64) hamming code as the upper bound for our evaluation and exclude all configurations of SAFER, *ECP* and *IdealECC* that exceed this area overhead.

Figure 7 shows the hardware overheads for the different configurations for *IdealECC*, *ECP*, and SAFER. The configuration names for each of the techniques include the maximum number of fails that can be recovered. The number above each bar in the graph shows the size of meta-data for the corresponding configuration. For example, for the 512 bit data block, ECP6 represents the ECP technique with six fail pointers that can recover up to six fails, and uses 61 bits for the meta-data; IdealECC8 represents the ideal eight bit Hamming Code correction technique which requires a minimum of 59 bits; and the SAFER32 can correct up to 32 fails with an additional 55 bits.

5.1 Experimental Setup

We use Monte Carlo simulations to evaluate SAFER and compare against *IdealECC* and *ECP*. Since PCM is the closest to mass production among resistive memories, our evaluations are based on ITRS projections for PCM endurance. We use the following assumptions for the Monte Carlo simulations:

1. We assume the lifetime of each memory cell to follow the normal distribution with a mean lifetime (μ) of 10^8 and without any correlation between neighboring cells [6]. Our experiments with different standard deviation (σ) values (10^7, $2 \cdot 10^7$, and $3 \cdot 10^7$) did not show significant variation in lifetime patterns. Hence, we use a standard deviation (σ) of 10^7 for our evaluations.

2. We assume a perfect wear-leveling scheme so as to focus only on the impact of the fail recovery scheme on the lifetime. The wear-leveling scheme evenly wears out the entire memory space at a block granularity equal to the line size of the last level cache as in the *Randomized Region-based Start-Gap* [16] and the *Security Refresh* [18]. We use 256 bytes for the last level cache line size, which implies that all the 256 byte memory blocks have the same number of writes because of the perfect wear-leveling scheme. Based on this, we measure the lifetime of one 256 byte data block.

3. A write request to memory is converted to a sequence of a read, a write, and a read request. The first read eliminates *silent writes* to memory by comparing the memory data read with the data to be written. We assume that 50% of the writes are *silent writes*. The second read verifies that the data written to memory matches the intended write data which allows us to recognize cell failures. SAFER requires another write with necessary bit inversions if cell failures are detected during write verification. However, a hit in the fail cache will avoid the second write because the bits are already suitably inverted to account for the cell failures based on the information stored in the fail cache.

4. We assume that four x16 memory chips compose a x64 DIMM memory module so that each chip can deliver 512 bits of data.

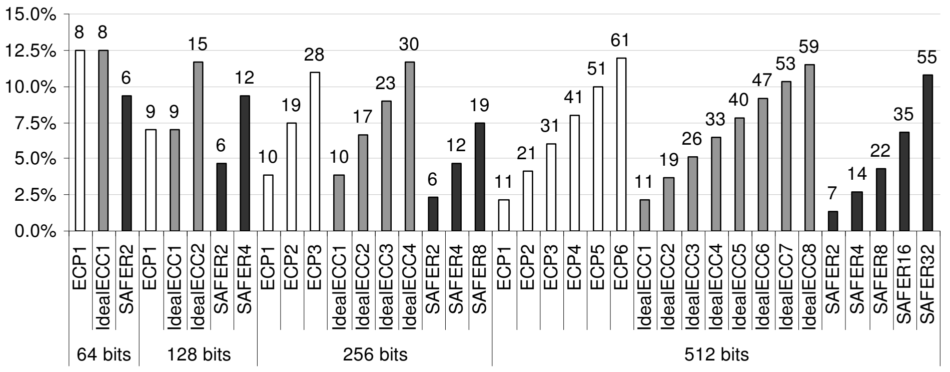


Figure 7: Hardware Overhead for Recovery Schemes

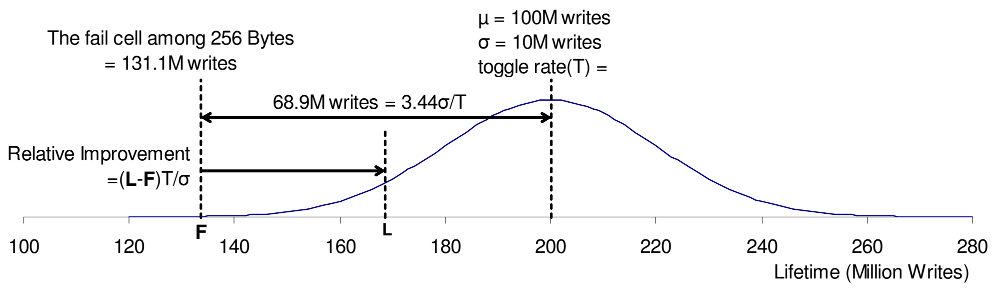


Figure 8: Definition of Lifetime Improvement

In the Monte Carlo simulation, each configuration is run 50000 times, and the average result is reported. For each run, our simulator allocates the array equivalent to the number of required cells including the 256 byte data block and its meta-data corresponding to each configuration. A random write endurance value according to the aforementioned normal distribution is assigned to each array element. For each write to a cell, we considered the toggling rate of the value to determine the available lifetime. Simulation continues until a given configuration cannot recover from a failure any longer. We take into consideration that all the meta-data do not have the same toggling rate. For example, the fail pointer in the ECP scheme and the partition fields in SAFER are updated only once when a new fail occurs. On the other hand, the meta-data for Hamming Coding (excluding the bit indicating the validity of the meta-data), the replacement cells in the ECP scheme, and the flip bits in SAFER are written with the same toggling rate (i.e., 0.5) as the data. For SAFER, the simulation also accounts for an additional write that is needed if the write verification detects a failure.

6. RESULTS

This section describes the simulation results focusing on the following figures of merit: lifetime improvement due to fail recovery, number of fails recovered for a given size of data block, and the cost of meta-bits for the observed lifetime improvement. Finally, we show the effectiveness of the fail cache in eliminating the additional writes and correspondingly improving the lifetime.

6.1 Lifetime Improvement

Our simulations assumed that the lifetime of memory cells follows a normal distribution $N(\mu, \sigma)$, where μ is 10^8 writes and σ is 10^7 writes. Furthermore, we assumed that each bit toggles with a probability $T = 0.5$. However, for reliable analysis, we present the lifetime improvement as a function of σ.

Figure 8 describes the method used to determine the relative lifetime improvement. In the example shown in Figure 8, the first fail shown as F occurs at 131.1 million writes. If SAFER were to increase the lifetime to L, then the relative lifetime improvement is calculated as $(L-F)T/\sigma$ to account for the dependence of the observed lifetime improvement on both σ and T. If SAFER were to increase the lifetime to the mean lifetime, then the relative lifetime improvement is $((2 \cdot 10^8 - 1.311 \cdot 10^8) \cdot 0.5/10^7) = 3.44$.

Figure 9 shows the relative lifetime improvement for each configuration with different data block sizes. For these results, SAFER does not use the fail cache thereby requiring the additional overhead of a second write if the write verification detects a failed cell. We observe that, even without the fail cache, SAFER improves the lifetime more than ECP for all the configurations. For a 512 bit data block size, SAFER32 increases lifetime by 21.6 million $(= 1.08 \cdot 10^7/0.5)$ writes, and ECP increases lifetime by only 21.1 million $(= 1.05 \cdot 10^7/0.5)$ writes while still using 10% more meta-data (Figure 7) than SAFER.

Also, each bar of the IdealECCn represents the lifetime improvement at the time of occurrence of the $(n+1)^{th}$ fail. For example, for a 512 bit data block, the lifetime improvement due to IdealECC2 is 12.8 $(= 0.64 \cdot 10^7/0.5)$ million writes when the third failure occurs.

6.2 Number of Fails Recovered

Figure 10 shows the average number of fails recovered per memory block for each configuration. It appears that ECP and IdealECC show linear increment in fails recovered with increase in the maximum number of recoverable fails. On the other hand, SAFER shows an exponential improvement. It is important to note that the maximum number of recoverable fails for SAFER increases expo-

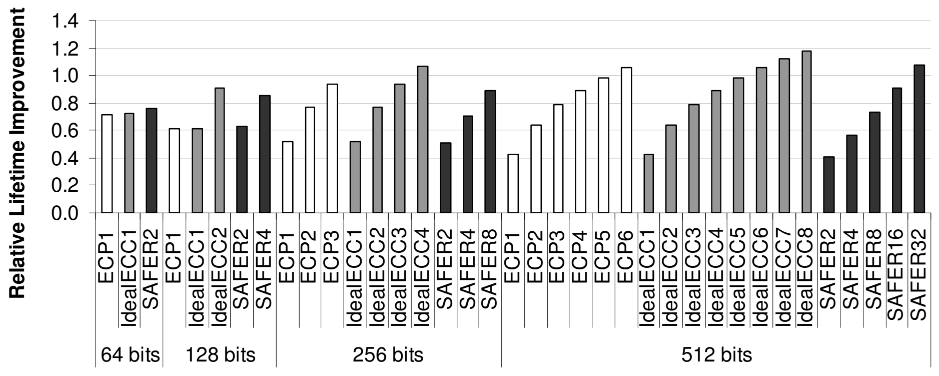


Figure 9: Relative Lifetime of 256B Memory Block

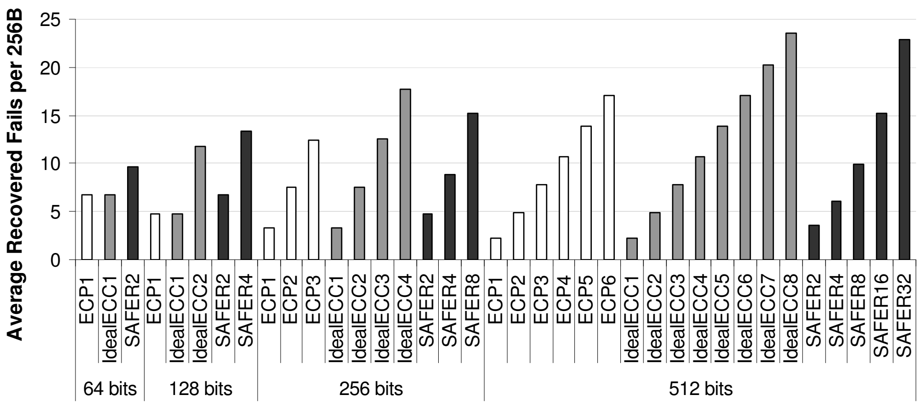


Figure 10: Fail Recovery in a 256B Memory Block

nentially as the number of partition fields increases linearly.

As shown in Figure 10, for a 512 bit data block, SAFER32 recovers from 22.94 fails whereas ECP6 recovers from only 17.08 fails. However, the relative improvement in lifetime with SAFER is only 2% better than the improvement with ECP. The key reason why the 34% improvement in the fail recovery of SAFER is not translated to larger improvement in lifetime (compared to ECP) is the additional write required by SAFER if the write verification phase identifies a failed cell when we do not use a fail cache. We show in Section 6.4 that using a fail cache with SAFER significantly removes the additional writes and shows gains in lifetime improvement even relative to IdealECC8.

6.3 Meta-bit overhead vs. Lifetime Improvement

Another important figure of merit of a recovery technique is the cost of meta-data for the observed lifetime improvement. Figure 11 shows the contribution of each meta-data bit to the overall lifetime improvement for a memory block size of 256 bytes. From Figure 11, we observe that, for a data block of 512 bits, SAFER32 has a 13.4% better utilization of the additional meta-data relative to ECP6.

6.4 SAFER with Fail Cache

So far, we have evaluated lifetime improvement and meta-bit efficiency of SAFER without fail cache. By using the fail cache, however, the lifetime can be extended even longer. Fail cache enables SAFER to avoid the additional write by providing information about the fail bits so that the data to be written can be suitably inverted. We use the miss rate of the fail cache as a measure of its effectiveness in reducing the additional write to the memory.

To determine the fail cache miss rate to enable n bits of data to recover from a maximum of k fails, we randomly set the fail bits in a memory of size 1GB, such that each n bits of memory had at most k failures. As soon as any n bits of data block has more than k fails, the fail insertion was terminated.

Using this set-up, we simulated 26 applications from SPEC2006 suite using the PIN instrumentation tool [11]. The following memory hierarchy was simulated: 32KB 8-way set-associative L1 data cache, 1MB 8-way set-associative unified L2 cache, and 8MB 8-way set-associative L3 DRAM cache, and finally a 1GB main memory. Out of 26 applications, we only used ten applications (Table 2), which has more than one million writebacks to the memory. In this set of simulations, we simulated five billion instructions.

Table 2: Applications with more than 1M writebacks to memory

Application	Number of Writebacks
410.bwaves	3.92M
429.mcf	8.17M
433.milc	7.72M
436.cactusADM	1.29M
437.leslie3d	4.75M
450.soplex	3.87M
458.sjeng	1.13M
459.GemsFDTD	9.37M
462.libquantum	7.62M
473.astar	2.45M

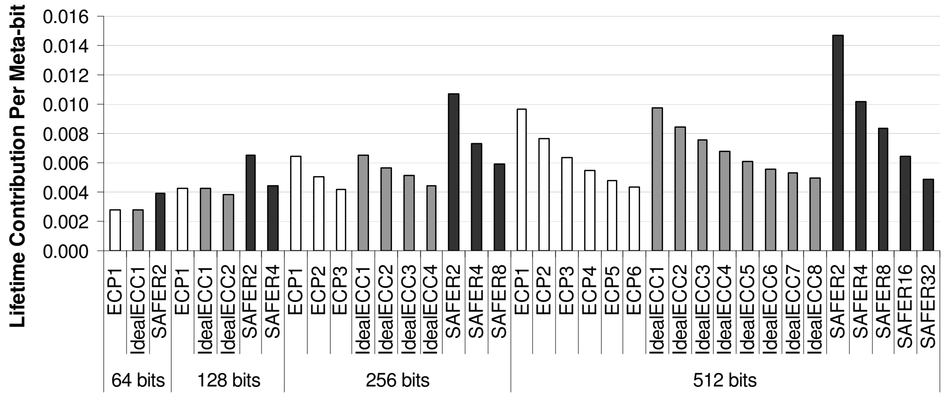
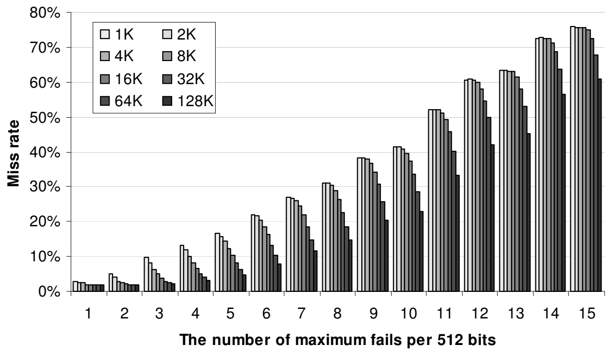
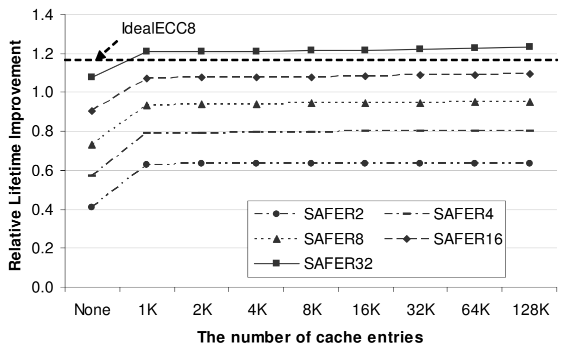


Figure 11: Meta-bit Contribution for Lifetime



(a) Fail Cache Miss Rate



(b) Relative Lifetime Improvement

Figure 12: SAFER with Fail Cache

The geometric mean miss rate of the above applications are shown in Figure 12(a) for different cache sizes for different maximum recoverable fails in a 512 bit data block. Note that different bars represent fail caches with different numbers of entries. From Figure 12(a), we observe that cache miss rate not only increases as we decrease the cache size, but also increases substantially as we increase the number of maximum recoverable fails. However, as the number of recoverable fails increase, the contribution to lifetime improvement due to each additional bit recovered continues to decrease. For example, from Figure 9, we observe that, for 512 bit data block, IdealECC2 achieves 54.6% of the relative lifetime improvement of IdealEEC8 by correcting up to only two errors. From Figure 12(a), we observe that, to correct up to two errors, the fail cache miss rate is only 5%.

Figure 12(b) depicts the relative lifetime improvement when we use a fail cache. Based on the above discussion, we observe that even a small fail cache with 1K entries has comparable lifetime improvement as a 128K entries cache. Furthermore, we observe that SAFER32 has better lifetime improvement relative to even IdealECC8 with just a fail cache of 1K entries.

7. CONCLUSION

Existing ECC mechanisms are geared towards correcting transient errors in DRAM memories and are not suitable to correct permanent stuck-at faults. Permanent stuck-at faults increase due to wear-out as the cells continue to age. The aging rate is particularly severe for several emerging non-volatile memory technologies. Furthermore, with process technology scaling, the lifetime variation of the cells increase, which leads to early multiple cell failures. We proposed and evaluated SAFER, a stuck-at fault error recovery technique for memories, which efficiently recovers from multiple stuck-at faults and which works in conjunction with existing wear-leveling techniques.

SAFER handles the growing stuck-at-fault errors by dynamically partitioning a data block into multiple groups and by ensuring that each group has at most one failed cell. SAFER reduces hardware overhead by exploiting the property that failed cells with a stuck-at value are still readable and uses the failed cell to continue to store data. Our evaluation based on phase-change memories shows that SAFER has 11.91% and 11.52% better hardware efficiency relative to ECP and ideal hamming coding schemes, respectively. Furthermore, SAFER achieves 14.75% and 3.07% better lifetime improvement relative to ECP and ideal hamming coding scheme, respectively.

Acknowledgment

This research is supported in part by an NSF grant CCF-0811738 and NSF CAREER Award CNS-0644096.

8. REFERENCES

[1] International Technology Roadmap for Semiconductors, Emerging Research Devices, 2009.

[2] R. J. Baker. *CMOS: Circuit Design, Layout, and Simulation*. Wiley-IEEE Press, 2007.

[3] S. Cho and H. Lee. Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance. In *Proceedings of the International Symposium on Microarchitecture*, 2009.

[4] V. Cuppu, B. Jacob, B. Davis, and T. Mudge. A performance comparison of ciontemporary DRAM architectures. In *Proceedings of the International Symposium on Computer Architecture*, 1999.

[5] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[6] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda. Dynamically Replicated Memory: Building Reliable Systems from Nanoscale Resistive Memories. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–14, 2010.

[7] S. Kang, WY Cho, B.H. Cho, K.J. Lee, C.S. Lee, H.R. Oh, B.G. Choi, Q. Wang, H.J. Kim, M.H. Park, et al. A 0.1-μm 1.8-V 256-Mb Phase-Change Random Access Memory (PRAM) with 66-MHz Synchronous Burst-Read Operation. *IEEE Journal of Solid-State Circuits*, 42(1):210–218, 2007.

[8] K. Kim and S.J. Ahn. Reliability investigations for manufacturable high density PRAM. In *Proceedings of the 2005 IEEE International Reliability Physics Symposium*, pages 157–162, 2005.

[9] K. Kim et al. Technology for sub-50 nm DRAM and NAND flash manufacturing. *IEDM Tech. Dig*, pages 323–326, 2005.

[10] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of the International Symposium on Computer Architecture*, 2009.

[11] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 190–200, 2005.

[12] F. Matsuoka and F. Masuoka. Numerical analysis of alpha-particle-induced soft errors in floating channel type surrounding gate transistor (FC-SGT) DRAM cell. *IEEE Transactions on Electron Devices*, 50(7):1638–1644, 2003.

[13] Y. Moon, Y.-H. Cho, H.-B. Lee, B.-H. Jeong, S.-H. Hyun, B.-C. Kim, I.-C. Jeong, S.-Y. Seo, J.-H. Shin, S.-W. Choi, H.-S. Song, J.-H. Choi, K.-H. Kyung, Y.-H. Jun, and K. Kim. 1.2V 1.6Gb/s 56nm 6F2 4Gb DDR3 SDRAM with Hybrid-I/O Sense Amplifier and Segmented Sub-Array Architecture. In *Proceedings of the 2009 IEEE International Solid-State Circuits Conference*, 2009.

[14] T. Nirschl, J. B. Phipp, T. D. Happ, G. W. Burr, B. Rajendran, M. H. Lee, A. Schrott, M. Yang, M. Breitwisch, C. F. Chen, et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *IEEE International Electron Devices Meeting*, pages 461–464, 2007.

[15] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System using Phase-change Memory Technology. In *Proceedings of the 36th annual International Symposium on Computer Architecture*, pages 24–33, 2009.

[16] M. K. Qureshi, J. Karidis, M. Fraceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing Lifetime and Security of Phase Change Memories via Start-Gap Wear Leveling. In *Proceedings of the International Symposium on Microarchitecture*, 2009.

[17] S. Schechter, G. H. Loh, K. Strauss, and D. Burger. Use ECP, not ECC, for Hard Failures in Resistive Memories. In *Proceedings of the International Symposium on Computer Architecture*, 2010.

[18] N. H. Seong, D. H. Woo, and H.-H. S. Lee. Security Refresh: Prevent Malicious Wear-out and Increase Durability for Phase-Change Memory with Dynamically Randomized Address Mapping. In *Proceedings of the 37th annual International Symposium on Computer Architecture*, pages 383–394, 2010.

[19] M. R. Stan and W. P. Burleson. Bus-invert coding for low-power I/O. *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 3(1):49–58, 1995.

[20] N. Wax. On Upper Bounds for Error Detecting and Error Correcting Codes of Finite Length. *Information Theory, IRE Transactions on*, 5(4):168–174, 1959.

[21] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *Proceeding of IEEE International Symposium on Circuit and Systems*, 2007.

[22] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the International Symposium on Computer Architecture*, 2009.