Aegis: Partitioning Data Block for Efficient Recovery of Stuck-at-Faults in Phase Change Memory

Jie Fan[†] fanj11@mails.tsinghua.edu.cn
Song Jiang[‡] sjiang@eng.wayne.edu
Jiwu Shu[†*] shujw@tsinghua.edu.cn
Youhui Zhang[†] zyh02@tsinghua.edu.cn
Weimin Zhen[†] zwm-dcs@tsinghua.edu.cn

[†]Department of Computer Science and Technology
Tsinghua University
Beijing, China

[†]Tsinghua National Laboratory for Information Science and Technology
Beijing, China

[‡]Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI, USA

ABSTRACT

While Phase Change Memory (PCM) holds a great promise as a complement or even replacement of DRAM-based memory and flash-based storage, it must effectively overcome its limit on write endurance to be a reliable device for an extended period of intensive use. The limited write endurance can lead to permanent stuck-at faults after a certain number of writes, which causes some memory cells permanently stuck at either '0' or '1'. State-of-the-art solutions apply a bit inversion technique on selected bit groups of a data block after its partitioning. The effectiveness of this approach hinges on how a data block is partitioned into bit groups. While all existing solutions can separate faults into different groups for error correction, they are inadequate on three fundamental capabilities desired for any partition scheme. First, it can maximize probability of successfully re-partitioning a block so that two faults currently in the same group are placed into two new groups. Second, it can partition a block into a small number of groups for space efficiency. Third, it should spread out faults across the groups as uniformly as possible, so that more faults can be accommodated within the same number of groups. A recovery solution with these capabilities can provide strong fault tolerance with minimal overhead.

We propose *Aegis*, a recovery solution with a systematical partition scheme using fewer groups to accommodate more faults compared with state-of-the-art schemes. The uniqueness of Aegis's partition scheme lies on its guarantee that any two bits in the same group will not be in the same group after a re-partition. Empowered by the partition scheme, Aegis can recover significantly more faults with reduced space overhead relative to state-of-the-art solutions.

[*]Corresponding author: Jiwu Shu (shujw@tsinghua.edu.cn).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
MICRO-46, December 07 - 11 2013, Davis, CA, USA
Copyright 2013 ACM 978-1-4503-2638-4/13/12 ...$15.00
http://dx.doi.org/10.1145/2540708.2540745.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Primary Memory*; C.4 [**Performance of Systems**]: Fault Tolerance

Keywords

Phase-change Memory, Reliability, Stuck-at Faults, Partition Scheme, and Cartesian Plane.

1. INTRODUCTION

Resistive memories have drawn great attention recently as the scaling of DRAM technology to smaller feature sizes (beyond 30 nm) becomes increasingly difficult [1, 2]. With higher scalability and being non-volatile, they hold great promises to complement or even replace DRAM as main memory or flashes as storage device. Among a number of resistive memories currently available, including phase-change memories (PCM), spin-torque-transfer magnetoresistive memory (STT-MRAM), ferroelectric memory (FRAM), and memristors (RRAM), PCM is the most promising technology for volume production [3, 4] and has seen the most research efforts [5, 8, 10, 13, 14, 16]. One of most challenging constraints on the device is its limited endurance. That is, after a limited number of writes on a memory cell (on average about 10^7 - 10^8), the cell is stuck at one of its two states representing either 0 or 1. Different from transient errors that are induced by alpha particle and usually found in DRAM, the so-called stuck-at-fault hard error is permanent and is PCM's major type of errors. When a cell has a stuck-at fault, its stuck-at value is still readable but cannot be changed.

1.1 Addressing the Issue on PCM's Stuck-at Faults

Being a major barrier towards PCM's wide adoption, stuck-at faults have been studied extensively. There are a number of solutions proposed to address this specific reliability issue. Some of the solutions rely on the operating system (OS) to hide the faults. For example, a solution can be to simply exclude memory pages containing faulty bits from being allocated. However, by doing this usable memory can be quickly depleted once there are memory bits approaching their lifetime limits and starting to fail. This issue is particularly severe because of (1) existence of early cell failures due to lifetime variation, and (2) wide-spread fault occurrences due to lack of spatial locality in the cell failures. Though dynamical pairing scheme [7] can recycle faulty pages by using a pair of pages with error bits at different offsets in the allocation of one page, it does not support wear-leveling, a technique essential for high reliability and security [12, 15]. For this reason, it is critical to build an error protection mechanism within the PCM chip for individual data blocks as the first-line of defense.

At the chip level error protection is applied on individual data blocks whose size is usually smaller than page size defined by OS. There are two general approaches for the protection. The first approach is to directly record the address of each faulty bit (a pointer) within a protected data block and use a replacement bit to store data on behalf of the faulty one. This approach is represented by the *ECP* (Error Correcting Pointers) scheme [14]. In *ECP*, a pointer and its corresponding replacement bit compose a correction entry, and each entry corresponds to one correctable fault in a block. With the sizable entries used in the approach, the number of correctable faults can be small under limited space budget.

The second approach is based on the fact that stuck-at values are still readable. In this approach a block is partitioned into a number of groups. If there is only one faulty cell in each group, the cell can be still used for storing data. When the cell's stuck-at value is opposite to the value being written into the cell, data in the group can be stored in an inverted form with a flag bit indicating that an inversion has been conducted. In this approach, it is groups, rather than bits as in the first approach, that are identified for correction operation. Thus, the space overhead for bookkeeping can be smaller, or with the similar amount of space cost more faults can be tolerated. This approach is represented by the *SAFER* (Stuck-At-Fault Error Recovery) scheme [16]. While the advantage of the partition-and-inversion approach is apparent, effective exploitation of its full potential relies on the efficacy of partitioning, or how bit cells in a data block are partitioned into different groups so that the error(s) in a group can be corrected with data inversion.

1.2 Partitioning Data Blocks for Inversion-based Correction

For a partition-based scheme to be functional, it must minimally meet two requirements on how cells in a data block are partitioned into groups. First, for any existing faults in a block to be tolerated with any new data to be written, at most one fault is allowed in a group. Second, to accommodate a new fault that occurs in a group already containing a fault, the data block must be repartitioned to separate them so that no group has more than one fault. The *SAFER* scheme meets these requirements by selecting one or multiple bits from the offset address of a bit within a block to form a partition vector[1]. Under the partition configuration defined by a partition vector, any two bits in a block whose partition vector values (derived from their respective offset addresses) are not the same will be in different groups (See Figure 1(a)). When a new fault occurs at a location with the same vector value as an existing fault (or the two faults collide in a group), *SAFER* expands the vector by adding a new bit at which the two faults' addresses are different (See Figure 1(b)). In this way, the new fault has a unique vector value and is exclusively in a group. While *SAFER* meets the minimal requirements, it does not have some highly desirable properties for partitioning with high space efficiency and fault tolerance capability.

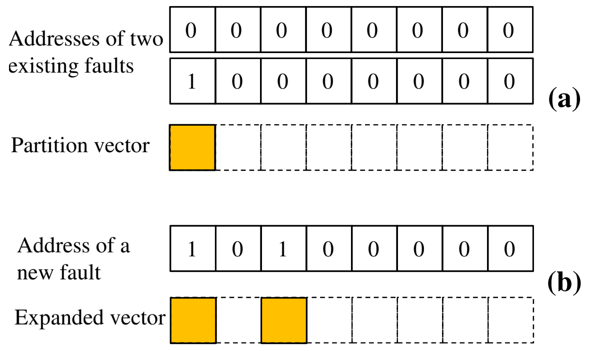

Figure 1: Illustrating how bit positions in the 8-bit offset address in a 256-bit data block are selected to form a partition vector. In (a) the first bit position is selected as the partition vector when the addresses of two existing faults differ at this position. When a new fault occurs with its vector value ('1') equal to that for one of the existing faults, which means it collides with the existing fault in a group, the vector has to be expanded (shown in (b)) so that all three faults have different vector values, which are 00, 10, and 11.

First, there should be a sufficient number of candidate partition configurations to accommodate many faults, as a new configuration is demanded whenever two faults collide in a group. In *SAFER*, each partition vector represents a candidate configuration. Though theoretically there are 2^n configurations for a data block of 2^n bits, the actual number of usable configurations for a block is only $n+1$. In *SAFER*, an additional configuration is generated by adding a bit into the current vector. In the worst scenario where every new fault generates a fault collision, only $n+1$ faults can be tolerated and then any additional fault will fail the entire block. If a new partition vector could be generated by allowing removal or replacement of bits in the current vector, there is no assurance that exiting collision can be eliminated or new collisions are not introduced. While exhaustive search over the space of 2^n candidate configurations is way too expensive to be feasible, *SAFER* has to accept relatively small set of configurations from which to determine a collision-free partition.

Second, the number of groups should be small, as it determines the space overhead. For *SAFER* a major overhead space is used for bookkeeping which groups' data are inverted. With increasing number of faults, the number of groups is increased exponentially (each additional bit in the partition vector doubles the group count.). When the partition vector has m bits for a 2^n-bit data block ($m \leq n$), there are 2^m groups and 2^m bits are needed for the bookkeeping, each indicating if the data in a group are inverted. When $m = n$, the overhead space is the same as that for storing

[1]Partition vector is coined in this paper to concisely and accurately describe *SAFER*'s fundamental design principle.

real data. Therefore, in practice the maximum number of groups has to be limited and the number of recoverable faults (m) has to be reduced. Note that in practice this overhead has to be budgeted according to the maximum number of groups in each data block.

Third, faults should be spread out across the groups as uniformly as possible. We intend to use smaller number of groups to accommodate more faults. In the meantime, two faults are not allowed to collide in the same group. Therefore, groups have to be well utilized for holding faults. In other words, a partition scheme should scatter bits originally in the same group into different groups and prevent faults from meeting in the same group in a re-partition. *SAFER* does ensure that faults currently in the same group will be separated into different groups in its re-partition. However, it does not explicitly take effort to shuffle bits across the groups and to increase the chance for bits, including faulty bits, to move among all the groups.

1.3 Our Solution: an Efficient Partition Scheme Tolerating More Faults

With the aforementioned desirable properties in mind, we design a partition scheme that (1) has a larger set of candidate partition configurations for resolving fault collision to tolerate more faults, (2) has smaller number of groups in each configuration to reduce space overhead, and (3) intensively shuffles the bits among the groups attempting to uniformly distribute faults across different groups. Using the proposed partition scheme in which a set of candidate partition configurations are defined, we guarantee that *any two bits in the same group of a data block in a partition configuration will not be in the same group in a different partition configuration from the set.* With this assurance, the number of configurations causing collisions among a set of faults is well bounded. As long as the candidate set is sufficiently large, we can always find a configuration accommodating the faults without a collision. Compared with exponentially increasing number of groups with the increase of tolerable faults in *SAFER*, the proposed scheme increases the group count at a polynomial rate. A much smaller group count not only saves space for bookkeeping as explained, but also makes feasible an implementation that considers distinction between the stuck-at values and the data being written. Such an implementation can further substantially improve PCM's fault tolerance (see Section 2.4).

We name the error recovery scheme based on the proposed partition scheme as well as its implementation design as *Aegis*. Our evaluation shows that *Aegis* can use substantially reduced overhead space to tolerate much more faults.

2. THE DESIGN OF *AEGIS*

The foundation of *Aegis* is its partition scheme that can efficiently distribute faults into different groups. In this section we first describe the partition scheme as well as its theoretic basis. Next we describe the *Aegis* error recovery scheme based on its partition scheme as well as its implementation. Depending on the hardware affordability, the scheme may know whether a stuck-at value at a fail cell equals to the data being written into the cell. Assuming this knowledge is available, we enhance *Aegis* by possibly allowing multiple faults in one group to further improve its fault tolerance capability. In the last part of this section, we describe design of *Aegis*'s variants with the enhancement.

2.1 The Partition Scheme for *Aegis*

A partition scheme defines a set of partition configurations, each specifying how bits in a data block are distributed into different groups so that not any two faults are in the same group. When a new fault is detected in a group already containing a fault in the current partition configuration, a re-partition, or selecting another partition configuration in the set, is required to resolve the fault collision. That is, the minimal requirement on any partition scheme is to ensure separation of two faults in a group into two different groups by switching to another partition configuration. However, *Aegis*'s partition scheme is more powerful. It separates not only two faulty bits but also *any* two bits in a group into two different groups after switching to a different partition configuration. This helps to evenly spread faults in a block across different groups and promotes wear leveling with each block.

Aegis's partition scheme is inspired by the observation that on a Cartesian plane any two different points on a line uniquely determine slope of the line. If we change the slope of the line, we can only keep at most one point on the original line to stay on a new line of a different slope. If we consider all points on a line as a group, a partition configuration is to use lines of a common slope to cover points on the plane so that each of them is on a unique line, or in a unique group. While a partition configuration corresponds to a particular slope, changing a configuration is simply to use lines of a different slope to make any two points that were original in the same group not be in the same group in the new configuration.

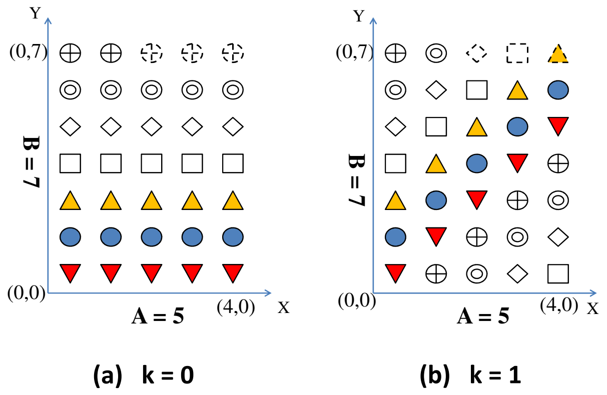

Figure 2: Illustrating how bits in a 32-bit data block are partitioned into 7 groups, each of 5 bits. For (a) the partition configuration adopts slope $k = 0$, and (b) represents a different configuration using slope $k = 1$. In total there are 7 configurations in the partition scheme defined by the $A \times B$ rectangle (5×7). The bits, each represented by a symbol, are mapped into a Cartesian plane. As the (5×7) rectangle contains three more positions than the 32 bits in the block, the three dotted symbols on the top right are unmapped. Different symbols are used to distinguish bits in different groups.

We arrange bits of an n-bit data block in a rectangle on the Cartesian plane. Coordinates of four corners of the rectangle are $(0, 0)$, $(A, 0)$, $(0, B)$, and (A, B), as illustrated in

Figure 2. A and B are positive integers and $A \leq B$. Furthermore, B must be a prime number. A bit at offset x in the block is mapped to (a, b) in the Cartesian plane, where $aA + b = x$. For a given B, we choose an A so that the rectangle is sized just large enough to accommodate n bits, or $A(B-1) < n \leq AB$. While the rectangle can be larger than N, there can be a small number of points unmapped. As long as an $A \times B$ rectangle is determined, a partition scheme is defined, including number of partition configurations, number of groups in each configuration, and group size. For this reason, we call the corresponding scheme an $A \times B$ *Aegis* scheme.

Theorem 1. For a point (a, b) in the above rectangle and a given integer k ($0 \leq k < B$), there exists a unique y ($0 \leq y < B$) so that $b = (ak + y)\%B$.

Proof. To satisfy $b = (ak+y)\%B$, there must be $ak+y = mB + b$ or $y = mB + b - ak$, where m is an integer. Apparently there is one and only one m that produces a y that belongs to $[0, B)$. Therefore, we have a unique y ($0 \leq y < B$) so that $b = (ak+y)\%B$. ■

According to the theorem, a point (a, b) in the rectangle can be uniquely represented as $(a, (ak+y)\%B)$. When $a = 0$, point $(0, y)$, which is on the Y axis, is called the anchor point for any points $(a, (ak+y)\%B)$ ($0 \leq a < A$). All these points are on a line whose slope is k ($0 \leq k < B$).

Aegis's partition scheme consists of B partition configurations. In the kth configuration ($0 \leq k < B$), there are B groups. Its yth group consists of all points $(a, (ak + y)\%B)$ ($0 \leq a < A$, $0 \leq y < B$) with their anchor point at $(0, y)$ and corresponding slope of k. Figure 2 shows how a 32-bit block is partitioned into 7 groups, each of 5 bits. Figure 2 (a) show the partitions using the first configuration (with slope $k = 0$) and Figure 2 (b) shows the partition using the second configuration (with slope $k = 1$). As shown in the figures, each slope value corresponds a partition configuration and each anchor point corresponds to a group in the partition configuration. Two points $((a_1, (a_1k_1 + y_1)\%B)$ and $(a_2, (a_2k_2 + y_2)\%B))$ are in the same group as long as $k_1 = k_2$ and $y_1 = y_2$.

Theorem 2. Under an $A \times B$ *Aegis* partition scheme, where $0 < A \leq B$ and B is a prime number, any two points that are in the same group in a partition configuration must not be in the same group again in a different partition configuration.

Proof. According to Theorem 1, any point must be in a group and only in one group. Any two points that are in the same group in a particular partition configuration can be represented as $(a_1, (a_1k+y)\%B)$ and $(a_2, (a_2k+y)\%B))$, where $0 \leq k < B$, $0 \leq y < B$, and $0 \leq a_1 < a_2 < A$. If a different partition configuration is used, or the slope is changed to k' ($k \neq k'$ and $0 \leq k' < B$), the two points should be represented as $(a_1, (a_1k'+y_1)\%B)$ and $(a_2, (a_2k'+y_2)\%B))$, where $0 \leq y_1 < B$ and $0 \leq y_2 < B$. As two points in the same group must have the same anchor point, we only need to prove $y_1 \neq y_2$ to show that the two points are not in the same group in the new partition configuration.

Because $(a_1k + y)\%B = (a_1k' + y_1)\%B$, we have $y_1 = a_1(k - k') + y + m_1B$ (m_1 is an integer). Similarly, we have $y_2 = a_2(k - k') + y + m_2B$ (m_2 is an integer). Now we have $y_1 - y_2 = (a_1 - a_2)(k - k') + (m_1 - m_2)B$. If $y_1 = y_2$ could be held, we would have $(a_1 - a_2)(k - k') = (m_2 - m_1)B$, and therefore $(a_1 - a_2)(k - k')$ is a multiple of B. Because $|(a_1 - a_2)| \in (0, A)$, $A \leq B$, and $|(k - k')| \in (0, B)$, neither $(a_1 - a_2)$ nor $(k - k')$ can be a multiple of B. Because B is a prime number, $(a_1 - a_2)(k - k')$ is not a multiple of B, which contradicts. Therefore, $y_1 \neq y_2$, and with a different slope k' the two points are not in the same group. ■

2.2 The *Aegis* Error Recovery Scheme

Aegis's partition scheme can separate any two faults in the same group into two different groups using a re-partition, or switching to a different partition configuration. However, it does not ensure that two faults that were originally in different groups are not placed into the same group after the re-partition. This seems to be a serious challenge on the application of the partition scheme as it is possible to take multiple re-partition trials before settling on a configuration without any fault collisions. Fortunately, the number of the trials is theoretically capped at a very small value and the cost of the re-partitions, if amortized over all writes, is negligible.

Assuming that a data block has f faults, which can generate C_2^f, or $f(f-1)/2$, different pairs of faults. When a pair of faults are in the same group in one partition configuration, they are not in the same group in any other configurations defined in the partition scheme. In the search of a collision-free partition configuration, after trial of each configuration at least one pair of faults are rendered impossible to collide with each other in the remaining configurations. That is, if the scheme has more than C_2^f configurations, or minimally $C_2^f + 1$ configurations, there exists at least one collision-free configuration in which every pair of faults are not in the same group. Therefore, at the time when a data block accumulates f faults at most $C_2^f + 1$ re-partitions have been performed. While tens of millions of writes have probably been performed at the time, the cost associated with the re-partitions amortized over the writes is negligible.

Aegis adopts the proposed partition scheme. In the meantime it uses the general partition-and-inversion framework proposed in the *SAFER* scheme [16]. The framework needs a verification read after each write, or reading data just written for comparing it with the original one, to detect any error(s). Note that in the design of the basic *Aegis* scheme, we do not assume it can afford the cost to remember where individual faults are and what their respective stuck-at values are. The only recorded information about faults is whether a group of bits had been inverted for masking error(s) in the group. This verification read operation should not be regarded as an added overhead for the partition-and-inversion approach, as it has been required for resistive memories [11].

For each data block, an $A \times B$ *Aegis* scheme maintains a counter, named as slope counter, recording current slope value k ($k = 0, 1, ..., B-1$) and an (inversion) bit vector whose bth bit indicating whether data in the bth group is inverted ($b = 0, 1, ..., B-1$). b is the identifier (ID) of the group. Initially the slope counter is 0 and the vector is reset. For a write in *Aegis*, a subsequent verification read will reveal faults whose stuck-at values are different from the written data (if any). For each of the faults, *Aegis* uses a pre-wired logic to derive ID of the group it belongs to (an example logic is illustrated in Figure 3). If more than one faults is found to have the same ID, there is a collision. *Aegis* increments the slope counter by one to reach a new partition configuration. Once again, *Aegis* examines the faults to see if there are collisions. Theoretically this re-partition step

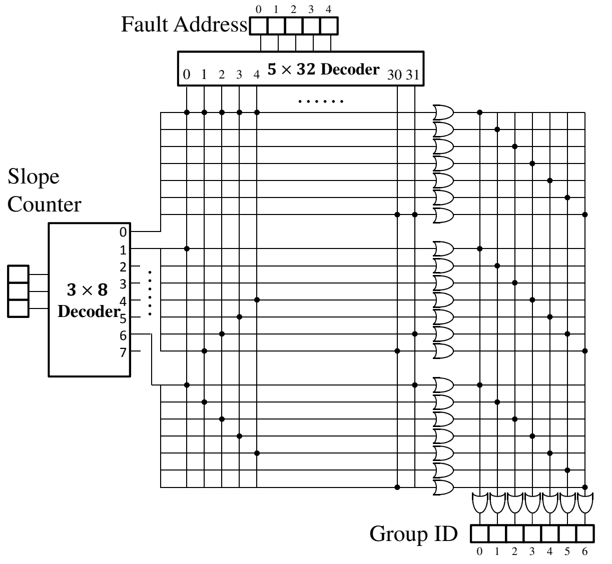

Figure 3: An implementation logic to know which group a fault at a given 5-bit address belongs to in a 32-bit block with a given 3-bit slope counter value. On the left of the circuit is a 49×32-bit ROM recording information on which bits are in the same group with a given slope value. On the right is a 49×7-bit ROM in which groups of the same ID are placed in the same column.

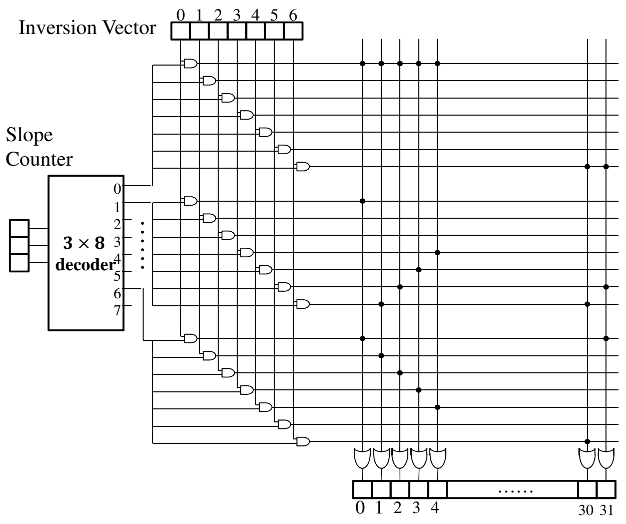

Figure 4: An implementation logic to know which bits of a 32-bit data block need to be written in their inverted form for a given slope counter value and a given 7-bit inversion vector. The array of AND gates on the left of the circuit generate signals, each corresponding to a combination of a slope and a group. On the right is a 49×32-bit ROM recording the relationship between the combination and the group's member bits.

might be repeated for a number of times. However, as we have explained the repetition is rare in practice and the cost is negligible. With a collision-free configuration, *Aegis* knows which group(s) have the detected fault(s). It sets the corresponding bit(s) in the inversion vector and inverts the bits that belong to the group(s). An example logic on how bits in the selected groups are identified for inversion is illustrated in Figure 4. Writing these inverted bits is also followed with a verification read. If this read reveals new fault(s), these fault(s) must collide with the faults detected before the inversion write, and therefore demand re-partition(s).

2.3 *Aegis*'s Fault Tolerance Capability and Space Overhead

A scheme's Fault Tolerance Capability (FTC) can be measured with two metrics, each from a different perspective. One is hard FTC, referring to number of faults a scheme guarantees to tolerate regardless of distribution of the faults in a data block and actual data being written. The other is soft FTC, referring to number of faults that can be tolerated when the faults located at certain addresses and a number of writes of certain values happen in a particular order over the time. As an example, for a 512-bit data block *SAFER* using a 5-bit partition vector, its hard FTC is 6, and its soft FTC can be any number between 6 and 32, depending on fault and write occurrence patterns. In this section we only analyze hard FTC as well as its associated space overhead, and leave discussions on soft FTC in a practical setup in Section 3.

For the $A \times B$ *Aegis* scheme, its hard FTC is the largest integer f_{max} that satisfies $C_2^{f_{max}} + 1 \leq B$. To support this hard FTC, for each n-bit data block *Aegis* needs a slope counter of $\lceil log_2 B \rceil$ bits and an inversion vector of B bits. Therefore the hardware cost associated with each data block is $\lceil log_2 B \rceil + B$ bits. If the required hard FTC f satisfies $C_2^f + 1 < B$, the hardware cost can be reduced to $\lceil log_2(C_2^f + 1) \rceil + B$. Table 1 summarizes number of bits required to minimally tolerate a given number of faults, or to support hard FTC, for each 512-bit block in ECP, SAFER, and Aegis (as well as two Aegis's variants to be described in Section 2.4.) The required space for ECP and SAFER is calculated with formulas provided in their respective papers [14, 16].

As shown in the table, *SAFER*'s cost rises at the fastest rate with the increase of hard FTC because its group count is increased exponentially to cover increasing number of faults. *ECP*'s cost increases linearly, as it uses a fixed-size pointer for each fault. However, by indicating errors with pointers, it may cap the soft FTC as low as its hard FTC, greatly limiting a scheme's error correction potential in practice. In contrast, at moderate and high FTCs *Aegis* provides a much smaller cost than either of the existent schemes as it consistently uses a small number of groups. As example, to tolerate 8 faults *SAFER* has to use 128 groups while *Aegis* only needs 31 groups. From another perspective, with a similar number of groups *Aegis* can tolerate more faults. For example, with 31 groups *Aegis* can tolerate 8 faults, or accommodate these faults each in a different group. In contrast, using 32 groups *SAFER* can only tolerate 6 faults. This is because *Aegis* is more powerful to distribute faults into different groups, as revealed in Theorem 2. *Aegis* is not designed for a PCM whose faults are capped at a very small count, as it provides minimally 23 groups for a 512-bit block. This is why its cost at small hard FTCs are higher. The cost can be reduced by directly recording IDs of bit-inverted groups.

Hard FTC	1	2	3	4	5	6	7	8	9	10
ECP	11	21	31	41	51	61	71	81	91	101
SAFER	1	7	14	22	35	55	91	159	292	552
N (*for SAFER*)	*1*	*2*	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
Aegis	23	24	25	26	27	27	28	34	43	53
$A \times B$ (for *Aegis*)	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*18 × 29*	*14 × 37*	*11 × 47*
Aegis-rw	23	24	25	26	26	27	27	28	28	34
Aegis-rw-p	1	8	9	15	15	21	21	27	27	32
$A \times B$ (*for Aegis-rw/Aegis-rw-p*)	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*23 × 23*	*18 × 29*

Table 1: The hardware cost, in terms of bit count, associated with each 512-bit block when different error recovery schemes (*ECP*, *SAFER*, and *Aegis*) are used to tolerate a given minimal number of faults (Hard FTC). The row about N lists specific *SAFER* schemes using different number of partition groups (N). The rows about $A \times B$ list specific *Aegis* schemes that are used to achieve respective hard FTCs. *Aegis-rw* and *Aegis-rw-p* are two *Aegis*'s variants. *Aegis-rw-p* for hard FTC of 1 is a special case where only one inversion bit is needed.

2.4 Enhancing *Aegis* with Known Faults

Comparing the stuck-at value and actual value being written, we can classify a stuck-at fault either as a stuck-at-Wrong fault (W fault in short), if the two values are different, or as a stuck-at-Right fault (R fault in short), if the two values are the same. A verification read following a write reveals W faults, and a verification read after an inverted write reveals R faults. If we are aware of the distinction among faults (R or W faults), a group can accommodate more than one fault as long as they are of the same type. For example, if there are two W faults and zero R fault in a group, we can recover the errors by inverting the bits in the group without re-partition. Therefore, a partition scheme only needs to make sure W faults are not mixed with R faults in the same group. By relaxing the requirement for tolerating faults, it is possible to enhance *Aegis*'s fault tolerance capability. However, the feasibility of the enhancement depends on whether it is affordable to obtain the knowledge about R and W faults.

One option to obtain the knowledge is to use double writes, one with the actual data and the other with the inverted data, of an entire data block to know all R and W faults. Accordingly, an enhanced *Aegis* can take advantage of this additional knowledge for more efficient partition and determining groups for inverted writing. However, this option is too expensive to be adopted. In this option all bits in a block have to be written twice. Actually in practice to reduce wear on the PCM, before a write a read is performed to determine which bits are different between the data being written and the one currently being stored, and only the different bits are actually written [18]. Even worse for the option, unlike indiscriminate-fault distribution, which is persistent until a new fault occurs, R/W fault distribution could change upon service of any write request as the involved data may change. Accordingly, the aforementioned double writes have to be performed during servicing every write request, making its latency too high and its induced wear too much.

A more practical option is to adopt the fail cache proposed as an addition to the *SAFER* scheme for reducing additional latency and wear due to inverted writes in groups with faults [16]. The fail cache is a directly-mapped cache in an SRAM recording locations of recently detected faults (their data block addresses and in-block offsets) and their stuck-at values. Before each write into a block, every bit in the block has to be checked in the cache to see if it is a fault. If all faults in a block (if any) are found in the cache, SAFER can avoid the additional write into the group(s) with fault(s). Depending on the write locality as well as affordability of the cache in terms of its hardware cost and relative times of reading from the cache and writing into the PCM, the fail cache is possible in future's PCM chips.

Assuming availability of the knowledge on R/W faults in a block before a write, *Aegis* can find a partition configuration without trials of slopes. To this end, for the $A \times B$ *Aegis* scheme we build an $n \times n \times \lceil log_2 B \rceil$ ROM for a PCM with n-bit blocks. According to Theorem 2, any two bits collide, or stay in the same group, in only one partition configuration, or on only one slope. The ROM records the unique slope on which any two bits collide. To know on which slope two given bits collide, we can use one bit's address as the column address and the other bit's address as row address to read the slope from the ROM. Suppose there are f_W and f_R W faults and R faults, respectively, in a block. *Aegis* picks each of f_W W faults and each of the f_R R faults to access the ROM and knows the set of slopes on which at least a W fault collides with a R fault. Any slope that is not in this set can be used as a collision-free partition configuration. As long as the *Aegis* scheme provides $f_W \times f_R + 1$ slopes, it is guaranteed that such a configuration exists. In a block with f ($f = f_W + f_R$) faults, $f_W \times f_R + 1$ is smaller than $C_2^f + 1$. So the enhanced *Aegis*, denoted as *Aegis-rw*, can provide a higher hard FTC with lower cost, especially when a block has a large number of faults. For example, for hard FTC of 10, *Aegis* needs 46 slopes while *Aegis-rw* needs only 26 slopes. Accordingly, with 34 bits *Aegis* provides a hard FTC of 8 while *Aegis-rw* provides a hard FTC of 10, as shown in Table 1. It is noted that if *Aegis-rw* and *Aegis* use the same ($A \times B$) as their formation, they are of the same space cost.

If the expected number of faults are well smaller than the number of groups, the inversion vector can be replaced with directly recorded group pointers, or IDs of inverted groups, much like what *ECP* does. In *Aegis-rw* only groups containing W faults are inverted and their IDs are recorded. However, a block containing f faults can have as much as f such groups, and *Aegis-rw* has to use f pointers to achieve a hard FTC of f. Fortunately we can enhance it to a scheme, named as *Aegis-rw-p*, using only $\lfloor f/2 \rfloor$ group pointers because (1) *Aegis-rw* knows all R faults and W faults; and (2) according to the pigeonhole principle we have either $f_W \leq \lfloor f/2 \rfloor$ or $f_R \leq \lfloor f/2 \rfloor$. If $f_W \leq \lfloor f/2 \rfloor$, groups with W faults are written in their inverted form and pointers to these W groups are recorded. If $f_R \leq \lfloor f/2 \rfloor$, groups without R faults are written in their inverted form and pointers to these R groups are recorded. To read data stored in the block, PCM first inverts the groups identified by the pointers, and then inverts the entire block. In addition to the pointers a bit is needed to distinguish the two cases. In both cases, *Aegis-rw-p* performs only one write if the fail cache reveals all faults. Otherwise, additional writes are needed. The minimal hardware cost for *Aegis-rw-p* to tolerate a given number of hard FTC is listed in Table 1. The $(A \times B)$ *Aegis-rw-p* scheme with p pointers needs $log_2(min(\lfloor f/2 \rfloor \times \lceil f/2 \rceil + 1, B)) + \lfloor f/2 \rfloor \times \lceil log_2 B \rceil + 2$ bits to protect each data block, which includes a bit indicating whether inversion of entire block is conducted and another bit indicating whether all pointers have been used. As shown in the table, using group pointers can further substantially reduce space overhead. However, use of fixed number of pointers can compromise reliability in terms of soft FTC. It is noted that as *Aegis-rw* and *Aegis-rw-p* are derived from *Aegis*, they are also specified by $(A \times B)$ used to define *Aegis* scheme.

3. PERFORMANCE EVALUATION

In this section we evaluate efficacy and cost-effectiveness of *Aegis* for tolerating PCM's stuck-at faults. In the evaluation, we compare *Aegis* with thee other error recovery schemes specifically designed for stuck-at faults, including *ECP*, *SAFER*, and *RDIS*. Among them *RDIS* is an error recovery scheme using the partition-and-inversion approach [10]. It recursively selects bits arranged on a two dimensional array into different sets to separate R faults from W faults, so that bit inversion can be applied to correct the faults. While the distinction of W and R faults is required in *RDIS* rather than as an option in *SAFER* and *Aegis*, we always supply it with a sufficiently large cache to provide information about any faulty cells in the experiments. In addition, we use *RDIS-3*, or the *RDIS* applying recursive partition for three times, to represent *RDIS*, as does in the RDIS paper [10]. For *ECP*, we evaluate its use of different numbers of pointers. The scheme using N pointers is denoted as *ECP*$\mathbf{N}$. For *SAFER*, we include both the scheme without using cache, denoted as *SAFER*$\mathbf{N}$ where N is the number of partition groups, and the scheme using a sufficiently large cache to provide fault information for any writes (or a cache without misses), denoted as *SAFER*$\mathbf{N}$-*cache*, in the evaluation. To be conservative, we always use *Aegis* without a cache, denoted as *Aegis*, in its comparison with previously proposed schemes, and only two *Aegis* variants, *Aegis-rw* and *Aegis-rw-p*, assume the existence of such a cache.

3.1 Experimental Setup

We use Monte Carlo simulations for the experimentation, whose setup is similar to the configuration commonly adopted in the evaluation of previously proposed schemes [10, 13, 14, 16]. For each PCM cell, we assign it with a lifetime in terms of number of writes before its failure. This lifetime follows the normal distribution with a mean lifetime of 10^8 and a 25% coefficient of variance. There is no correlation between neighboring cells on their fault occurrences.

There are two kinds of blocks in the simulation. One is the data block on which an error recovery scheme applies its protection, whose size is expected to be in the range between 128 bits and 512 bits, equal to a physical row [16]. The other is the memory block, on which memory allocation is carried out. A memory block consists of a number of data blocks. When any of its data blocks has an unrecoverable fault, the memory block is considered to be a failed one and is excluded from allocation for storing and accessing data, which concludes the lifetime of the memory block. A memory block can be of the size of the last-level cache line (e.g., 256 Bytes) or be an operating system page (e.g., 4K Bytes). In the paper we only present results for 4KB pages, and the results for the other memory block size (256B) show a similar trend.

We assume a perfect wear leveling operation across the memory blocks, so that writes are uniformly distributed over the live memory blocks. This assumption is well taken as techniques such as Randomized Region-based Start-Gap [12] and the Security Refresh [15] have demonstrated an effect close to this.

We assume there is a read operation immediately preceding any write request to a data block, so that the data to be written to a data block and the data currently stored in the block can be compared. In this way, the bits of the same values in the two data items can be excluded from the PCM write operation to reduce cell wearing [8, 18]. We assume that a cell has a 50% probability to be excluded in serving a write request. In the simulation, we continuously issue page writes to a 8MB PCM memory until all memory blocks are dead to collect statistics for reporting.

3.2 Comparing *Aegis* with Existing Schemes

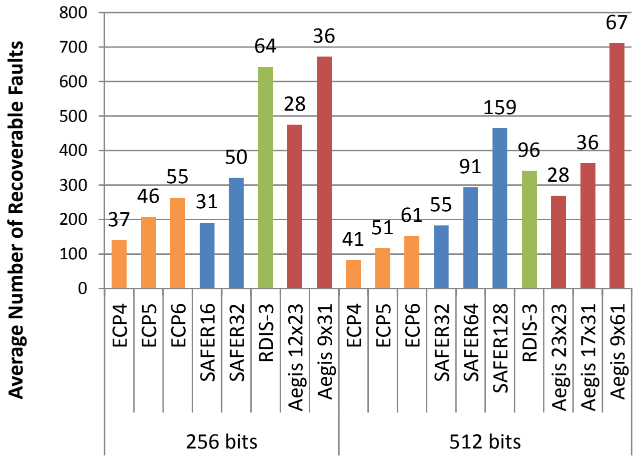

Figure 5: Average number of recoverable faults in a 4KB page with either 256-bit data blocks or 512-bit data blocks. Results for various *ECP*, *RDIS*, *SAFER*, and *Aegis* schemes are shown. In addition, the numbers of bits required for protecting each data block is shown above respective bars.

Figure 5 shows average number of faults that can be recovered in a 4KB-page before any of the page's data blocks has an unrecoverable fault, or soft FTC. The data block size

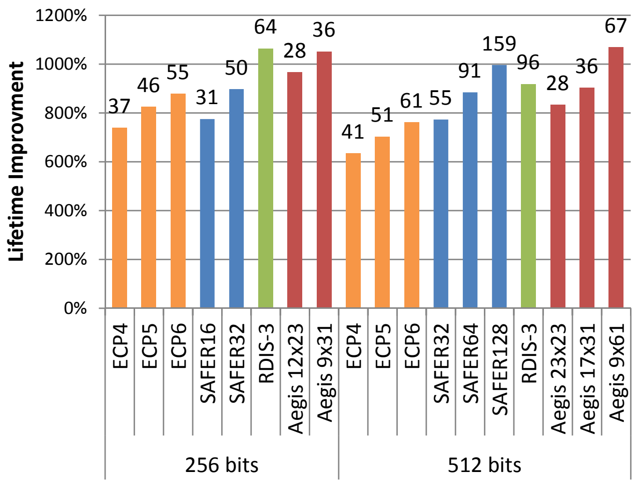

Figure 6: Improvement of a 4KB-page's lifetime in percentage over that of a 4KB page without any error protection. A page comprises either 256-bit data blocks or 512-bit data blocks. Results for various *ECP*, *RDIS*, *SAFER*, and *Aegis* schemes are shown. In addition, the numbers of bits required for protecting each data block is shown above respective bars.

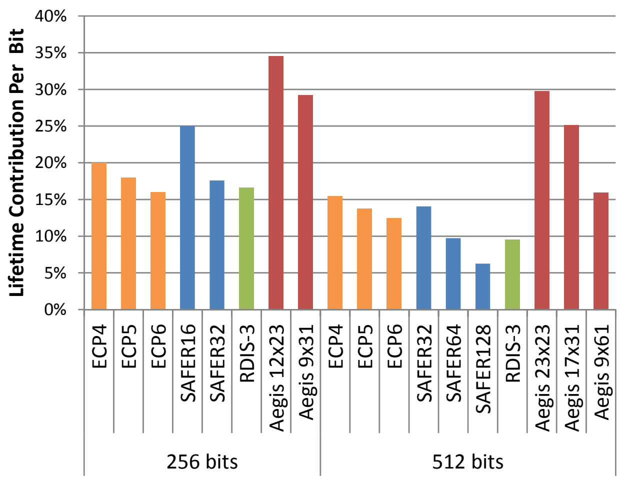

Figure 7: Each-overhead-bit's contribution to the improvement of a 4KB-page's lifetime. The improvements are the ones shown in Figure 6.

can be either 256 bits or 512 bits. Each scheme's required overhead bits are also shown. In the figure we observe that with similar or even lower space overhead *Aegis* can tolerate a much higher number of faults. For example, with 512-bit blocks *Aegis* 9×61 spends 67 bits to tolerate 711 faults in a page, while *SAFER64* spends 91 bits to tolerate only 293 faults. Even worse, *SAFER128* spends much more bits (159) but can only tolerate 465 faults. With 256 bits, *Aegis* 12×23 spends only 28 bits to achieve 474-fault tolerance, while *ECP6* needs 55 bits to reach 264-fault tolerance capability. *RDIS-3* can be too expensive to use for small data blocks. With 256-bit data blocks, *RDIS-3*'s space overhead is 25% of data space. This overhead is reduced to 19% with 512-bit blocks. In the meantime, *Aegis* 9×61's overhead is only 13% but it provides a much stronger fault tolerance (711 faults for the *Aegis* vs. 342 faults for *RDIS-3*).

Figure 6 shows improvement of lifetime of a 4KB-page, or the number of page writes it can sustain before seeing its first unrecoverable fault, under various error recovery schemes over that of a 4KB-page without any fault protection. The trend of the improvement well matches that of tolerable faults shown in Figure 5. However, the lifetime gaps between different schemes shown in Figure 6 are smaller than those about number of recoverable faults shown in Figure 5. For example, with 512-bit data blocks *Aegis* 9×61 can tolerate 95% more faults than *Aegis* 17×31 (711 vs. 364). However, it can only improve lifetime by 19% (10.7X vs. 9.0X). The reason is that faults mostly occur when a page approaches the end of its lifetime, or the density of faults is high at the last segment of its lifetime. Therefore, recovering a relatively large number additional faults may be translated into a small reduction of lifetime. Even so, *Aegis* still provides the most cost-effective fault protection. If we use the 12.5% space overhead of the (72, 64) Hamming coding, the most popular ECC scheme, as the upper bound of any schemes' space overhead, there are six schemes shown in Figure 6 meeting this criteria, which are *ECP4*, *ECP5*, *ECP6*, *SAFER32*, *Aegis* 23×23, and *Aegis* 17×31 with 512-bit data blocks. Among the schemes, the two *Aegis* schemes provide the highest lifetime improvements with the smallest overhead. For example, with only 5.5% space overhead *Aegis* 23×23 provides a 32% larger lifetime improvement (8.3X vs 6.3X) than *ECP4*, the non-Aegis scheme with the smallest overhead (8%).

In all the schemes, increased overhead space leads to more recoverable faults and larger lifetime improvement. In the meantime, each-overhead-bit's contribution to a page's lifetime improvement is decreasing, as shown in Figure 7. Among the scheme, *ECP* has the smallest decrease as each of its additional pointers guarantees one more recoverable fault. Both *SAFER* and *Aegis* suffer substantial decrease, indicating that increasing a scheme's fault tolerance capability can compromise efficiency of its space used for fault protection, as it is challenging to ensure that additional partition groups can be well utilized to accommodate faults before a data block fails. However, if we compare the per-bit contributions between different schemes, *Aegis* has a clear advantage. In both 256-bit data block and 512-bit data block scenarios, the *Aegis* schemes with the lowest contribution (*Aegis* 9×31 and *Aegis* 9×61, respectively) provide higher contribution values than any other schemes in their respective scenarios. Considering that they also provide the (almost) highest lifetime improvements, these *Aegis* schemes are strong error-recovery scheme candidates for PCM chips demanding long lifetime.

While Figure 5 reports average number of faults that makes a 4KB-page fail, Figure 8 depicts how likely each of its 512-bit data block fails after a particular number of faults occur in the block. Apparently before these schemes' hard FTCs are reached, the block's failure probability is 0%. After the hard FTCs, *ECP*'s curves almost vertically rise as it cannot tolerate any faults beyond hard FTC. *Aegis* generally performs better than *SAFER*, even with a lower space cost. For example, *Aegis* 9×61 has a lower space cost (67 bits) than *SAFER64* ((91 bits) and *SAFER128* (159 bits). However, it has lower failure probabilities. Furthermore, it is even better than *SAFER64-cache*, which uses a sufficiently large cache

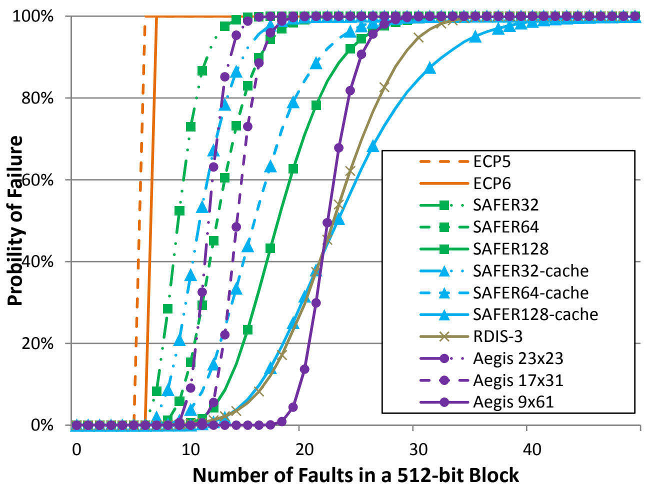

Figure 8: Failure probability of a 512-bit data block with various numbers of faults under different fault recovery schemes.

and substantially improves *SAFER64*. Without a cache, *Aegis* 9×61 has to generate intensive inversion writes in response to a large number groups containing faults when there are more than 20 faults in the block. In contrast, with the help of a cache *SAFER128-cache* and *RDIS-3* perform better when the fault count is beyond 22. *Aegis* can regain its advantage when it is upgraded to its cache-available variants.

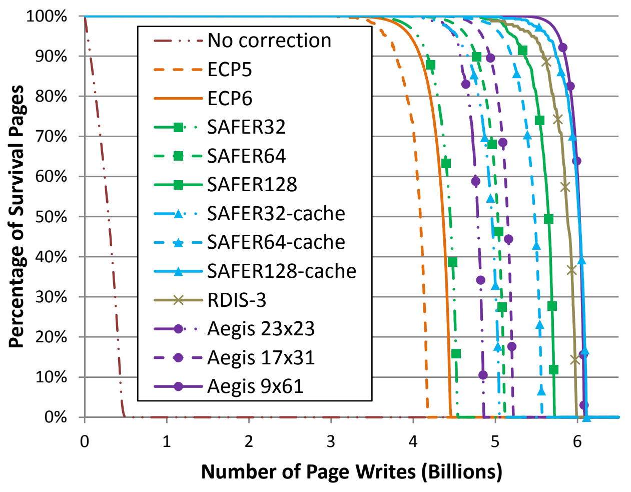

Figure 9: 4KB-page survival rate with continuous page writes under various fault recovery schemes. 512-bit data block is adopted.

While Figure 6 shows *average* lifetime improvement of a 4KB-page, Figure 9 shows percentage of 4KB pages in the tested 8MB memory that are still alive after certain number of page writes. As shown, for all the schemes the curves precipitate once failed pages start to substantially appear, indicating that a PCM chip would soon become unusable once page failures become not rare. This is attributed to the prefect wear leveling assumed in the evaluation. The trend about disparity among the schemes well matches that shown in Figure 8 about block failure probability. We use number of page writes causing half of pages in an 8MB chip to fail as the metric, named as half lifetime. Among various schemes, *Aegis* provides the best half lifetime. For example, *Aegis 17* $\times$ *31* and *SAFER32* use similar number of partition groups (31 vs. 32), *Aegis 17* $\times$ *31* extends *SAFER32*'s half lifetime by 0.7 billion page writes, or by 16%. Furthermore, it even has a larger half lifetime than *SAFER32-cache*. As shown, *Aegis 9* $\times$ *61* has a half lifetime almost equal to that of *SAFER128-cache*. More important, it achieves this without using a cache and with only 42% of the overhead bits used by *SAFER128-cache* for protecting each data block (67 vs. 159).

3.3 Comparing *Aegis* with its Variants

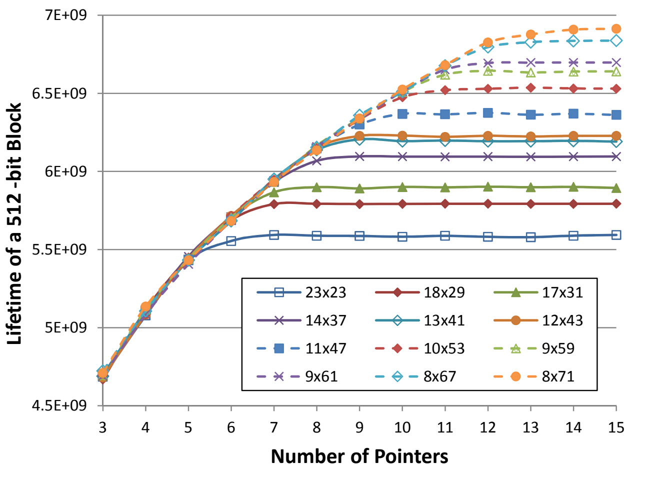

Figure 10: Lifetime of a 512-bit data block protected with various *Aegis-rw-p* $A \times B$ schemes that use different numbers of pointers (p).

In this section we evaluate two *Aegis* variants, *Aegis-rw* and *Aegis-rw-p*, and compare their results with *Aegis*'s to understand how these two variants may improve *Aegis*. Because we assume existence of a sufficiently large cache for the two variants, the experiment results presented in this section also help to reveal how much *Aegis* can benefit from the use of a cache.

Depending on the number of pointers used in *Aegis-rw-p*, or the p value, the scheme exhibits different cost-effectiveness. Figure 10 shows the lifetime of a 512-bit data block in terms of number of writes under the *Aegis-rw-p* scheme when we use different p values and different $A \times B$ formations. As shown in the figure, for each $A \times B$ curve the lifetime increases quickly with the increase of p when p is relatively small. Then it reaches a plateau and further increasing p receives little return on the block's lifetime. The pointers are used to record faults that can be recovered by the corresponding *Aegis-rw* scheme. When there are a sufficiently large number of pointers, the constraint on *Aegis-rw-p*'s fault tolerance capability is shifted from pointer count to the capability of *Aegis-rw*. It is noted that the plateau represents the lifetime provided by corresponding *Aegis-rw* scheme. The lifetime provided by an $A \times B$ *Aegis-rw* scheme is determined by the prime number B. As shown, the lifetime increases by as much as 24% when B increases from 23 to 71.

As *Aegis-rw-p*'s fault tolerance capacity is bounded by that of *Aegis-rw*, it has to at least have an advantage on space overhead to be considered as a candidate scheme. To this end, we choose some representative *Aegis-rw-p* schemes that have relatively strong capability but smaller space overhead than corresponding *Aegis-rw* schemes to compare with *Aegis* in the following experiments. The schemes are *Aegis-rw-p* 23×23 with 4 pointers, *Aegis-rw-p* 17×31 with 5 pointers, *Aegis-rw-p* 9×61 with 9 pointers, and *Aegis-rw-p* 8×71 with 9 pointers.

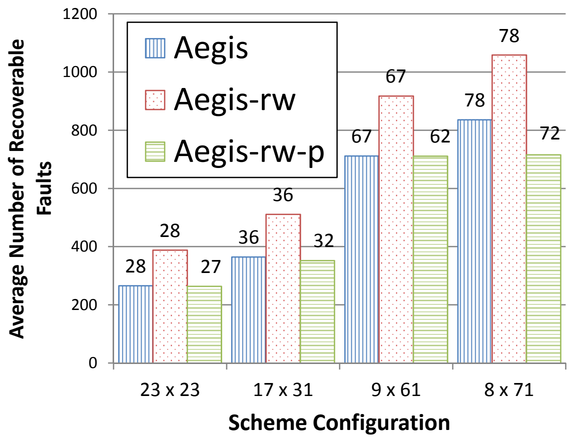

Figure 11: Average number of recoverable faults of a 4KB page with 512-bit data blocks. Results are for *Aegis*, *Aegis-rw*, and *Aegis-rw-p* with different $A \times B$ formations. In addition, the numbers of bits required for protecting each data block are shown above respective bars.

Figure 11 shows number of recoverable faults in a 4KB page for *Aegis*, *Aegis-rw*, and *Aegis-rw-p*. Taking advantage of knowledge on the distinction of R and W faults, *Aegis-rw* substantially increases number of recoverable faults than *Aegis* (by 52%, 41%, 33%, and 28% for *Aegis*es 23×23, 17×31, 9×61, and 8×71, respectively). This improvement is expected as *Aegis-rw* explicitly allows multiple faults to be in the same partition group as long as they are of the same nature (R or W faults). In addition, *Aegis-rw* removes extra inversion writes, which can slow down the cells' wearing. However, this improvement does not come for free. A cache is required to remember all faults on a PCM chip if every write request is expected to obtain the fault information about the page to be written before actual write operation is performed. When the chip is still in its early age, there are only a few faults for the cache to record. However, when the chip gets old and many cells approach their lifetimes, the cache has to be sized sufficiently large to accommodate a flush of faults. *Aegis-rw-p* provides a means to flexibly adjust space overhead by varying the number of pointers. However, as soon as its overhead is lowered below that of *Aegis-rw*, *Aegis-rw-p*'s fault tolerance capacity is reduced to a number close to or even smaller than that of *Aegis* (See Figure 11). In the meantime, as shown in Figure 12 *Aegis-rw-p*'s lifetime improvement is consistently higher than that of corresponding *Aegis*, because it removes extra inversion writes and reduces writes to the cells. This figure also shows that *Aegis-rw* produces the largest lifetime improvement, though at a smaller scale than that for recoverable faults.

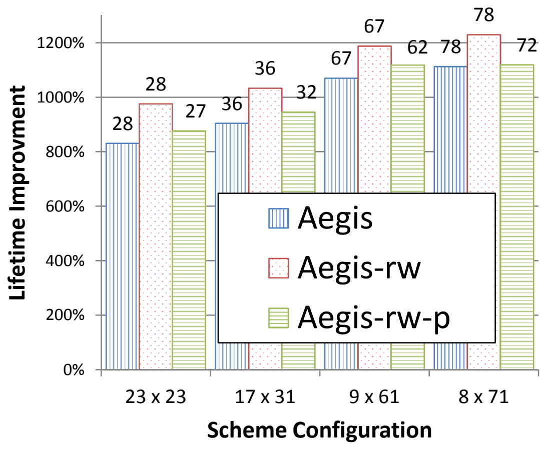

Figure 12: Improvement of a 4KB-page's lifetime in percentage over that of a 4KB page without any error protection. A page comprises 512-bit data blocks. Results are for *Aegis*, *Aegis-rw*, and *Aegis-rw-p* with different $A \times B$ formations. In addition, the numbers of bits required for protecting each data block are shown above respective bars.

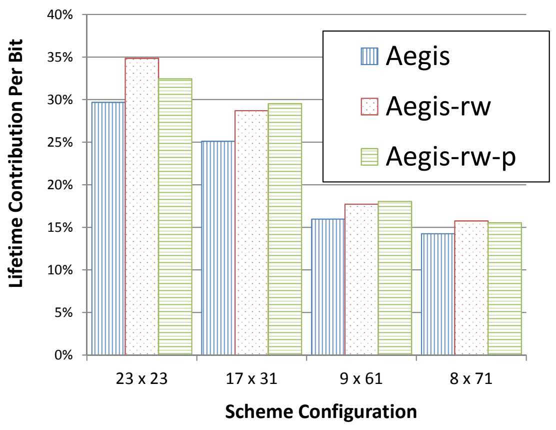

Figure 13: Each-overhead-bit's contribution to the improvement of a 4KB-page's lifetime. The improvements are the ones shown in Figure 12.

Figure 13 shows each-overhead-bit's contribution to the improvement of the lifetime shown in Figure 12. According to the results, the two *Aegis*'s variants can more efficiently use the overhead space. In particular, the per-bit contribution of *Aegis-rw-p* can be higher than that of *Aegis-rw*. However, taking into account of the space used for the fail cache, *Aegis* is likely more efficient on its use of overhead space than the two variants.

In the experiments we do not evaluate the impact of cache size on the two variants' results, as the impact is subject to the workload characteristics, lifetime variation, and economic factors. We would like to leave the study of two variants's merits in a more general context as a future work. In summary, without relying on a possibly expensive cache *Aegis* has been able to cost-effectively improve PCM's reliability. *Aegis-rw* is used when strong reliability takes priority while *Aegis-rw-p* can be used for trading reliability for cost.

4. RELATED WORK

Aegis is designed to recover permanent stuck-at faults for PCM using an optimized partition-and-inversion approach. Below we discuss related works on error protection for memories, especially the PCM memory.

Transient and Permanent Faults. Unlike resistive memories, typical errors in DRAM are transient and are possibly induced by alpha particles. Usually the Error Correcting Code (ECC) schemes, such as hamming code, are used to correct the errors. However, it is expensive to use ECC to correct multiple errors in a data block. This is not a serious issue for DRAM as multiple transient errors simultaneously induced by alpha particles in a block as small as 64 bits is rare [9]. However, PCM is less likely to have transient errors. Instead, it has permanent stuck-at faults that can be gradually accumulated over time within the lifetime of a data block. Hence, the more faults a scheme can tolerate, the longer the block's lifetime is. *Aegis* allows substantially more faults to be corrected and the effort can directly benefit lifetime and usability of the PCM memory. Permanent faults can be recovered at different levels, including at a higher level for leveraging support from operating system (OS) and at a low level built within a chip to protect individual data blocks.

OS-assisted Recovery of Stuck-at Faults and On-chip Recovery. The most intuitive way for OS to tolerate faults is to exclude any memory blocks (or pages) containing unrecoverable faults from memory allocation. To be effective, this strategy assumes an on-chip fault tolerance scheme that has provided strong protection of data blocks within a page. Otherwise, the pool of allocatable pages can be quickly depleted. There have been a number of optimized schemes proposed to address the concern. The Dynamic Pairing scheme attempts to reuse faulty pages by pairing two pages that do not have faults at the same offsets [7]. A pair of faulty pages are accessed in place of one fault-free page. In addition to the use of page pairing technique, the RePRAM scheme introduces parity code across a group of faulty pages to add another layer of protection [6]. While these schemes can slow down the rate of page loss, they are not compatible with wear-leveling techniques that are critical in the use of PCM. FREE-p is another scheme relying on OS to re-direct access of a faulty block of cacheline size to a non-faulty block via a pointer embedded in the faulty block [17]. This re-direction is initiated after the in-block protection becomes incapable of correcting faults. With *Aegis*'s strong fault tolerance capability, the re-direction as well as loss of faulty pages can be substantially delayed. In contrast with the OS-assisted scheme, on-chip recovery schemes provide the first line of defense against PCM faults.

Pointer-based and Partition-and-Inversion-based Recovery Approaches. As we have mentioned in Section 1, there are two approaches to tolerate stuck-at faults. *ECP* [14] belongs to the pointer-based approach that explicitly records fault addresses and uses replacement bits to accommodate data written to the addresses. The concern with this approach is that the number of tolerable faults is limited by the number of pointers. As the space overhead is proportional to the number of pointers, this number has to be relatively small. By applying the pigeonhole principle, the *Aegis-rw-p* scheme can more than double the number of correctable faults using the same number of pointers and make the use of pointers a more viable approach. *SAFER* [16] and *RDIS* [10] belong to the second approach using bit partition and group inversion. The efficacy of the approach depends largely on how a data block is partitioned into groups, or the partition scheme. On this aspect *Aegis* has its advantage over *SAFER* by turning the exponential increase of group count into a polynomial increase and spreading faults more evenly across different groups. *Aegis* has its advantage over *RDIS* by being able to guarantee recovery of a much higher number of faults (only 3 for the *RDIS* scheme suggested in its paper [10]), smaller space overhead, and optional use of the possibly expensive fail cache.

As there is a high variability in lifetime across memory cells, it is not cost effective to spend equal amount of space on every data block without considering actual number of faults in individual blocks. To this end, Qureshi et al. proposed a Pay-As-You-Go (PAYG) framework that associates smaller amount of space with each data block for local error correction (LEC) to correct up to one fault per block and allocates space from a global error correction (GEC) pool for any blocks whose faults cannot be corrected by the LEC [13]. As PAYG is a framework that can employ any error correction scheme in its GEC component, *Aegis* complements PAYG with its strong fault tolerance capability and its space efficiency.

5. CONCLUSION

We propose the design and implementation of *Aegis*, an efficient fault recovery scheme, in response to the urgent need for protecting resistive memories, such as PCM, from stuck-at faults. Different from transient errors, the permanent faults in PCM can be accumulated. A recovery scheme that can efficiently tolerate a large number of faults in a data block is demanded for wide adoption of the technique. *Aegis* intelligently partitions a block into a relatively small number of groups and effectively distributes faults into different groups, so that the faults can be tolerated by using bit inversion in selected groups. Compared with representative fault tolerance schemes such as *ECP* and *SAFER*, *Aegis* requires smaller overhead space to store information on how to correct errors and achieves higher fault tolerance and longer memory lifetime. As an anecdotal evidence, our experiments show that *Aegis* 17×31 uses only 7% of the memory as overhead space to tolerate 24% more faults than *SAFER64*, which has to use 18% of the memory as overhead space. This overhead is even larger than that of *Aegis* 9×61 (13%). However, *Aegis* 9×61 tolerates 142% more faults and extends its a memory page's lifetime by 21% if compared with *SAFER64*.

In addition, *Aegis* can be flexibly configured to meet requirement on PCM reliability. If a cache is available, *Aegis* can take advantage of it for higher fault tolerance capacity. It can also choose a larger prime number as B in *Aegis* $A \times B$ to accommodate more faults. While cost-effectiveness of the cache depends on workload characteristics, we leave it as a future work to study optimality of *Aegis*'s variations in various specific application scenarios.

6. ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant No. 60925006), the State Key Program of National Natural Science of China (Grant No. 61232003), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), and Shanghai Key Laboratory of Scalable Computing and Systems, and the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and Tsinghua University Initiative Scientific Research Program.

7. REFERENCES

[1] Emerging research devices. In *International Technology Roadmap for Semiconductors*, 2011.

[2] K. Kim. "Technology for sub-50nm DRAM and NAND flash manufacturing", In *International Electron Devices Meeting*, 2005.

[3] Micron Announces Availability of Phase Change Memory for Mobile Devices. In *http://investors.micron.com/releasedetail.cfm ?ReleaseID=692563*, July, 2012.

[4] Samsung Ships Industry's First Multi-chip Package with a PRAM Chip for Handsets. In *http://www.samsung.com/us/aboutsamsung/news/newsIrRead.do?news_ctgry=irnewsrelease&news_seq=18828* April, 2010.

[5] J. Condit, E. Nightingale, C. Frost, E. Ipek, D. Burger, B. Lee, and D. Coetzee. "Better I/O Through Byte-Addressable, Persistent Memory", In *Symposium on Operating Systems Principles*, October 2009.

[6] J. Chen, G. Venkataramani, and H. H. Huang. "RePRAM: Re-cycling PRAM Faulty Blocks for Extended Lifetime", In *IEEE International Conference on Dependable Systems and Networks*, 2012.

[7] 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*, 2010.

[8] B. 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*, June 2009.

[9] F. Matsuoka and F. Masuoka. "Numerical Analysis of Alpha-particle-induced Soft Errors in Floating Channel Type Surrounding Gate Transistor (FC-SGT) DRAM Cell", In *IEEE Transactions on Electron Devices*, 2003.

[10] R. Melhem, R, R. Maddah, and S. Cho. "RDIS: A Recursively Defined Invertible Set Scheme to Tolerate Multiple Stuck-At Faults in Resistive Memory", In *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2012.

[11] 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*, 2007.

[12] 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.

[13] M. K. Qureshi. "Pay-As-You-Go: Low-Overhead Hard-Error Correction for Phase Change Memories", In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, December, 2011.

[14] S. Schechter, G. 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*, June 2010.

[15] 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*, 2010.

[16] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H. S. Lee. "SAFER: Stuck-At-Fault Error Recovery for Memories", In *Proceedings of the 43th Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[17] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. Jouppi, and M. Erez. "FREE-p: Protecting Non-volatile Memory against both Hard and Soft Errors", In *Proceedings of IEEE 17th International Symposium on High Performance Computer Architecture*, 2011.

[18] 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.