Yielding Optimized Dependability Assurance through Bit Inversion

Jiangwei Zhang, Donald Kline Jr, Liang Fang, Rami Melhem, Alex K. Jones

*College of Computer, National University of Defense Technology, Changsha 410073, China*

*State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China*

*Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261, U.S.A.*

*Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, U.S.A.*

**Abstract**

Phase change memory (PCM) is a promising alternative to conventional DRAM main memories, due to its read performance, density, and non-volatility and resulting low static energy. Unfortunately, reliability is still a significant challenge as limited write endurance, exacerbated by process variation, leads to increasing numbers of stuck-at faults over the memory’s lifetime. This includes a significant number of stuck-at faults that appear early in the memory’s service. Error-correcting Pointers (ECP) is a popular proposal to mitigate stuck-at faults in PCM by recording the addresses and the values of faulty bits in order to extend the lifetime of the memory. We propose a method to extend the effectiveness of ECP coverage called Yoda, which utilizes a small number of additional encoding bits in order to dramatically improve the effectiveness and fault correction capability of ECP. By adding one additional bit to ECP which corrects f faults, Yoda can correct 2f+1 faults. Further improvements are possible introducing small numbers additional bits. Our simulation results demonstrate that Yoda has a 3.0× improvement in fault coverage compared to a fault-aware ECP with a similar overhead, while also providing a 2.5-3.0× improvement over state-of-the-art schemes with comparable complexity. Furthermore, Yoda provides a method to protect the auxiliary bits, also with a small overhead. By adding one auxiliary bit to protect the auxiliary bits, Yoda can achieve extra improvement.
Figure 1: Cell failure rate goes up as the number of writes to each memory row increases for different process variations

*Keywords:* Phase Change Memory, Reliability, Stuck-at Faults, and Error-correcting Pointer

1. Introduction

Phase change memory (PCM) is being extensively studied as a potential replacement to conventional memories, such as DRAM and Flash. The scaling of conventional memories to very small feature sizes has become increasingly challenging due to physical limitations, yield problems, and poor reliability [1, 2, 3]. In contrast, PCM is an attractive alternative due to its scalability, near DRAM performance, high density, and non-volatility which, in particular dramatically reduces static energy [4, 5, 6, 7]. Moreover, PCM is nearing mass production from several vendors and is the heart of the recently announced Intel XPoint memory. Unfortunately, PCM does have a limited lifetime due to write endurance challenges of PCM cells.

PCM’s endurance is limited due to the physical phase-change process between high and low resistance states. Typical PCM cells can achieve between $10^8$ to $10^9$ write cycles. Unfortunately, materials research does not appear poised to dramatically improve the endurance of PCM in the near term [8, 9, 10, 11]. Additionally, early failures of some “weak cells” has been noted due to process variation. In Figure 1, we show the relationship between number of writes to a cell and its probability of failure assuming a mean time
to failure of $10^8$ writes, for different coefficients of variation (CoV). For example, if CoV=0.25, the cell failure ratio is $10^{-4}$ at $9 \times 10^6$ writes and is $10^{-2}$ at $4.2 \times 10^7$ writes. Thus, if more of these failures can be tolerated, the value and longevity of PCM can be significantly improved.

PCM endurance faults manifest as “stuck-at” faults. The cell may still be read as the stuck value but cannot be written. Thus, for the cell to remain in service, all values written to the cell must either be the same as the stuck-at value (stuck-at right or SA-R) or if it is different (stuck-at wrong, or SA-W) it must be corrected via some error correction capability. An important note is that across stuck-at faults, only the SA-Ws can lead to memory errors. Thus, a method to leverage SA-Rs and minimize SA-Ws can enhance reliability. One such method is to invert data blocks that contain SA-Ws to convert them to SA-Rs. By partitioning data into blocks that segregate SA-Ws and SA-Rs, all faults can be converted into SA-Rs. This encoding concept is called “partition-and-flip.”

In this paper, we present Yoda [12], which is a new fault recovery scheme for stuck-at faults called Yoda. Yoda is compatible with the error-correction pointers (ECP) technique proposed by Microsoft but significantly improves the fault protection guarantee of ECP by adding small number of encoding bits. Thus, Yoda combines the concept of partition-and-flip style encoding with ECP in order to allow multiple stuck-at faults to coexist within the same partition. Moreover, we propose a mechanism to protect the encoding bits, allowing them to be stored in traditional PCM, as opposed to a fault-free storage that is often assumed for auxiliary bits. The resulting correction is able to correct $3.0 \times$ the number of faults of ECP alone and $2.5-3.0 \times$ the number of faults of previous partition and flip with similar en/decoding complexity. Additionally, for small and moderate numbers of faults, Yoda provides better guaranteed protection than significantly more complex partitioning approaches. In particular, this paper makes the following contributions:

1. We propose Yoda, a method to extend pointers with partition and flip capabilities to dramatically improve guaranteed number of protected bits over pointers alone with very low extra overhead.
2. We demonstrate a “small” Yoda method that further decreases the number of required encoding bits to achieve the same protection of Yoda at the expense of small additional hardware encoding and decoding overhead.
3. We propose a compatible low overhead technique to protect the encod-
ing bits so that they may also be stored in potentially faulty memory as opposed to a guaranteed protected memory space.

4. We provide a characterization of the recovery from stuck-at faults of Yoda, and Small Yoda in comparison to ECP and other comparable partition and flip schemes through an extensive study.

5. We provide a detailed probabilistic evaluation of reliability for all possible data patterns for different fault distributions, as well as the lifetime improvement overall and per bit of our Yoda schemes compared to state-of-the-art schemes with similar overhead.

The remainder of the paper is organized as follows. In Section 2, we provide a background and discuss the related work on fault-tolerance schemes applied to stuck-at faults in emerging memories. In Section 3, we describe Yoda in detail, as well as introduce the proposed method to protect the auxiliary bits. In Section 4, we demonstrate that Yoda is a system with a guaranteed protection. Section 5 provides experimental methodology and experimental results of Yoda schemes compared to the current leading approaches. In Section 6, we relate conclusions.

2. Background

To tolerate a SA-W bit, there are generally three classes of approaches. The first class is partition-and-flip schemes, such as SAFER [13], RDIS [14], and Aegis [15]. These approaches attempt to partition a data block to separate stuck-at cells into different groups and use a flag bit to potentially invert each group. If the stuck-at value is SA-W in a group the group is inverted, but if the stuck-at value is SA-R, the group is not inverted. The second class is error-correcting pointers (ECP) and its related schemes [16, 17]. ECP uses pointers to record the addresses of stuck-at faults within the memory block and stores the replacement value to use in place of the one that could be SA-W in the faulty cell. Finally, fault-corrections strategies such as error correction codes (ECC) [18] can be used to attempt to recreate the faulty SA-W bits upon subsequent read accesses. Other schemes, such as coset encoding, attempt to reduce energy and extend lifetime by reducing bit changes [19, 20]. Coset encoding could also potentially hide stuck-at faults within the memory by encoding the data into multiple candidates that may not contain any SA-W bits in the resulting data word, however, these approaches are complex to implement and cannot guarantee fault correction as their are probabilistic
in nature. Thus, we focus on the three main categories for description and comparison.

2.1. Error Correction Codes

Error Correction Codes (ECC) [18] are designed to correct transient faults in memory. SECDED (single error correction, double error detection) is the most frequently used form of ECC, commonly applied to DRAM main memory. ECC can easily be applied to protect PCM against stuck-at faults. In fact, for Flash, higher forms of ECC that correct as many as 5 faults may be employed due to endurance problems of that technology. However, stuck-at faults are more predictable than transient faults. Unlike transient memory faults, where the fault is unpredictable, the location of stuck-at faults remains fixed. Moreover, error tolerance for these stuck at faults can leverage this fact to reduce the encoding overhead to correct more faults. Thus, while ECC can be employed to correct stuck-at faults in PCM, it is unnecessarily bulky and lower overhead alternatives are possible.

2.2. Partition and Flip Schemes

Partition and flip (PAF) correction schemes utilize the knowledge that stuck-at faults are permanent, are still readable (read-only), and we can distinguish if a stuck-at cell is either SA-W or SA-R in order to improve on the encoding overhead compared to more generic fault-tolerance provided by ECC. Flip-N-write (FNW) is a scheme originally designed to save energy in PCM by reducing its dynamic power by minimizing the number of bit changes during write operations. If more than 50% of the bits would change between the old and new value, the value is flipped to reduce the transitions to under 50%. FNW also increases the memory lifetime by reducing the bit changes that lead to endurance faults. FNW is simple to implement but has limited effectiveness as a PAF scheme as faults are often clustered and must all be all SA-R or SA-W to be effective. Despite this shortcoming, several existing schemes have substantially improved FNW, as a base operation, by a more complicated partitioning scheme, including SAFER [13], RDIS [14], Aegis [15] and Aegis’ variants (Dynamic Aegis [21] and Relaxed Partitioning [22]). The distinction of SA-W and SA-R bits is required for RDIS, while it is an option for SAFER and Aegis.

RDIS is the first work to leverage the distinction of SA-W and SA-R for correction of stuck-at faults [14]. RDIS transforms a one-dimensional (1D) data block into a two-dimensional (2D) matrix and recursively selects bits
into different sets to separate the SA-W and SA-R bits. The SA-W bits can be tolerated by inverting the corresponding sets while the SA-R sets are not inverted. An obvious shortcoming of this scheme is the large capacity overhead. For a 512-bit data block, RDIS-3, a general application of this scheme, needs 96 flag bits to apply three-deep recursive partitioning, which represents a 19% capacity overhead.

SAFER dynamically partitions a data block into a variable number of groups to avoid collisions of SA-W and SA-R bits in a same group [13]. SAFER starts with a maximum number of groups, say $N$, then it allows $\log_2 N$ repartitions. Repartitioning is done whenever the current partition cannot deal with a new fault. When the $\log_2 N$ repartitions are exhausted, SAFER fails to tolerate a new fault that the current partition cannot tolerate.

Aegis, inspired by the principle that any two points in a line on a Cartesian plane determine the slope of the line, generates an optimized configuration (partition) that completely distributes stuck-at faults into different groups by dynamically examining different slopes [15]. Although Aegis is an effective scheme to tolerate the stuck-at faults in the data block, it is more costly in terms of encoding and decoding overhead to search for a suitable configuration. For example, Aegis utilizes three large look-up tables for its encoding and decoding. Thus, Aegis is a fairly complex scheme that would require a large overhead if implemented in software, or significant modification to existing hardware, such as a memory controller, for implementation.

2.3. Targeted Error Correction (ECP)

In contrast to PAF schemes, the recent ECP approach provides a fixed number of programmable “correction entries” within the memory row. Each entry contains a pointer to record the address of a faulty bit and a replacement bit to store the correct value. When a stuck-at fault is detected, a new entry is applied to replace this faulty location. For example, we show an ECP example in Figure 2(a) that contains five correction entries to protect a 512-bit data block. To record the pointers used, ECP uses a flag bit per block to indicate whether the entries are all full, and if not, the last pointer is encoded to record the used pointers. In our example, there is one SA-R bit and one SA-W bit in the data block, so the first two entries are applied to mitigate them and the corresponding encode in the last pointer is ‘0011’.

Both PAF and ECP have useful qualities to correct stuck-at faults. In the next section we present a new scheme that combines elements of both
ideas to efficiently address these faults with simple encoding and improved fault-tolerance.

3. Yoda

We propose a technique we call Yoda that intelligently combines knowledge of SA-R and SA-W bit locations through pointers, partitioning, and flipping to improve fault tolerance. Yoda is actually short for yielding optimized dependability assurance (Yoda) through one-bit inversion with allotted numbers-of-bits (Obi-wan). In this section we describe a simple extension to ECP to make it more effective with knowledge of SA-R and SA-W bits (Yoda-0), a single bit extension to enhance the error correction (Yoda-1), and a generalized form that can increase error correction (Yoda-N). Furthermore, we describe Small Yoda, a method to decrease the encoding storage overhead of Yoda-N at the expense of more complex encoding/decoding logic as well as a method to protect the encoding bits.
3.1. Stuck-at aware ECP (Yoda-0)

If the system can distinguish between SA-R and SA-W bits, a pointer-based approach could tolerate more potential faults by releasing any entries used for the SA-R bits. In PCM this can be easily accomplished by a read followed by the write to determine the SA-W bits\(^1\). Moreover, in this case, the replacement bit in each entry is not required because only SA-W bits are recorded and can be read and inverted rather than replaced. Furthermore, the flag bit can also be eliminated by having unused pointers set to a non-faulty bit. An inverted value is stored in this location and then flipped back during error correction.

We demonstrate Yoda-0 through an example in Figure 2(b). Yoda-0 does not include a flag bit and each correction entry is just the pointer, recording the address of a SA-W bit. The other unused entries point to the first non-faulty bit (generally the zeroth bit), which is intentionally written in inverted form. Yoda-0 can also be implemented with a flag bit, similar to ECP, for easier decoding. Because Yoda-0 merely records the addresses of the SA-W bits, it doubles the fault-tolerance capacity on average over original ECP. Moreover, this can immediately be applied to systems that implement ECP with minimal modification.

3.2. Yoda-1

Yoda-1 extends Yoda-0 by adding one additional inversion bit to indicate whether the entire block should be written in inverted form. Yoda-1 is a cost-effective scheme that substantially extends ECP’s fault-tolerance capability by adding one additional flag bit per data block. This inversion bit is used to tell whether the number of SA-W bits is larger than that of SA-R, and if yes, then record a ‘1’ and invert the data block, otherwise, record a ‘0’ and directly store the data block. In this way, the faults which comprise the majority (SA-W or SA-R) can be tolerated using this inversion flag. Meanwhile, the pointers are leveraged to mitigate the kind of faults in the minority (SA-R or SA-W) by recording their addresses. By adding this one additional bit to Yoda-0 which can guarantee correction of \(k\) faults, Yoda-1 can guarantee to correct \(2k+1\) faults.

\(^1\)Often PCM writes already employ a similar technique to write and check, followed by multiple re-writes, to ensure correctly written values [23]. Stuck-at bits could be considered faults that occur after a certain number of rewrites, determined by a threshold.
In Figure 3, we demonstrate two examples for Yoda-1. First, there are two SA-R bits and one SA-W bit in a 512-bit data block, data. [Figure 3(a)]. In this case the majority of stuck-at faults (SA-W or SA-R) are SA-R’s. In this scenario, the inversion bit is set to ‘0’ and the data block is not inverted, while the first pointer is used to record the address of the only SA-W bit, data[0]. The unused pointers are initialized to any non-faulty bit in the data block, for example data[1], writing the opposite value of this bit. In our implementation, all unused pointers point to the lowest order bit that is not stuck-at.

In the second example there is one SA-R bit and three SA-W bits in the data block, so the majority of stuck-at bits (SA-W or SA-R) are SA-W’s. The flag bit and the data block are inverted to tolerate these three SA-W faults. Then, the previous SA-W bits become SA-R bits, while the previous SA-R bit becomes a SA-W bit, which can be corrected by recording its address in the first pointer. The unused pointer is initialized to data[0] as this bit is non-faulty. To tolerate the stuck-at faults in this example, Yoda-1 only

![Diagram](9)

Figure 3: Examples of Yoda-1 with two pointers: (a) two SA-R bits and one SA-W bit and (b) one SA-R and three SA-W bits.
3.3. Yoda-N

By combining pointers with knowledge of stuck-at faults we get improvements in fault tolerance (Yoda-0) and by adding inversion we can guarantee a much higher fault tolerance (Yoda-1). We can further enhance fault tolerance by partitioning the block into groups to apply inversion. Thus, in Yoda-N, we partition the data block into $N$ groups, similar to FNW. For each group, we use an inversion bit to store the status whether the group should be inverted based on the majority of stuck-at faults in the group. After inverting the data block using the flag bits, if there are uncorrectable faults, we use pointers to record their addresses to mitigate them. Pointers could point to faults within a particular group, be spread across many groups, or some hybrid between. Yoda-N is a general form that covers Yoda-0 and Yoda-1. Yoda-1 ($N=1$) contains a single group, the whole data block, which can be inverted and Yoda-0 ($N=0$) has zero data blocks that may be inverted.

In Figure 4, we illustrate an example for which the stuck-at faults can be tolerated using two pointers by Yoda-2 (partitioning into two groups), but cannot be tolerated by Yoda-0, or Yoda-1. Yoda-2 only requires one additional encoding bit over Yoda-1. There are seven faults in the data block. The faults happen to be accumulated as the combination of errors appearing in the data blocks of Figures 3(a) and (b), resulting in three SA-R bits and four SA-W bits. The data block is partitioned into two groups, one requiring one pointer but Yoda-0 would have required three.

Figure 4: Example of Yoda for $N=2$ (Yoda-2) with two pointers.

---

2Yoda refers to Yoda-N in the remainder of the paper.
containing the most significant bits (group_1) and one containing the least significant bits (group_0). The zeroth group has three SA-W bits and one SA-R bit, so the group is inverted and the corresponding inversion bit is encoded as ‘1’. The first group has two SA-R bits and one SA-W bit, so the corresponding inversion bit is encoded as ‘0’. After this inversion step, there are only two uncorrectable faults remaining: one inverted SA-R bit in the zeroth group and one SA-W bit in the first group. To tolerate these faults, two pointers are used to point to these locations. In this case, ECP, Yoda-0, and Yoda-1 require seven, four, and three pointers (61, 36, and 28 auxiliary bits), respectively, to tolerate all the faults while Yoda-2 requires only two pointers with two flag bits (20 auxiliary bits).

3.3.1. Small Yoda

For systems requiring three or more pointers, the pointers for Yoda can be compressed at the expense of additional encoding and decoding overhead. We call this extension to Yoda, “Small” Yoda. The space overhead of Small Yoda (also referred to as Yoda-S) can be reduced to be lower than Yoda-1 or Yoda-0. In Figure 5, we describe the compression method of Small Yoda pointers based on the observation that pointers may be specified relative to the group. In the figure, there are three pointers which initially consist of nine-bits to address the 512-bit data block. The data block is partitioned into two groups, each with 256 bits. Relative to the 512-bit block, if the most significant bit (MSB) of a pointer is ‘0’, the pointer corresponds to a faulty bit in the zeroth group, while if the MSB of a pointer is ‘1’, the pointer links to a faulty bit in the first group. The MSB of each pointer can be removed and replaced with an encoding of the grouping, and in the case of Yoda-2 with three pointers there are four permutations of pointer locations: “00”: three pointers in group zero, “01” two pointers in group zero one in group one, “10” one pointer in group zero two in group one, and “11” all pointers in group one. In this way, by using two bits to store the permutations of the three pointers we actually save one auxiliary bit. As more pointers are available and the data block is partitioned into more groups, we can save even more bits through this compression. For example, if the data block is partitioned into four groups (Yoda-4) with six pointers to mitigate 13 stuck-at faults, then Small Yoda-4 with six pointers saves five and two auxiliary bits over Yoda-4 and Yoda-1, respectively.

In general, the number of encoding bits depends on the number of pointer grouping permutations. For example, if there are m grouping permutations,
we require $\lceil \log(m) \rceil$ encoding bits. If the data block is partitioned into $N$ groups, the length of each pointer can be compressed to $9-\lfloor \log(N) \rfloor$ bits. If there are $k$ pointers, the savings through compression is $k \lfloor \log(N) \rfloor - \lceil \log(m) \rceil$.

To compress the pointers, we require a small lookup table to translate the grouping modes to the corresponding code of the encoding bits. Notably, the fault-tolerance capacity of Small Yoda is equivalent to that of Yoda. The reduced space overhead is achieved by a tradeoff in the encoding and decoding complexity.

3.3.2. Protection of Overhead Bits

Storing the auxiliary bits in a safe reliable memory, such as SRAM within the memory controller, is a significant overhead. However, if the auxiliary bits are stored in the memory susceptible to stuck-at faults, such as additional PCM bits, the auxiliary bits also require suitable protection. ECC is one option to tolerate the potential faults in the auxiliary bits, which also protects against faults in the ECC parity bits. However, ECC has a high overhead. For example, for Yoda with three pointers, 29 auxiliary bits are required, which would require six parity bits for SECDED (ECC-132).

In contrast and in the spirit of Yoda, we propose adding only a single error correction bit to protect the auxiliary bits. This bit is an additional inversion bit, which allows tolerance of one stuck-at fault in the auxiliary bits, including the additional inversion bit itself, which has a similar correction capability of ECC-1. In particular, if there is only a single SA-W bit (or SA-R bit) within the auxiliary bits, wherever the fault is, it can be tolerated.
Figure 6: Overhead bits required to guarantee a particular fault-tolerance for a 512-bit data block.

Figure 7: Overhead bits required to guarantee a particular fault-tolerance for a 1024-bit data block.

by inverting (keeping the values of) the auxiliary bits and the additional bit. Furthermore, if there is only one class of stuck-at faults, meaning that only SA-R or SA-W bits, the faults can always be corrected by this additional bit. In contrast, if there are at least two SA-W bits, ECC-1 cannot correct both of them.

This scheme can tolerate a stuck-at fault in the auxiliary inversion flag bit. If the inversion bit is stuck-at ‘1’ (inverting) then the complement of the auxiliary bits are written. Otherwise the inverting bit is stuck-at ‘0’ meaning the original auxiliary bits would be written. Of course, if the auxiliary protection bit becomes stuck-at, it cannot guarantee correction of any other faults in the auxiliary bits.
4. Fault-tolerance Guarantee

In fault-tolerance it is desirable to provide a system that has a guaranteed number of corrected faults. As such, we compared the space overheads for Yoda against different recovery schemes to tolerate a given number of stuck-at faults for 512-bit and 1024-bit data blocks. In particular, we provide results for ECC, SAFER, and Aegis\_rw. RDIS is not designed to scale to large numbers of faults, and is reported for three faults (RDIS-3), which requires 96 and 128 auxiliary bits per 512-bit and 1024-bit data block, respectively.

Figure 6 and Figure 7 show the overhead comparison for 512- and 1024-bit blocks, respectively. SAFER is effective for low numbers of faults, it has the minimum cost for correcting one (tied with Yoda, excluding Yoda-0) and two bits, and is better than RDIS-3 for three bits and Aegis\_rw for three and four bits, respectively. However, it scales very poorly beyond a small number of errors, requiring more than double the storage space for a 512-bit block to correct ten faults. Aegis\_rw provides the best guarantee for a large number of faults (i.e., typically around 1-2%), however, it scales poorly to both small and moderate numbers of faults (i.e., \( \leq 1\% \)) and very large numbers of faults (i.e., \( \geq 3\% \)). This is due to the nature of Aegis, which requires a significant overhead to establish the partitioning through slopes. It also has a knee around \( \log(\text{block\_size}) \), at which point it has a sharper rise causing it to exceed Yoda. In particular, Aegis\_rw has a smaller overhead than Yoda when there are between six and 14 faults for a 512-bit block and between ten and 16 faults for a 1024-bit block.

The cost for Yoda increases linearly. As with traditional ECP, the slope for Yoda-0 is sharper as it requires 9 bits or 10 bits more (essentially an additional pointer) to tolerate an extra fault. The slopes for Yoda-1, Yoda, and Yoda-S are less sharp. This is due to the fact that with \( k \) pointers, Yoda-0 can guarantee protection of \( k \) stuck-at faults, while Yoda-1, Yoda, and Yoda-S can guarantee correction \( 2k+1 \) faults. When guaranteeing correction of few faults, the overheads of Yoda-1, Yoda, Yoda-S are similar, if not identical. As the fault guarantee increases, Small Yoda’s cost has the best scalability, but Yoda-1 and Yoda also scale well. In general, Yoda-1, Yoda, and Yoda-S require an additional 4.5, 4.9, and 4.2 bits, respectively, per additional guaranteed fault.

These results indicate that Yoda has a clear scalability advantage over ECC, SAFER, and RDIS. In addition, SAFER, ECC-1, RDIS, and Yoda can be implemented using logic in the memory controller, making a reason-
able comparison. In contrast, Aegis requires a series of large look-up tables for its implementation, because its encoding algorithm is considerably more complex than SAFER, ECC-1, RDIS, and Yoda. Moreover, Aegis requires potentially many tests of different irregular partitions to attain the required protection, whereas Yoda is deterministic and can determine the encoding directly. We include the fault-correction guarantee for comparison against Yoda, even though Aegis is not as lightweight an encoding scheme. Moreover, Yoda does provide a better fault-tolerance guarantee for small, moderate, and very large numbers of faults. Furthermore, while Yoda-S would require a look-up table for partitions with more than two groups, it still follows the same deterministic encoding algorithm as Yoda, making it much lighter weight than Aegis.

![Figure 8: Faults corrected before failure in a 4KB page for various fault mitigation schemes. The number of auxiliary bits per block is shown above its bar. For Yoda, the overhead is shown for regular and small Yoda, respectively.](image)

5. Evaluation

To evaluate the efficacy and cost-effectiveness of Yoda for tolerance to stuck-at faults, we experimentally compare Yoda approach with SECDED ECC, FNW, and three comparable overhead error recovery schemes, which were designed for stuck-at faults: ECP, SAFER, and RDIS. We assume that, by using a read-after-write method [24] or using a fault cache [15], SA-R and SA-W faults may be distinguished for encoding. To ensure the fairest comparison, we use fault-aware ECP with $k$ pointers (Yoda-0$k$). For SAFER we
Figure 9: Lifetime improvement (in terms of accesses) of a 4KB page for various fault mitigation schemes compared to an unprotected page. Auxiliary bits are shown consistent with Figure 8.

also provide equivalent fault information. Similarly, it is denoted as SAFER$_N$ where $N$ means the number of partition groups. For RDIS as before, we apply recursive partitioning three times to represent RDIS (RDIS-3) [14]. In the evaluation, we separate Yoda-1 from Yoda for reference as Yoda-1 is a powerful special case of Yoda. Small Yoda (Yoda-S) is equivalent to Yoda for fault-tolerance but with reduced space overhead at the expense of higher encoding and decoding complexity.

5.1. Experimental Methodology

In our evaluation, stuck-at faults can be tolerated in a data block for each of the schemes as follows:

1. For a Hamming code based error correction (ECC-1$_{64}$), one SA-W and any number of SA-Rs can be tolerated in each group (64 bits).
2. For FNW, there is no group that has both SA-W and SA-R bits in the group’s data.
3. For SAFER, after its distinct partitioning, there is no group that has both SA-W and SA-R bits.
4. For RDIS-3, after recursively partitioning three times, SA-R and SA-W bits are fully segmented.
5. For Yoda-$0_k$ (Yoda-0 with $k$ pointers), the number of SA-W bits is not more than $N$.
6. For Yoda-1\(_k\) (Yoda-1 with \(k\) pointers), if either the SA-R bits or the number of SA-W bits do not exceed \(k\), the stuck-at faults in the data block can be tolerated.

7. For Yoda-N\(_k\) (Yoda with \(N\) groups and \(k\) pointers), after inverting \(N\) groups, if the number of uncorrectable faults throughout all the groups is not more than \(k\), the stuck-at faults in the data block can be tolerated. This is the same as Small Yoda-N\(_k\) (Yoda-S \(N\) groups and \(k\) pointers), which has the same fault tolerance with fewer auxiliary bits.

We use Monte Carlo simulations to conduct our evaluations of fault-tolerance [15, 14, 13]. Similar to RDIS and SAFER, we also assume the auxiliary encoding bits are stored in a separate fault-free memory [14, 13]. For each cell in PCM memory, we assume its lifetime follows the normal distribution which has a mean value of \(10^8\) and a 25\% coefficient of variance. The data block size we examined is 512 bits, while we assume a 4KB operating system page. We assume writes are uniformly distributed over the whole memory which is consistent with the expectation from an effective wear leveling method [25, 26, 27]. Further, we assume a differential write operation is adopted to further minimize cell wear-out. In our simulation, we continuously issue page writes to the memory protected by different schemes until there is an unrecoverable fault. We record the average numbers of recovered faults in a 4KB page for the various recovery schemes and the average lifetime improvement of a 4KB page protected over that of an unprotected 4KB page.

To further compare the effectiveness of Yoda variants and the baselines

![Figure 10: Lifetime improvement per bit of various fault mitigation schemes.](image-url)
of ECC and FNW, we created a memory model with fault rates from $10^{-3}$ to $10^{-5}$. To assess the scenario that the auxiliary bits can be faulty, we developed a PIN-based simulator [28] to evaluate the fault tolerance capacity of the recovery schemes. The PIN simulator evaluates main memory writes by encoding or partitioning the data block and the auxiliary bits, and recording a fault if the value of any fault bit being written is opposite to its stuck-at value. This simulator was evaluated over the PARSEC benchmark suite [29], and the results shown in Section 5.2 are averaged over the suite. To model the stuck-at faults, a fault map, including fault bits stuck at ‘0’ or ‘1’, was created using a weighted Bayesian distribution to mimic the impact of process variation including typical spatial correlation of faults [30, 31]. In particular for this work we followed the model described in [30] to generate maps of weak cells for a 4GB PCM.

5.2. Memory Lifetime Evaluation

In Figure 8, we compare the average number of recovered faults by various Yoda variants against SAFER and RDIS-3. ECC and FNW were omitted because they had much lower capability. Note that for Yoda, we set the number of groups ($N$) to $2^{\log_2 k}$, where $k$ is the number of pointers, in order to have the number of partitions scale along with the number of pointers. From the figure, we see that Yoda tolerates more stuck-at faults with the same or even lower overhead capacity. For example, Yoda-8₉ tolerates 908
faults in a page by using 89 auxiliary bits per 512-bit data block, while Yoda-0, SAFER and RDIS spend 90, 91 and 96 auxiliary bits to tolerate only 371, 293 and 364 faults, respectively, per block. Small Yoda-8 provides further reduction, with the same 908 faults tolerated for only 76 auxiliary bits. Yoda-1 is the second most effective scheme (not including any Yoda-S schemes) that requires 91 auxiliary bits per block to achieve 824-fault tolerance in a page which is still lower than that of Yoda-8 with 2 auxiliary bits less per data block. Furthermore, Yoda-4 with 58 auxiliary bits per data block tolerates 533 faults in a page, which is larger than the numbers of tolerated faults for Yoda-0, SAFER and RDIS with a much higher overhead capacity. On average Yoda provides a 3×, 2.5×, 3× improvement over Yoda-0 (essentially an improved ECP), RDIS, and SAFER, respectively for similar overhead.

Figure 9 shows a similar trend of improvement as the number of tolerated faults in Figure 8, though the gaps between different recovery schemes is smaller. For example, Yoda-8 tolerates 70.4% more faults than Yoda-4, while the lifetime improvement brought by Yoda-8 is only 13.5% higher than that brought by Yoda-4. The reason is that, when approaching the end of the lifetime, the acceleration of cell failure rate quickly overruns the additional fault-tolerance. Nonetheless, Yoda still achieves a higher lifetime improvement with a similar or even less overhead capacity over the other schemes in this section, among which Yoda-1 is still the most cost-effective scheme. For example, Yoda-4 achieves 33.7%, 28.2%, and 4.2% larger life-
time improvements than Yoda-06, SAFER32, and Yoda-16, respectively.

For all the aforementioned schemes, with a higher overhead capacity, more faults can be tolerated and the lifetime can be improved. However, the contribution by each auxiliary bit provides diminishing returns in extending the lifetime, as shown in Figure 10. All the schemes (except RDIS) suffer significant depreciation, illustrating that to increase the fault tolerance of a scheme requires a sacrifice in the efficiency of the auxiliary bit overheads. Although Yoda is not immune to this decline in efficiency, there is still a clear advantage of Yoda over Yoda-0, SAFER and RDIS counterparts. Yoda is comparable to Yoda-1 in lifetime contribution per bit, while Yoda-S is more efficient than Yoda-1. For example, Yoda-110 and Yoda-89 achieved 12.2% and 12.7% lifetime improvement contributions per bit, while Small Yoda-89 obtained a 14.9% contribution per bit, corresponding to 2.7% and 2.2% contribution improvements over Yoda-110 and Yoda-89, respectively.

Figure 8 depicts the number of faults that cause a 4KB page to fail under different recovery schemes, while Figure 11 illustrates the probability of failure with different numbers of faults in a data block. The failure probability for SAFER and RDIS is obtained through Monte Carlo simulations, while the ones for ECP and Yoda variants are calculated by assuming that SA-W and SA-R have the same probability (both 50%) and they are distributed uniformly over different groups in the data block. ECP and Yoda-0 are not concerned with fault distribution, while Yoda-1 and Yoda take it into account. In this figure, before the number of stuck-at faults reaches the thresh-
olds that the recovery schemes can fully tolerate, the probability of failure remains zero. Yoda-1 and Yoda tolerate $k+1$ faults more than Yoda-0 when using $k$ pointers, so their failure probabilities begin to rise with $k+1$ more faults. RDIS-3 starts to lose its perfect protection capability at three faults, but its curve keeps a low increasing rate, making this scheme comparable to Yoda-0_{10} and Yoda-1_{10} on average. SAFER does not exhibit any advantage over the other schemes with a similar cost. Among all these recovery schemes, Yoda manifests a lower failure probability than the other schemes with an equivalent overhead, and that, as the space overhead increases, this gap becomes larger. Yoda-1 keeps its trend to beat Yoda-0, SAFER, and RDIS, illustrating its second position in fault tolerance.

5.3. Probabilistic Evaluation

In this evaluation, all possible data patterns in every row of the generated memory model are simulated and the probability of failure of a perfectly even data distribution are discovered. For the Yoda variants (Yoda-0, Yoda-1, and Yoda), if one error correction bit is added to protect the auxiliary bits, we donate them as Yoda-0_{ob}, Yoda-1_{ob}, and Yoda_{ob}, respectively.

Figure 12, Figure 13, and Figure 14 show the results of the faults mitigation schemes for the memory model at the fault rates of $10^{-3}$, $10^{-4}$, and $10^{-5}$, respectively. In these figures, the results for ECC-1_{64} are marked as a line because its overhead is constant. For the other schemes, the numbers of auxiliary bits required for each data block are provided above the bars. For each group of bars, the overheads for Yoda variants and FNW are similar as
the Yoda variants all use the same number of pointers and the overhead for FNW is given the maximum of the Yoda variants. The overheads for Yoda\_ob variants are always one bit per data block more than their counterparts.

Yoda\_ob and Yoda obtain a $11.2\times$ and $2.0\times$ lower error rate than Yoda-1\_ob and Yoda-1, respectively, while Yoda-1\_ob and Yoda-1 obtain a $25.6\times$ and $3.6\times$ lower error rate than Yoda-0\_ob and Yoda-0, respectively. Yoda-0\_ob, Yoda-1\_ob, Yoda\_ob, Yoda-1, Yoda-0, achieve $6.2\times$, $44.0\times$, and $6.7\times$ lower error rates, compared to Yoda-0, Yoda-1, and Yoda, respectively. In Figure 14, Yoda-1\_ob and Yoda\_ob were fault free with three (or more) pointers.

6. Conclusion

Phase change memory suffers from endurance limitations which is a challenge for its mass adoption. Technology scaling and process variation further exacerbate this problem and reduce the potential lifetime of the memory. In this paper, we proposed Yoda, which is compatible with the ECP technique proposed by Microsoft. Yoda substantially improves the lifetime of PCM by tolerating many more faults than ECP by adding a small number of encoding bits. Meanwhile, we proposed a compressed version Yoda-S that reduced the space overhead by tradeoff the advantage of encoding and decoding latency. For example, to tolerate 10 stuck-at faults, Yoda needs 49 auxiliary bits per 512-bit data block, while Yoda-S reduces the number by four using a 70-byte lookup table to encode and decode the grouping mode. Our simulations, including Monte Carlo simulations and the probabilistic evaluation, shown that Yoda significantly improved the efficacy and cost-effectiveness over ECP, SAFER, and RDIS. Moreover, for small, moderate, and very large numbers of faults, Yoda provides a better fault tolerance guarantee than the much more complex Aegis approach. Additionally, by using one extra auxiliary bit, all the auxiliary bits are self-protected. This technique can also be introduced to other fault-tolerance schemes without auxiliary bit protection.

7. Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No. 61332003), US National Science Foundation Graduate Research Fellowship (Grant No. 1747452), and SHREC industry and agency members and by the IUCRC Program of the National Science Foundation (Grant No. CNS-1738783).
Reference


Jiangwei Zhang received the M.E. degree in Electronic Science and Technology from National University of Defense Technology, Changsha, China, in 2014, where he is currently pursuing the PhD degree. From 2016 to 2018, he is a visiting PhD student with Department of Electrical and Computer Engineering, University of Pittsburgh, PA, USA. His current research interests include memory reliability, emerging memories and image processing.

Donald Kline Jr received his BS degree in Computer Engineering in the spring of 2015, and his MS in Electrical Engineering in 2017, both at the University of Pittsburgh. He is currently a NSF Graduate Research Fellow pursuing his Ph.D. in Electrical and Computer Engineering at the University of Pittsburgh under the guidance of Dr. Alex Jones and Dr. Rami Melhem. His research interests currently include computer architecture, emerging memories, reliability, compilers, and machine learning.

Dr. Liang Fang received the Ph.D. degree in computer science from National University of Defense Technology, Changsha, China, in 1995. He is currently a Professor with the School of Computer, National
Dr. Rami Melhem received a Ph.D. degree in Computer Science from the University of Pittsburgh in 1983. He was an Assistant Professor at Purdue University prior to joining the faculty of The University of Pittsburgh in 1986, where he is currently a Professor in the Computer Science Department which he chaired from 2000 to 2009. His research interests include Power Management, Parallel Computer Architectures, Real-Time and Fault-Tolerant Systems, Optical Networks and High Performance. Dr. Melhem is a fellow of IEEE and a member of the ACM.

Dr. Alex K. Jones received the MS and PhD degrees in 2000 and 2002, respectively, in electrical and computer engineering at Northwestern University, Evanston, IL, USA. He is a MCSI Sustainability Faculty Fellow, and Professor of Electrical and Computer Engineering and Computer Science (by courtesy) at the University of Pittsburgh, Pennsylvania. His research interests include compilation techniques for configurable systems and architectures, sustainable computing, and fault-tolerant computing. Dr Jones is a senior member of the IEEE and ACM.