2017 IEEE 35th International Conference on Computer Design

Yoda: Judge me by my size, do you?

Dramatically Improving Error Mitigation Through Small Increases in Encoding Bits

Jiangwei Zhang*†, Donald Kline, Jr †, Liang Fang*, Rami Melhem‡, and Alex K. Jones†
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, China*
Department of Electrical and Computer Engineering, University of Pittsburgh, USA †
Department of Computer Science, University of Pittsburgh, USA ‡
Email: jiz148@pitt.edu, dek61@pitt.edu, lfang@nudt.edu.cn, melhem@cs.pitt.edu, akjones@pitt.edu

***Abstract*—Phase change memory is a promising alternative to conventional memories such as DRAM due to its density and non-volatility. Unfortunately, reliability is still a challenge as limited write endurance, exacerbated by process variation, leads to increasing numbers of stuck-at faults over the memory's lifetime. Error-correcting Pointers (ECP) is a popular proposal to mitigate stuck-at faults by recording the addresses and the values of faulty bits in order to extend the memory lifetime. In this paper, we propose Yoda, a method to extend ECP with one or a small number of additional encoding bits in order to dramatically improve the effectiveness and guaranteed fault correction capability of ECP. 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.**

***Index Terms*—Emerging Memories, Reliability, Stuck-at-Fault**

I. 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]–[3]. In contrast, PCM is an attractive alternative due to its scalability, non-volatility, and high density [4]–[6]. 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 the phase-change process.

In this paper, we propose 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 by adding a 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 which is subject to endurance faults. The resulting correction is able to correct 3.0× the number of faults of ECP alone and 2.5-3.0× 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 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 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.

This work was supported by National Natural Science Foundation of China Grant No. 61332003 and US NSF Graduate Research Fellowship Grant No. 1247842.

II. Background

To tolerate stuck-at-wrong (SA-W) bits, there are generally three classes of approaches. The first class is partition-and-flip schemes, such as SAFER [7], RDIS [8], and Aegis [9]. These approaches attempt to partition a data block to separate stuck-at cells into different groups and use a flag bit to invert each group. If the stuck-at cell is SA-W in a group the group is inverted, but if it is stuck-at-right (SA-R), the group is not inverted. The second class is error-correcting pointers (ECP) and its related schemes [10]. 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-correction strategies such as error correction codes (ECC) [11] 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 [12], [13]. 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 they are probabilistic in nature. Thus, we focus on the three main categories for description and comparison.

1063-6404/17 $31.00 © 2017 IEEE
DOI 10.1109/ICCD.2017.68

III. 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 short for yielding optimized dependability assurance (Yoda) through one-bit inversion with alloted numbers-of-bits (Obi-wan). Furthermore, we propose Small Yoda, a method to decrease the encoding storage overhead of Yoda-N at the expense of more complex encoding/decoding logic.

The N in Yoda-N refers to the number of partition-and-flip groups the error-correcting pointers can be partitioned between. Thus, Yoda-0 extends ECP where it only points to stuck-at-wrong bits and Yoda-1 is a special case to optionally flip the entire data block (1 group) before pointing to the minimum of the SA-R and SA-W bits [14]. Yoda-N generalizes this idea to multiple groups.

A. Yoda-N

In Yoda-N, we partition the data block into N groups[1], similar to Flip-N-Write (FNW) [15]. 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.

In Figure 1, 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, including three SA-R bits and four SA-W bits. The data block is partitioned into two groups, one 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,

[1]Yoda refers to Yoda-N in the remainder of the paper.

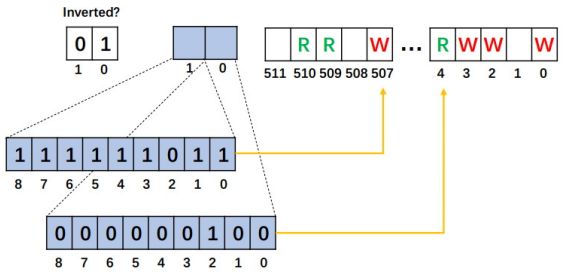

Fig. 1: Example of Yoda for N=2 (Yoda-2) with two pointers.

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).

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. An example of this compression is most clearly illustrated in the case for Yoda-2 with three pointers. 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.

IV. FAULT-TOLERANCE GUARANTEE

Figure 2 shows the required overhead to guarantee correction for a 512 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 (4.5 bits/fault), Yoda (4.9 bits/fault), and Yoda-S (4.2 bits/fault) 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.

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. 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. Regardless, Yoda provides 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.

V. Evaluation

To evaluate the efficacy and cost-effectiveness of Yoda for tolerance to stuck-at faults, we experimentally compare Yoda 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 [16], a fault cache [9], or a compact bit-level fault map [17], 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 also provide equivalent fault information. Similarly, it is denoted as SAFER$_N$ where N means the number of partition groups. For RDIS, we apply recursive partitioning three times to represent RDIS (RDIS-3) [8]. 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.

A. 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 k.
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. Small Yoda-N_k (Yoda-S N groups and k pointers) has the same fault tolerance with fewer auxiliary bits.

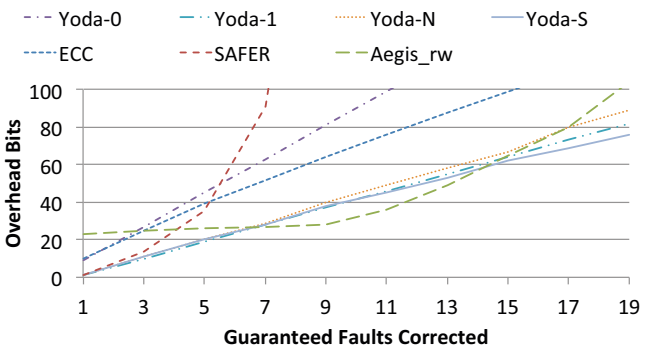

Fig. 2: Overhead bits to guarantee a particular fault-tolerance.

We use Monte Carlo simulations to conduct our evaluations of fault-tolerance [7]–[9]. Similar to RDIS and SAFER, we also assume the auxiliary encoding bits are stored in a separate fault-free memory [7], [8]. 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. We examined a 512-bit data block size with 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 [18], [19]. 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.

B. Memory Lifetime Evaluation

In Figure 3, 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 $2^{\lfloor log_2 k \rfloor}$, 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

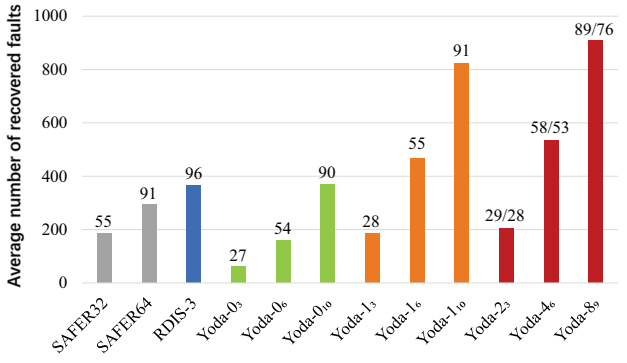

Fig. 3: Faults corrected before failure in a 4KB page. The number of auxiliary bits per block is shown above its bar. For Yoda, the overhead is shown for regular and small Yoda, respectively.

capacity. For example, Yoda-8_9 tolerates 908 faults in a page by using 89 auxiliary bits per 512-bit data block, while Yoda-0_{10}, SAFER and RDIS spend 90, 91 and 96 auxiliary bits and tolerate only 371, 293 and 364 faults, respectively, per block. Small Yoda-8_9 provides further reduction, with the same 908 faults tolerated with only 76 auxiliary bits. Yoda-1_{10} is the second most effective scheme (not including any Yoda-S schemes) that requires 91 auxiliary bits per block and tolerates 824 faults in a page, which is still less effective than Yoda-8_9 and requires two additional auxiliary bits per block. Furthermore, Yoda-4_6 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_{10}, SAFER and RDIS with a much higher overhead. On average Yoda provides a 3×, 2.5×, 3× improvement over Yoda-0 (essentially an improved ECP), RDIS, and SAFER, respectively for similar overhead.

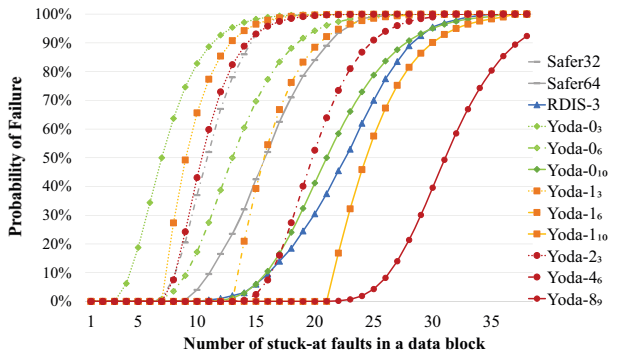

Fig. 4: Probability of failure with various stuck-at faults per data block under protection of different recovery schemes.

Figure 3 depicts the number of faults that cause a 4KB page to fail under different recovery schemes, while Figure 4 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 thresholds 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 a 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.

VI. 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. Our simulations show 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.

References

[1] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," *IEDM*, pp. 323–326, 2005.

[2] H. Zhang, N. Xiao, F. Liu, and Z. Chen, "Leader: Accelerating ReRAM-based main memory by leveraging access latency discrepancy in crossbar arrays," *DATE*, pp. 756–761, 2016.

[3] S. Li, P. Wang, N. Xiao, G. Sun, and F. Liu, "SPMS: Strand based persistent memory system," *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 622–625, IEEE, 2017.

[4] Y. Choi, I. Song, M.-H. Park, H. Chung, S. Chang, B. Cho, J. Kim, Y. Oh, D. Kwon, J. Sunwoo, *et al.*, "A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth," *ISSCC*, pp. 46–48, 2012.

[5] F. Xiong, E. Yalon, A. Behnam, C. Neumann, K. Grosse, S. Deshmukh, and E. Pop, "Towards ultimate scaling limits of phase-change memory," *IEDM*, 2016.

[6] G. W. Burr *et al.*, "Recent Progress in Phase-Change Memory Technology," *IEEE JETCAS*, Vol. 6, No. 2, No. 2, pp. 146–162, 2016.

[7] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," *MICRO*, 2010.

[8] R. Melhem, R. Maddah, and S. Cho, "RDIS: A Recursively Defined Invertible Set Scheme to Tolerate Multiple Stuck-at Faults in Resistive Memory," *DSN*, pp. 1–12, 2012.

[9] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," *MICRO*, pp. 433–444, 2013.

[10] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *ACM SIGARCH Computer Architecture News*, Vol. 38, pp. 141–152, ACM, 2010.

[11] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, Vol. 29, pp. 147–160, 1950.

[12] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," *HPCA*, pp. 222–233, IEEE, 2013.

[13] S. M. Seyedzadeh, R. Maddah, D. Kline, A. K. Jones, and R. Melhem, "Improving bit flip reduction for biased and random data," *IEEE Transactions on Computers*, 2016.

[14] R. Maddah, S. Cho, and R. Melhem, "Power of one bit: Increasing error correction capability with data inversion," *PRDC 2013*.

[15] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," *MICRO*, pp. 347–357, IEEE, 2009.

[16] L. Jiang, Y. Du, Y. Zhang, B. R. Childers, and J. Yang, "LLS: Cooperative integration of wear-leveling and salvaging for PCM main memory," *DSN*, pp. 221–232, 2011.

[17] D. Kline Jr, R. Melhem, and A. K. Jones, "Sustainable Fault Management and Correction for Next-Generation Main Memories," *IGSC*, 2017.

[18] M. K. Qureshi *et al.*, "Enhancing lifetime and security of phase change memories via Start-Gap wear leveling," *MICRO*, Vol. 14, p. 23, 2009.

[19] 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," *ACM SIGARCH computer architecture news*, pp. 383–394, 2010.