Designing Data Structures to Minimize Bit Flips on NVM

Daniel Bittman*, Matthew Gray*, Justin Raizes*, Sinjoni Mukhopadhyay*, Matt Bryson*,
Peter Alvaro*, Darrell D. E. Long*, and Ethan L. Miller*†
University of California, Santa Cruz* Pure Storage†
{dbittman, mtgray, jraizes, simukhop, mbryson, palvaro, darrell, elm}@ucsc.edu

***Abstract*—The advent of byte-addressable non-volatile memory technologies such as phase change memory (PCM) has spurred a flurry of research on topics including consistency and durability of data structures across power failures and optimizing systems for the low-latency nature of these technologies, while typically aiming to increase lifetime and reduce power consumption by reducing the number of *writes* to the non-volatile memory. However, in technologies such as PCM, it is *bit flips* that consume power and wear out cells, not writes. Thus, PCM controllers do not rewrite cells unless the cell changes value. However, this crucial optimization, reducing the number of bits flipped, has not been sufficiently explored for the rest of the hardware and software stack. We develop a framework for using the number of bit flips as the measure of "goodness" for a range of hardware and software techniques. We also introduce several simple and straightforward modifications to existing data structures that can reduce the number of bit flips over time, and profile use cases in which the approach with the fewest writes does not also minimize bit flips. Based on these findings, we discuss potential approaches that can further minimize bit flips, better optimizing hardware and software for non-volatile memory technologies such as PCM.**

***Index Terms*—power consumption, non-volatile memory, data structure design, bit flips**

I. Introduction

As byte-addressable non-volatile memories (BNVMs) become common [1, 2, 3], it is increasingly important that systems be optimized both to leverage their strengths and to avoid stressing their weaknesses. Historically, such optimizations include reducing the number of writes performed, either by designing data structures requiring fewer writes, or by using hardware techniques to reduce writes. However, for BNVMs such as phase-change memory (PCM), it is not only the number of *writes* that is important—it is the number of *bits flipped* by the writes.

BNVMs such as PCM suffer from two problems caused by flipping bits: energy usage and cell wear-out. Flipping a bit in a PCM consumes (relatively) significant power, so many controllers optimize by only flipping bits when the new value of the cell is different from the old value [4]. While this approach saves some energy, it cannot eliminate flips requested by software to modify data structures and write values in memory. An equally important concern is that PCM cells only support a limited number of write cycles. Unlike flash, however, PCM cells are written individually, so it is possible (and even likely) that some cells will be written more than others during a given period because of imbalances in values written by software. Reducing the number of bits flipped can thus both save energy and extend the life of PCM memory, yet existing approaches rarely attempt to optimize for reducing the number of bits that must be flipped to accomplish a task. Data structures designed for BNVM must take into account bits flipped by their updates in order to properly evaluate how they will affect BNVM.

We propose to attack this problem on multiple fronts, using both hardware and software techniques to minimize the number of bits flipped, even at the expense of sometimes *increasing* the number of *bytes* written to BNVM, by studying simple and widely applicable techniques that can reduce bit flips more than if we had focused solely on write reduction. On the software side, we evaluated a simple hash table design with varied implementation details and found that minimizing writes does not necessarily reduce bit flips and that we can apply some simple modifications to data structures to save a significant number of flips. We performed an analysis of XOR linked lists and found that their design saved more bit flips than would be expected by simply reducing the number of writes, further indicating that data structures can be designed with bit flip reduction in mind. We evaluate prior work on Write Efficient Memory [5], and discuss its applicability to BNVM. Finally, we outline some future work both in software and for hardware to further investigate bit flip reduction.

II. Background

Non-volatile memory technologies, including phase change memory (PCM) [1], Ferroelectric RAM (FeRAM) [2], and spin-torque transfer RAM (STT-RAM) [6], among others, promise to fundamentally change the design of our devices, operating systems, and applications. Although the technologies are starting to make their way into consumer devices [3], their full potential will be seen when they replace or exist alongside DRAM as byte-addressable non-volatile memory (BNVM). Such a memory hierarchy will allow the processor, and therefore applications, to access persistent storage with normal load and store instructions, bypassing the high-latency I/O operations of the operating system.

The design of data structures and systems for a particular technology must exploit the advantages of the underlying hardware while minimizing its disadvantages. For example,

data structures for disks are block oriented and avoid seeks while data structures for flash memory avoid random writes. Previous designs and evaluations of data structures and programming models for NVM [7, 8, 9, 10, 11, 12] typically exploit its direct-access nature while mitigating the slightly slower-than-DRAM nature of most BNVM technologies. At least in the case of PCM, treating BNVM like DRAM or block storage ignores two critical characteristics: asymmetric read/write power use, and avoiding redundant updates at the bit level [4, 13].

As an example, writes to PCM involve changing the phase of a material in each cell, which takes significantly more energy than reading the resistance of the resulting phase of the material. However, the PCM controller can avoid writing to a cell during a write if it already contains the desired value, meaning that the power use of a write is proportional not to the size of the write but to how many bits must flip in order to store the new value in the memory cells. Thus, we should be minimizing bits flipped by our data structures and algorithms instead of focusing on minimizing writes, as is more commonly done, because it not only reduces the energy consumption, but it also reduces the wear on the PCM cells.

Although many of the writes to BNVM come from writing data itself, a significant portion of writes come from updates to the metadata of a data structure used to manage said data. These updates are often in the form of pointers to nodes, bits indicating occupancy, or pointers to data. In an asymmetric access time memory such as PCM, it may be advantageous to avoid moving large pieces of data and instead move and update pointers to data, further increasing the importance of correctly designing the structure. Regardless, when writing data with an organizational data structure, the data itself must always be written regardless of the organizational structure chosen, meaning that if we are to reduce bit flips, we can look at both data and its organization in order to minimize bit flips.

While bit flips in BNVM have been studied previously, much of this work has focused on hardware encoding resulting in limited efficacy [14, 15, 16]. While hardware techniques are worth exploring, software techniques to reduce bit flips could be more effective because they can be designed with this goal in mind. Chen *et al.* [17] evaluate data structures on BNVM and argue that reducing bit flips is workload dependent and difficult to reason about, therefore we should aim to reduce writes as a close analogue for reducing bit flips. Although reducing writes is important and should be optimized for, Section III presents a counterexample to the claim that optimizing for writes is a close enough analogue while Section IV demonstrates an example where bit flips can be reduced beyond the savings from reducing writes. Furthermore, while the savings are workload-dependent, we believe there are techniques that can be employed that will often be effective in reducing bit flips.

A. Power Utilization of PCM and DRAM

Figure 1 shows the power consumption of PCM and DRAM as a function of the number of bits being flipped per second, using values from prior studies of memory systems [1, 17, 18, 19, 20]. While individual writes in DRAM require little power, the entire DRAM must be periodically refreshed (read and rewritten), resulting in a high idle power requirement but little added power from an increased write rate. PCM, on the other hand, requires much more energy to write a bit by flipping its value (around 50 pJ/b [21]), as compared to the relatively low additional overhead needed to write a DRAM page (1.17 pJ/b [1]), though the values vary in the papers. However, PCM does not require bits to be refreshed, so power consumption is proportional to the rate of bit flips, not to memory size, as for DRAM.

While the results in Figure 1 are only a rough estimate of the power usage behavior of these technologies, they clearly indicate that data structures designed for PCM need to consider techniques to minimize bit flips. DRAM power consumption is insensitive to the number of bits flipped (and even largely to write rate), since idle power dominates. PCM power consumption, on the other hand, is highly sensitive to the rate of bit flips—reducing bit flips directly reduces energy consumed. It is even possible for devices to reach the crossover point on the graph at which PCM becomes less power efficient than DRAM due to its high-energy cost to flip a cell.

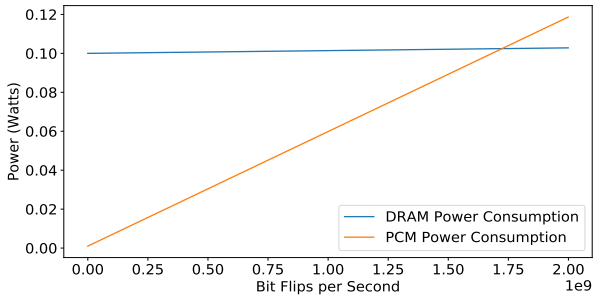


Fig. 1: Power use as a function of flips per second.

These initial results are relevant to Internet of Things (IoT) devices, since IoT devices may become a significant user of BNVM technologies [22, 23] for reasons including power and fast recoverability from power cycles. However, for asymmetric read/write power memory technologies, IoT devices may have to be careful to limit their bit flips per second in order to conserve power. Devices which collect large amounts of data and therefore write frequently to memory may find power usage increasing with BNVM depending on their access patterns. Thus, IoT devices stand to benefit significantly from bit-flip-aware systems and data structures.

B. Wear-out

Another significant advantage to avoiding bit flips is reducing memory cell wear-out. However, minimizing bit flips increases the risk that writes are biased (some of the bits updated more frequently than others). Take the case of writing and updating pointers—the upper bits likely remain the same, and the lower bits, with the exception of the lowest few bits, are likely to change. Thus, the middle bits in a 64 bit word

are more likely to wear out than the high or low order bits, unless something is done to spread the wear more evenly.

Unfortunately, a full remapping layer similar to a flash translation layer is infeasible for BNVM because the granularity of mapping is smaller and the added latency overhead would be large compared to the much shorter access time in BNVM. However, hardware techniques such as row shifting [24], content-aware bit shuffling [25], and start-gap wear leveling [26] have the potential to mitigate biased write patterns with low overhead, allowing BNVM to leverage bit flip reduction to reduce wear even if the result is that some bits are flipped more frequently than others. These techniques can work in tandem with the techniques presented in this paper.

III. Case Study: Avoiding Bit Flips in Hash Tables

Byte-addressable NVM brings up the possibility of memory-based hash tables as an indexing data structure in persistent storage [27]. The design space for in-memory persistent hash tables is large, with design questions including table length, collision resolution strategy, and key and value storage. To explore how these decisions might influence the number of bit flips during updates, we implemented several variants of a hash table and ran a randomly-driven workload on them, keeping track of the number of bits flipped along with the total number of bytes written by each operation. The counts collected include both metadata and data writes.

The hash table is an array of length l, where each element is a bucket containing s slots. When inserting a key/value pair, both 64 bit numbers, the key is hashed to a bucket, after which the insert function finds an empty slot and writes the key and the value into the slot. If a bucket fills up, l is doubled and the table is rehashed. Lookup is a matter of searching through a bucket's slots until the key is matched, and delete marks the slot corresponding to a found key as empty.

We test and compare three different design aspects of the hash table:

1) **Choosing slots**: When inserting, the simplest algorithm is a *find-first* method, where the first empty slot is used. An alternate, straight-forward optimization is to calculate the Hamming distance between each slot and the key/value pair being inserted and insert into the slot with the lowest distance, thereby minimizing the bit flips of the insert.
2) **Prime l versus power of 2**: A common design choice made when implementing a hash table is to make l a prime number, and instead of perfectly doubling when rehashing, assign l to a prime near to $2l$. However, one could also choose $l = 2^x$, and increment x when rehashing. While this makes rehashing and hashing somewhat cheaper, as an expensive division is not required, the resulting table is less randomly distributed. When optimizing for bit flips, such a design choice could be worthwhile, since particular key/value pairs may be more likely to hash to similar locations after rehashing.
3) **Used bitmap versus zeroed key**: To save space, the hash table could treat a zero key value as meaning the slot is empty. Alternatively, we can store a bitmap for the table of length sl bits, indicating which slots are empty. While this trades some cache locality for lookups, inserts can check many slots at once. The main benefit for bit flips comes from the fact that we no longer need to zero keys when rehashing or deleting so that future inserts may be able to flip fewer bits when inserting.

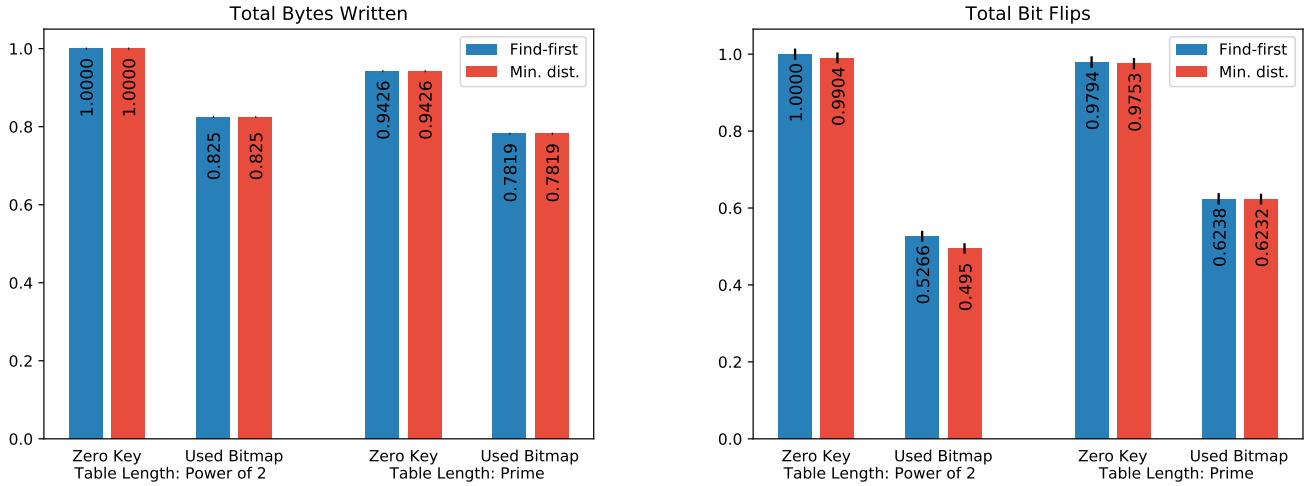
We simulated a large number of random inserts and deletes using biased random keys with pointers returned from `malloc` for values. The results are shown in Figure 2, where Figure 2a shows the total writes made to persistent memory in bytes, normalized to the power of two length, zeroed key, find-first variant of the hash table. Figure 2b shows the total bits flipped during the workload, normalized the same way.

The most striking difference in total bit flips is achieved by using a used bitmap instead of a zero key to signify slot occupancy. This is intuitive, since instead of writing eight bytes of zeros and flipping a large number of bits, we flip only one bit to clear the occupancy bit for the slot. However, an additional advantage comes when we insert a new key/value pair into a previously used slot because doing so overwrites eight bytes of non-zero bits, potentially avoiding some flips. Of course, this is only relevant when keys are biased; however, data often has a biased distribution, so we believe it to be an appropriate and relevant analysis.

Another result shows the difference between hash table length choices. The prime-length table had fewer total writes regardless of the other strategies chosen, but in the case of the used bitmap using a prime-length table *increased* the number of bit flips over the power of two length table. Although some [17] have argued that reducing writes is a good analogue for reasoning about and reducing bit flips, and while that is likely often the case, we find here that reducing writes does not always reduce bit flips. Not only does this hash table present a counter example, but it was the first data structure we tested. We speculate that the cause for this has to do with how the table length affects the calculation of the bucket when hashing a key, and that a prime-length table is more likely to redistribute keys to new locations after expanding the size and thereby reducing the effect of the used bitmap.

We can further reduce bit flips by performing a limited search across slots before inserting instead of picking the first empty slot. The effect is small—less than 1% in most cases, except for used-bitmap, power of two length case where it is significantly higher. This discrepancy corroborates our earlier observation that the power of two length table saves on bit flips because keys are more likely to map to prior buckets during a rehash. Although the savings are often small, when examined on a large table over the course of many operations, the reduction in bit flips builds up. The small price to pay is some additional work done during insert, but this is a small, tight loop on a limited number of slots and could be vectorized for a minuscule performance penalty.

Note that some of these effects may be dampened (not removed) by CPU caches and reordering. However, when updating data structures on BNVM, the programmer or system



(a) Count of written bytes during workload (normalized).

(b) Count of bit flips during workload (normalized).

Fig. 2: Analysis of bit flipping in hash tables.

must ensure that the data remain consistent across power cycles [12], meaning that the writes *must* make it to main memory eventually. For example, in the hash table each operation would be explicitly persisted to ensure transactional consistency. Methods for doing this range from using write-though caches [28] to cache-line flushing, but regardless of the method used, the updates will cause bits to flip in BNVM, so the results above will be relevant for data structure design.

Key Insights:

1) Minimizing writes does not always minimize bit flips. In the first data structure we examined, we found that bit flips are not always proportional to writes. While reducing writes is *correlated* with reducing bit flips, additional tests must be done to determine which techniques reduce bit flips the most, write-reduction approaches do not all reduce bit flips equally. This of critical importance because we should be designing for bit flip reduction as well as write reduction, particularly in systems where wear-out and energy usage are of primary concern.
2) Although a hash-table has comparatively little metadata, our techniques saved a significant number of bit flips. Especially in the case of small data, studying these techniques is worthwhile.
3) Using a separate bit to indicate that a value is zero may be worthwhile over actually zeroing memory. In our hash table tests we found that using a bit to indicate empty rather than zeroing the key resulted in a huge reduction in bit flips. This design choice is simple and widely applicable to many data structures.
4) Data placement matters and can reduce bit flips further, but such decisions are workload-dependent.

IV. Case Study: XOR Linked Lists

XOR Linked Lists [29] provide forwards and backwards traversal of a linked list while storing one fewer pointer than a traditional doubly-linked list with next and previous pointers. Instead of storing a pointer n to the next node and a pointer p to the previous node, each node stores $x = n \oplus p$. When traversing the list in either direction, one can XOR the previous node with the current node's x to recover the next node. XOR linked lists have similar performance characteristics to standard doubly linked lists while using less memory, but have an added penalty of requiring a reference to two adjacent nodes in the list to begin traversing or updating.

However, they have an additional benefit to bit flip reduction when used on NVM. Since each node stores only a single pointer-sized value instead of two, they cut down on the number of overall writes. Moreover, the value being stored is the XOR of two pointers, each of which are likely to have similar higher-order bits because allocators often return nearby pointers for similarly sized objects. This means that the stored value of x per node will have a significant number (half or more) of zeros in the higher-order bits of the value.

We compared XOR linked lists to doubly-linked lists by implementing both and counting the bit flips, pointer writes, and pointer reads during a workload of inserts and pops. The XOR linked list decreased both bit flips (by 52.4%) and writes (by 27.3%) over the doubly-linked list, and increased reads (by 42.9%), meaning XOR linked lists trade writes for reads during update operations. The data shows that while XOR linked lists save both writes and bit flips, the reduction in bit flips is nearly double the reduction in writes, reinforcing our claim that bit flips can be reduced using mechanisms that go beyond reducing the number of writes. The increase in reads compared to the reduction in bit flips is well worth it from the power usage standpoint (the reduction energy from bits flipped exceeds the increase of energy from extra reads [21]), and is advantageous for asymmetric access time memories.

V. Other Techniques, Algorithms, and Effects

Allocators and Pointers: Pointers are a commonly written piece of data in many data structures. When one pointer is

overwritten by another, the number of bits flipped depends on both the size of the pointed-to object and the allocator chosen. We performed a small study on the bit flips caused by overwriting a pointer using different allocators while varying the allocation size. While we do not have the space to reproduce the results here, the initial findings suggest that different allocators have significantly different behavior, motivating further investigation.

Sorting: Another investigation would be to examine sorting in-place in BNVM. Reducing movement is an important requirement among the many other reasons for choosing a given sorting algorithm, however different algorithms may result in different data movement patterns with different amounts of bit flips per move. While some have studied sorting on PCM [30], they focus on evaluating writes, not bit flips.

WEM codes: The concept of coding to reduce bit flips was first introduced by Ahlswede and Zhang [5] as the write-efficient memory (WEM) model, which has clear applications to BNVM. This paper examined Information Theoretic bounds on the trade-offs between length of codewords and number of bit flips. The bounds on binary codes are shown in Figure 3, showing the bounds on flip savings by adding coding bits.

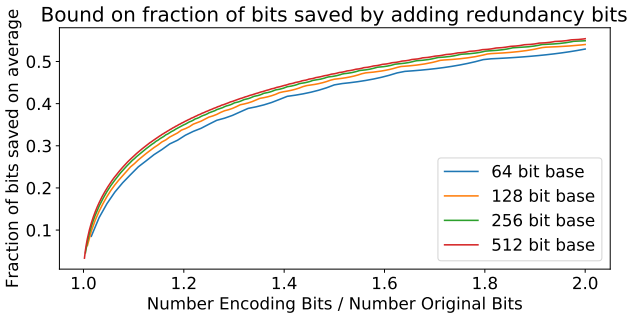


Fig. 3: Bounds on WEM code performance for unbiased data.

As shown in the analysis of hash tables, the functionality of a WEM code can be moved from hardware to software. This differs from previous work, which built WEM codes in hardware [14, 15, 16], in that it allows the use of pre-existing, unutilized memory rather than building additional cells.

Hardware Layers: While minimizing bit flips in software is promising, there are additional intermediate hardware layer effects to consider. Techniques such as encryption result in unavoidable bit flips because they result in random-looking data, meaning that half the bits are flipped on average by any given write. However, these techniques are not universally applied, meaning that targeted bit flip reduction is still possible.

Caching, flushing, and ordering constraints can also affect bit flips. Although data may be cached, any data which wishes to be persistent must find its way to BNVM. Thus, although certain writes can be coalesced and buffered, many of them will be flushed or not cached, so bit flip reduction techniques apply. We plan to develop a framework that includes these effects in order to study bit flip reduction further, but we expect that many of the techniques discussed here will apply.

VI. Conclusion

Minimizing bit flips is an important and challenging design goal for BNVM-aware data structures. Although minimizing writes *can* be a good proxy for minimizing bit flips, it is not always one; thus, we must be sure to incorporate bit flips as an explicit measurement when evaluating a data structure's performance and fit for BNVM. Even in the case where both writes and bit flips are reduced, simple design choices can allow the bit flip reduction to vastly exceed the expected reduction from simply reducing writes, indicating that we can improve over blindly reducing writes by either choosing which writes to minimize and when or by making those writes cheaper. Write reduction techniques may not reduce bit flips equally, so profiling bit flips directly is the only viable mechanism to determine the behavior of a data structure. Finally, we found that a few straight-forward implementation tweaks to a simple data structure resulted in significant savings, and many data structures can be similarly optimized.

Although we do not expect every data structure to be easily optimized for bit flips, and although we acknowledge that optimizing for bit flips can be difficult without knowing something about the workload, how it changes, and how data is distributed, there is research that can be done towards common and often applicable techniques, ranging from which data structures and algorithms to choose, to how to best encode data structures in memory to reduce bit flips, and to how these techniques will apply or change with intermediate hardware layers such as caching, write ordering, and wear leveling. We plan to further this work by doing these experiments on real hardware and full system simulators. These questions will require more research to begin to answer, but we believe they are necessary to answer as BNVM grows in popularity.

Acknowledgements

This research was supported in part by the National Science Foundation grant number IIP-1266400 and by the industrial members of the Center for Research in Storage Systems. The authors additionally thank the members of the Storage Systems Research Center for their support and feedback.

References

[1] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37. ACM, 2009, pp. 2–13.

[2] G. Fox, F. Chu, and T. Davenport, "Current and future ferroelectric nonvolatile memory technology," *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing, Measurement, and Phenomena*, vol. 19, no. 5, pp. 1967–1971, 2001.

[3] Intel Newsroom, "Intel and Micron produce breakthrough memory technology," July 2015.

[4] B. D. Yang, J. E. Lee, J. S. Kim, J. Cho, S. Y. Lee, and B. G. Yu, "A low power phase-change random access memory using a data-comparison write scheme,"

in *Proceedings of IEEE Int'l Symposium on Circuits and Systems*, May 2007.

[5] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Information and computation*, vol. 83, no. 1, pp. 80–97, 1989.

[6] T. Kawahara, "Scalable spin-transfer torque RAM technology for normally-off computing," *IEEE Design and Test of Computers*, vol. 28, no. 1, pp. 52–63, Jan 2011.

[7] J. Xu and S. Swanson, "NOVA: a log-structured file system for hybrid volatile/non-volatile main memories," in *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*, Feb. 2016.

[8] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A case for efficient hardware/software cooperative management of storage and memory," in *Proceedings of 5th Workshop on Energy-Efficient Design (WEED 2013)*, Jun. 2013.

[9] K. M. Greenan and E. L. Miller, "PRIMS: Making NVRAM suitable for extremely reliable storage," in *Proceedings of the Third Workshop on Hot Topics in System Dependability (HotDep '07)*, Jun. 2007.

[10] H. Volos, A. Jaan Tack, and M. M. Swift, "Mnemosyne: Lightweight persistent memory," in *Proceedings of ASPLOS '11*, Mar. 2011.

[11] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," in *Proceedings of ASPLOS '11*, Mar. 2011.

[12] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better I/O through byte-addressable, persistent memory," in *Proceedings of SOSP '09*, Big Sky, MT, Oct. 2009, pp. 133–146.

[13] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy, "Overview of candidate device technologies for storage-class memory," *IBM Journal of Research and Development*, vol. 52, no. 4, pp. 449–464, Jul. 2008.

[14] S. Cho and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," in *Proceedings of MICRO '09*. ACM, 2009, pp. 347–357.

[15] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," in *Proceedings of High Performance Computer Architecture (HPCA '13)*. IEEE, 2013, pp. 222–233.

[16] 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*, vol. 65, no. 11, pp. 3345–3356, 2016.

[17] S. Chen, P. B. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," *5th Biennial Conference on Innovative Data Systems Research*, pp. 21–31, January 2011.

[18] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proceedings of the 46th IEEE Design Automation Conference (DAC '09)*. IEEE, 2009, pp. 664–669.

[19] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: a circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, Jul. 2012.

[20] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 24–33, 2009.

[21] F. Bedeschi, C. Resta, O. Khouri, E. Buda, L. Costa, M. Ferraro, F. Pellizzer, F. Ottogalli, A. Pirovano, and M. Tosi, "An 8MB demonstrator for high-density 1.8 V phase-change memories," in *Symposium on VLSI Circuits 2004 Digest of Technical Papers*. IEEE, 2004, pp. 442–445.

[22] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the internet of things," in *Int'l Symposium on Low Power Electronics and Design (ISLPED '14)*. New York, NY, USA: ACM, 2014, pp. 375–380.

[23] H. Jayakumar, A. Raha, and V. Raghunathan, "Quick-recall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers," in *27th Int'l Conference on VLSI Design and 13th Int'l Conference on Embedded Systems*. IEEE, 2014, pp. 330–335.

[24] 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 36th Int'l Symposium on Computer Architecture*, 2009, pp. 14–23.

[25] M. Han, Y. Han, S. W. Kim, H. Lee, and I. Park, "Content-aware bit shuffling for maximizing PCM endurance," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 3, pp. 48:1–48:26, May 2017.

[26] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," in *Proceedings of MICRO '09*, 2009.

[27] B. Debnath, A. Haghdoost, A. Kadav, M. G. Khatib, and C. Ungureanu, "Revisiting hash table design for phase change memory," *SIGOPS Operating Systems Review*, vol. 49, no. 2, pp. 18–26, Jan. 2016.

[28] K. Bhandari, D. R. Chakrabarti, and H.-J. Boehm, "Implications of CPU caching on byte-addressable non-volatile memory programming," *Hewlett-Packard, Technical Report HPL-2012-236*, 2012.

[29] P. Sinha, "A memory-efficient doubly linked list," *Linux Journal*, vol. 129, 2004, http://www.linuxjournal.com/article/6828.

[30] M. V. Vamsikrishna, Z. Su, and K.-L. Tan, "A write efficient PCM-aware sort," in *Proceedings of Database and Expert Systems Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 86–100.