Memory-Awareness: Boosting NVM Energy Efficiency and Endurance

Saeed Kargar
University of California, Santa Cruz
skargar@ucsc.edu

Faisal Nawab
University of California, Irvine
nawabf@uci.edu

ABSTRACT

Non-volatile memory (NVM) technologies find extensive use in data storage solutions and battery-powered mobile and IoT devices. Within this domain, wear-out and energy efficiency pose critical challenges for NVM utilization. This paper delves into exploring these challenges, detailing why previous methodologies have fallen short in attaining optimal efficiency when addressing NVM limitations. To overcome these hurdles, we introduce a software-level memory-aware solution, Hamming-Tree, designed to strategically select the memory segment where a write operation is applied, effectively mitigating these challenges [12]. Our proposal, Hamming-Tree, outperforms existing state-of-the-art solutions significantly. This innovative approach is adaptable to various indexing data structures, including trees and linked lists. Moreover, it not only complements existing indexing methods but can also synergize with preceding hardware-based solutions to further enhance efficiency. Real-world evaluations conducted on an Optane memory device demonstrate that our proposed solution achieves a reduction of up to 67.8% in energy consumption.

1 INTRODUCTION

NVM technologies suffer from two main challenges that needs to be taken into consideration: (1) NVM write operations demand a significant amount of current and power. For flipping an individual bit in PCM, for instance, it requires around 50 pJ/b. This is significant when compared to writing a whole DRAM page which needs only 1 pJ/b [9, 11]. (2) NVM has low write endurance (the number of writes that can be applied to a segment of storage media before it becomes unreliable.) NVM write endurance is on the order of 10^8–10^9 writes, which is significantly lower than DRAM write endurance which is on the order of 10^{15} writes [9].

To overcome the energy consumption and write endurance problems in NVM, two approaches were developed. The first approach develops hardware-based write optimization techniques [3] that are mostly based on a *Read-Before-Write* (RBW) pattern. In RBW, a write operation w to a memory location x is always preceded with a read of x. The value to be written by w is compared with the old content of x, and only the bits that are different are written. This reduces the number of flipped bits, which reduces energy consumption and increases write endurance.

The second approach tackles the problem of energy consumption and write endurance by minimizing write amplification [2, 6, 14, 15]. However, these methods conflate the problem of write endurance with the problem of write amplification. Although in many cases a technique that leads to reducing write amplification has the side-effect of increasing energy efficiency and write endurance, this is not always the case as shown by prior work [8, 9]. As shown in [7, 9, 12], designing to reduce bit flipping can significantly increase energy efficiency and write endurance beyond what existing systems—that consider write amplification only—achieve [10, 12].

2 METHODOLOGY

Motivation. In this work, we identify a crucial opportunity to increase energy efficiency and write endurance that prior solutions overlooked—*memory-awareness*. Prior methods pick the memory location for a write operation arbitrarily (new data items select an arbitrary location in memory, and updates to data items overwrite the previously-chosen location.) This misses the opportunity to judiciously pick a memory location that is similar to the value to be written (in terms of their hamming distance), which can reduce the number of updated bits. Reducing the number of bit flips increases write endurance and reduces power consumption in many NVM technologies [5, 8].

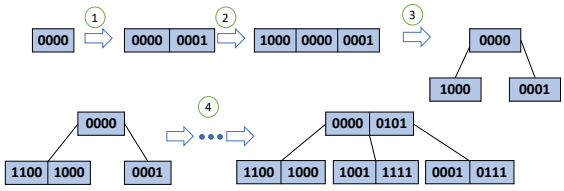

Figure 1: An example of how Hamming Tree is formed.

Hamming-Tree. We present a software-level memory-aware solution, Hamming-Tree. Hamming-Tree is implemented in software and does not suffer from the compute and space constraints of solutions implemented in the memory controller. Hamming-Tree is implemented as a data structure that achieves this objective by organizing free memory locations based on their hamming distance. Unlike a regular system, where updates are applied in place and there exists only one option to write the data, our proposed method determines the best existing free memory location in terms of hamming distance to minimize the number of bit flips. The intuition behind our proposed memory-aware-based method is that by placing the write operation in the right memory location that minimizes the hamming distance between the old and the new data, the number of bit flips can be significantly reduced.

Hamming-Tree uses an underlying indexing structure such as B-Tree or RB-Tree, to map available (free) memory locations of NVM. The only difference between Hamming-Tree and other tree-based indexing data structures is the way it compares the items and orders them; Hamming-Tree orders free memory locations according to their hamming distance. For example, in a regular B-Tree, number 1 is ordered before number 8 because 1 is less than 8. However, in Hamming-Tree the two numbers are compared and ordered based on the density of 0 and 1 (0/1) bits. Specifically, the intuition behind Hamming-Tree is to map memory locations that have the same density of 0/1 bits in different segments together. With this mapping, Hamming-Tree enables a new write to find a memory location that matches its density of 0/1 bits, which means that the selected memory location's bit-wise content is similar to the new write content. This leads to reducing bit flips as the new write is applied to a memory location with similar content.

Hamming-Distance-Based Comparison Function. The core idea of our proposed method is the Hamming-Distance-Based Comparison Function, which is a way of comparing and ordering blocks of string of bits (memory blocks) based on the distribution of 1's to 0's. The comparison function works

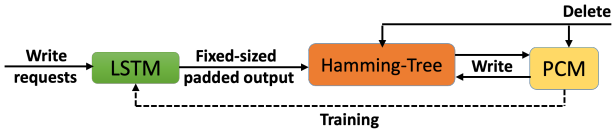

Figure 2: Block diagram of the proposed method.

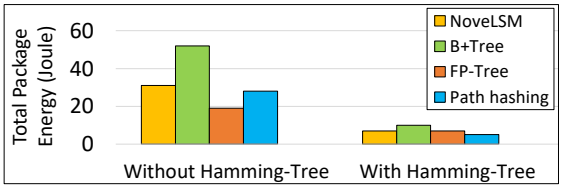

Figure 3: The average energy consumption per memory segment before and after augmentation by Hamming-Tree.

as follows: The comparison of two items d1 and d2 starts by measuring the density of 1's to 0's between the left half and the right half of the data items. A "Diff" function returns the difference of the number of 1 bits in the right segment to the number of 1 bits in the left segment (a positive Diff value represents that the number of 1's in the right segment is higher than the left segment, and vice versa). Diff is applied to both d1 and d2 and they are compared. If Diff(d1) is smaller than Diff(d2), then d2 is considered greater than d1 (this reflects that a higher density of 1's in the right segment translates to being bigger). Similarly, if Diff(d1) is greater than Diff(d2), then d1 is considered bigger than d2. If Diff(d1) and Diff(d2) are equivalent, then we recursively measure the density difference in the half with more 1's. This continues until we find a segment of an item that has a higher density compared to the corresponding segment of the other item, and then they are ordered accordingly. Figure 1 illustrates an example of Hamming-Tree which is built on a B-Tree of order m=3.

Hamming-Tree's Mapping Structure. Figure 2 provides an example of Hamming-Tree's mapping structure where Hamming-Tree is in DRAM and maps the available (free) memory locations in the NVM data zone, in which the actual data or K/V pairs are stored. Hamming-Tree does not need to be persisted in NVM because it can be reconstructed during recovery. When a DELETE, PUT, or UPDATE operation is applied, Hamming-Tree is traversed and updated accordingly to perform the operations and find the best free memory location.

Hamming-Tree's Padding Strategy.

In our upcoming work, we plan to address varying memory segment sizes using a data padding strategy employing an LSTM model to generate meaningful padded data similar to [7]. In our envisioned system, relying on fixed memory segments of size w, we aim to handle cases where input data (p) is smaller than the segment size, using padding (q=w-p bits) to extend it accordingly. The objective will be to align smaller inputs with similar items in the system. It is worth noting that the padded q-sized portion will serve for clustering purposes and won't be stored; only the actual p-sized data will be retained. Also, utilizing a Deep Q-Learning model for generating bit-wise meaningful padded data could provide a dynamically adapting real-time solution for addressing varying memory segment sizes [1].

3 RESULTS

In this paper, we use an energy profiler named Perf, which is a part of the Intel's RAPL interface [4, 13] and a performance analysis tool. Perf provides the collection of energy measurements from various components of a computer system such as: cores, Intel's GPUs, package (all the core and un-core components), DRAM, total power consumption of a node, and so on.

Figure 3 illustrates the performance of the indexing methods, such as B+Tree [2], Path Hashing [15], FP-Tree [14], and NoveLSM [6]), in terms of the amount of energy they consume in two different ways: before plugging Hamming-Tree, and after plugging Hamming-Tree. When not plugged to Hamming-Tree, B+-Tree has the worst energy consumption because, in a regular B+-Tree, the items in leaf nodes need to be sorted, which increases the number of movements and bit flips. After we tested the performance of the methods, we plugged them to Hamming-Tree and repeated the same tests. After plugging each method to Hamming-Tree, their performance—in terms of energy efficiency—improves by up to 91% by preventing a lot of unnecessary bits from being flipped.

4 CONCLUSION

Using a pluggable approach, Hamming-Tree reduces write-endurance issues by strategically directing write operations to memory locations, minimizing bit flips. This method seamlessly integrates DRAM-optimized data structures, like LSM-Tree and B+-Tree, into NVM systems without compromising performance or write-endurance. It also enhances the write endurance of current NVM data structures.

REFERENCES

[1] M. Andalibi, M. Hajihosseini, S. Teymoori, M. Kargar, and M. Gheisarnejad. A time-varying deep reinforcement model predictive control for dc power converter systems. In *2021 IEEE 12th International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, pages 1–6. IEEE, 2021.

[2] S. Chen and Q. Jin. Persistent b+-trees in non-volatile main memory. *Proceedings of the VLDB Endowment*, 8(7):786–797, 2015.

[3] S. Cho and H. Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 347–357, 2009.

[4] P. Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2(11), 2011.

[5] J. Huang, Y. Hua, P. Zuo, W. Zhou, and F. Huang. An efficient wear-level architecture using self-adaptive wear leveling. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[6] S. Kannan et al. Redesigning lsms for nonvolatile memory with novelsm. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 993–1005, 2018.

[7] S. Kargar, B. Gu, S. A. Jyothi, and F. Nawab. E2-nvm: A memory-aware write scheme to improve energy efficiency and write endurance of nvms using variational autoencoders. *Proceedings of the 26th International Conference on Extending Database Technology (EDBT)*, 2023.

[8] S. Kargar, H. Litz, and F. Nawab. Predict and write: Using k-means clustering to extend the lifetime of nvm storage. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 768–779. IEEE, 2021.

[9] S. Kargar and F. Nawab. Extending the lifetime of nvm: challenges and opportunities. *Proceedings of the VLDB Endowment*, 14(12):3194–3197, 2021.

[10] S. Kargar and F. Nawab. Hamming tree: The case for memory-aware bit flipping reduction for nvm indexing. In *CIDR*, 2021.

[11] S. Kargar and F. Nawab. Challenges and future directions for energy, latency, and lifetime improvements in nvms. *Distributed and Parallel Databases*, 41(3):163–189, 2023.

[12] S. Kargar and F. Nawab. Hamming tree: The case for energy-aware indexing for nvms. *Proceedings of the ACM on Management of Data (SIGMOD)*, 1(2):1–27, 2023.

[13] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(2):1–26, 2018.

[14] I. Oukid et al. Fptree: A hybrid scm-dram persistent and concurrent b-tree for storage class memory. In *Proceedings of the 2016 International Conference on Management of Data*, pages 371–386, 2016.

[15] P. Zuo and Y. Hua. A write-friendly hashing scheme for non-volatile memory systems. In *Proc. MSST*, 2017.