7-2020

Content-Aware Reduction of Bit Flips in Phase Change Memory

Arockia David Roy Kulandai

Thomas Schwarz

Content-Aware Reduction of Bit Flips in Phase Change Memory

Arockia David Roy Kulandai
Marquette University, Milwaukee, WI
Thomas Schwarz
Marquette University, Milwaukee, WI

Abstract:

The energy costs of Phase Change Memory (PCM) depends almost completely on the number of bits written per time unit. By using an encoding, we can reduce the number of bit flips when overwriting low-entropy data with low-entropy data. This is achieved by using a frequency table for bytes in classes of data to select the encoding. Using various corpora of mainly HTML files, we show that we can reduce the number of bit flips by about 0.5 bit flips per byte.

SECTION 1 Introduction

Phase Change Memories combine high density and low costs with access times about as fast as Dynamic Random Access Memories (DRAM). They are byte addressable and their endurance is orders of magnitude better than that of Flash.

PCMs store data in the phase of a chalcogenide alloy (GST: Ge2Sb2Te5) that switches between high or low resistivity as it switches between an amorphous and crystalline phase. We write by using a write current that heats the alloy for a short time and whose strength and shape determines the phase after cooling. Unlike DRAM, refresh operations are not necessary. Energy is only used for reading and much more prominently, for writing.

While the first PCM devices are now commercially available, the technology continues to evolve. In the future, they might be a single component unifying the functions of memory and storage. In this case, their limited endurance and their energy use become issues. A simple, but effective hardware scheme called Partial Write only writes a cell if its content changes. Thus, it is not the number of overwrites of a byte but of individual bits that matter. Bittman *et al.* have shown that even very simple changes to a data structure can not only lower the number of bytes written but disproportionally lower the number of bitflips [2], [3]. Here, we take their work one step further.

Our contribution starts with the observation that the way we store data also influences bitflips. We show that a very low-cost byte-for-byte translation based on byte frequency in broad categories of data can save about half a bitflip per byte when overwriting low entropy data (such as HTML, UTF-8, or ASCII files) with low entropy data.

SECTION 2 Related Work

Bit flip pressure can be alleviated in various manners, including how PCM is integrated architecturally in the system, or how it is used, for instance, by a compressing file system.

A large number of proposals at the hardware level exists. First, we can lower the number of bitflips by avoiding unnecessary writes. Data Comparison Write (DCW) only overwrites cells whose content has changed [10]. Second, we can use various compression schemes. At the hardware level, we can use Frequent Pattern Compression (FPC), used by Alameldeen and Wood to compress data in cache lines, but equally applicable to non-volatile memories [1], [7] using additional meta-data bits per word.

Other schemes encode data to be written differently based on previous contents. These schemes use additional meta-bits. Cho and Lee's *Flip-N-Write* (FNW) stores a word sometimes by inverting all its bits (indicated by setting an additional bit) [4].

Combining FNW and FPC and also storing words sometimes reverted constitutes Palangappa and Mohanram's Adaptive Flip-N-Write (aFNW) [8]. Their proposal also evens out bitflip pressure over all bits in a word through rotation, at the costs of additional metadata bits. Jalili and Sarbazi-Azad observe that bit-flips are not distributed uniformly over a word. Different applications generate different sets of hot locations over a word. By analyzing a block, their system, Captopril, determines by inspection which pattern prevails in a block about to be written and then use FNW only on those hot locations [6]. A much more thorough survey is given by Rashidi, Jalili and Sarbazi-Azad [9].

We already mentioned Bittman *et al.* discovery that even minute changes to the operating system and data structures can lower bitflip pressure [2], [3]. We are not the first to propose domain specific alterations. Fang and colleagues improve the lifetime of PCM used as a buffer for multimedia applications using Periodical Data Reversion (PDR), which is also part of our proposal, and Error-Tolerance Evaluation (ETE) if PDR is not used [5].

SECTION 3 Content Dependent Encoding

In high-entropy content (such as highly compressed or encrypted files), each bit has value 0 or 1 with about equal probability independent of the position of the bit within the word or in the storage system itself. If we overwrite high-entropy content with other high-entropy content, then we expect a bit to flip with probability close to 0.5. The same holds true when we overwrite high-entropy content with low-entropy content or vice versa. This is not true when we overwrite low-entropy content with low-entropy content. Fig. 1 shows the results of an experiment that includes the effects of our scheme. For the time being, only the first two columns are relevant. They show how often a given bit changes when overwriting a corpus in English with a corpus in German and vice versa., We see that the changes are concentrated almost completely in the bits 0, 1, 2, 3, 4, and 6 of a byte starting with the least-significant bit.

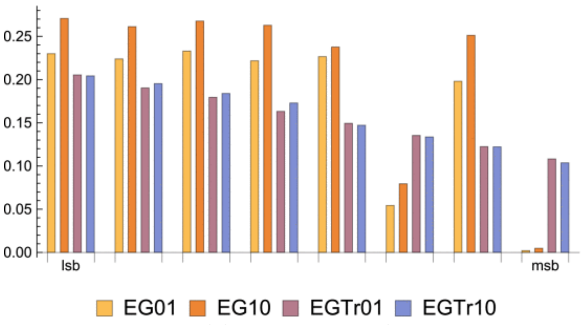

Fig. 1. Number of flips per byte for individual bits in a file when overwriting an English corpus with a German corpus. The left two bars are for raw and the right are for translated files. EG01 gives the proportion of bitflips from zero to one when overwriting an unaltered English file with a German file, EG10 the number of bitflips from one to zero and Tr refers to the same situation, but with both files translated.

We use a simple byte-to-byte translation scheme for low-entropy files. The particular translation only depends on the type of file (such as an HTML file in Tamil). A file-system can recognize the file type broadly by looking at extensions or at the "magic number" that some files such as JPEG files implement in their preamble. For HTML files, a tag such as <html lang="hi"> specifies the language. Even if a file is misclassified, some savings are likely (Section 4.3). However, most users in the English speaking world will store no or very little foreign language pages in their temporary internet cache. While it might appear that our scheme can only be effective if we overwrite a file with a file of the same type, this is not the case as our experimental results show. To exploit it, a file system labels data as either high or low entropy and stores them appropriately. For each file type, we need to store a translation table, which can be implemented as a byte array of 256 entries, so that it has size 256B. Translation processes the data as a byte array, replacing each byte by the one given in the translation table.

The goal of the translation table is to insure that overwriting a frequent byte with another frequent byte results in few bit-flips. We implement it as a simple dictionary. The keys of the dictionary are the bytes ordered by their frequency according to a language sample. For example, on one of our systems, there would be one translation table and its inverse each for English, Hindi, Gujarati and Tamil low-entropy text. To define the translation table, we also introduce an ordering for the bytes into which we translate. For this ordering, we write all bytes in binary $b_7b_6b_5b_4b_3b_2b_1b_0$ and then we associate to it the value $\sum_{\nu=0}^{7}(50+\nu)b_\nu$. We then associate the most frequent byte with the byte with the lowest value. The magic number 50 can be replaced by other constants of similar magnitude. Ordering bytes according to this value guarantees that the zero byte comes first, then bytes with a single one-bit, then bytes with two one-bits set, and so on, and within each group, bytes with one-bits closer to the LSB have lower value.

As a result, our translation table will assign the most frequent byte as 0000 0000, the second most-frequent byte as 0000 0001, the third most frequent byte as 0000 0010, and so on. The tenth most frequent byte is encoded as 0000 0011. The least frequent byte is encoded as 1111 1111. If we want to, we can create a different translation table by exclusive-or-ing a constant byte to all encoding.

SECTION 4 Experimental Verification

For the experimental verification, we created several different corpora. A first set consisted of HTML pages, separated by different languages, namely English, German, Tamil, and Hindi. The last two languages do not use a Latin alphabet, but a typical webpage in the language uses a mixture of Latin letters for metadata such as html tags and Tamil script and Devanagari for content in Tamil and Hindi. The majority of our collections were newspaper articles as we believe that they are typical of modern websites with a plethora of javascripts. In addition, we included corpora consisting of pdf files and one corpus each of English and German text files consisting of books downloaded from the Project Gutenberg and encoded in UTF-8. All of our corpora had a size of at least 10 MB, but smaller samples would not have lowered the confidence intervals visibly.

We calculated our translation array from a subset of one of the corpora in the language group. This means that the two Tamil corpora were encoding using the same translation (based on a frequency count of part of one corpus) and the same holds for the two German and the three English corpora we used. We do not display confidence intervals in our results because they are too small to show, but we shuffled the files in the corpora (so that no block of bytes remained in the same place) in order to obtain twenty different versions of each corpus.

4.1 Effect of Translation on Bits Within a Byte

We first illustrate the working of our encoding between two different corpora, one in English, the other one in German, Fig. 1. There, the number of flips depends very much on the location of the bit within the byte. Not only does translation lower the total number of bit flips, it evens them out, though there is a steady decrease from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Below, we show how this can be used to almost equalize the number of bit flips within a byte.

4.2 Comparison and Evaluation With Flip-N-Write

We counted the bit flips of the first 10 million bytes in each file against each other. We used four combinations, depending on the use of translation and the use of Flip-N-Write. Recall that Flip-N-Write adds one flip bit to each word on which it is evaluated. Typical word sizes are 32b and 64b, though its effectiveness goes down with larger word size. Fig. 2 shows the results. The savings of Flip-N-Write (with 32b words) are around 0.08 bit flips (2.5%) for original files and 0.018 bit flips (.7%) for translated files. In contrast, the savings of translation are 0.54 bit flips (18%). Unlike FNW, translation does not cause a storage overhead, but like FNW, there is a translation step between the raw data and the form in which data is stored. As we only process bytes, the result for translation does not depend on the word size.

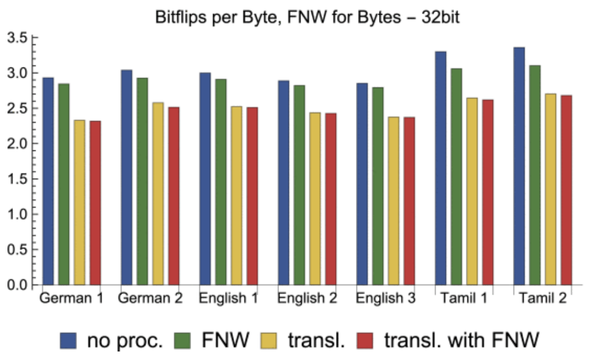

Fig. 2. Bit flips per byte for low-entropy corpora. We compare the bit flips per byte for original files, for original files with Flip-N-Write, for translated files, and for translated files with Flip-N-Write.

4.3 Inter-Corpora Comparisons

While much of the globe is uni-lingual, some societies are not. Many users there would store documents in several languages and it would be very cumbersome for a file system to maintain different storage areas in PCM for all different types. Fortunately, beyond the distinction between low and high entropy data, our following experiment shows this to be unnecessary. For each corpus in our collection, we created 20 concatenations of the same files, but in different orders. No portion of one of these 20 concatenations would be at the same position in a different concatenation. We then calculated the number of bit flips when overwriting the beginning 10 million bytes of one file with the same stream of the other file. Fig. 4 gives the results. As we can see, the number of savings are reasonably independent of the language used. Thus, a multi-lingual user can use our system to her advantage, provided of course that the system can attribute a file to a given dominant language in order to select the correct translation table. However, when we encoded the Tamil corpus with the English translation table, the results were only marginally worse, indicating that errors in language attribution are not grave.

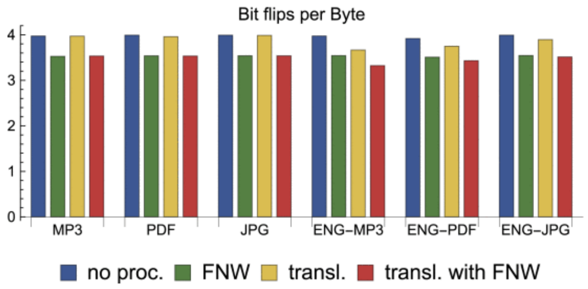

Fig. 3. Bit flips per byte with high to high and high to low entropy data.

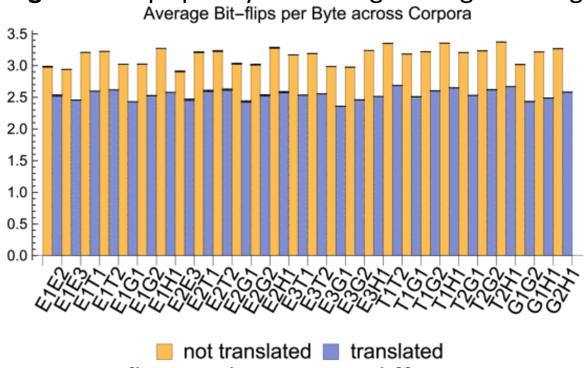

Fig. 4. Bit flips per byte across different corpora. The first and third letter denotes the language (English, German, Tamil, or Hindi) and the digit the corpora number.

4.4 High Entropy Data

When overwriting with or over random data (such as compressed or encrypted files), each byte is equally likely to be the exclusive-or of the old and the new byte, and we have to expect that on average four of the eight bits in a byte are overwritten. Many high-entropy data is not quite random, and to our surprise our results in Fig. 3 show that some situations benefit from translation, even when overwriting high entropy data with low-entropy data. The first three columns show the bit-flips resulting from overwriting a corpus with files of type mp3, pdf (in English), and jpg with the same corpus, but in different order, to simulate overwriting high entropy data with high entropy data. The last three columns show the number of bitflips per byte when overwriting these corpora using one of our corpora with html files in English. In our experiments, not limited to the ones depicted, found that translating brings small benefits. In general, translation stays close to the theoretical level (of 4 bit-flips per byte), whereas Flip-N-Write manages to save almost half a bit flip per byte. We use one FNW meta-bit for words of 32 bits, so that the costs are 3.1% storage overhead.

4.5 Balancing Bit Flips

Bit flips not only cost energy but also affect endurance. For endurance the total number of bit flips is not as important as the maximum bit flips in a cell. Therefore, we want to balance the expected number of bit flips in a word, or, in our case, in a byte, as the bit flips in a byte are independent from the bit flips of neighboring bytes. As we have observed previously, the expected bit flips between translated corpora shows an almost linear decrease between the Least Significant Bit (LSB) and the Most Significant Bit (MSB). Therefore, if at about the half-time of the PCM, the area containing low-entropy data reverses the bytes to be stored, the expected number of bit flips should be almost independent of the position of the bit inside a byte (and by extension, inside a word). We verified this experimentally. Fig. 5 gives the result of averaging the number of bit flips in bit i, $i \in \{0,1,\dots,7\}$, with the bit flips in bit $7-i$, i.e., assumes that we first overwrite one corpus with another in normal order and then overwrite the same corpus with another in reverse order of bits. Since we assume that such a reversal is relatively rare (maybe with a major update of the operating system once a year), the effects of overwriting a corpus in normal order with one in reversed order are not of interest. The results show that the strategy of periodically reversing the direction of the bytes balances the bit flip load very well for translated corpora, but has very limited success for raw low-entropy byte streams.

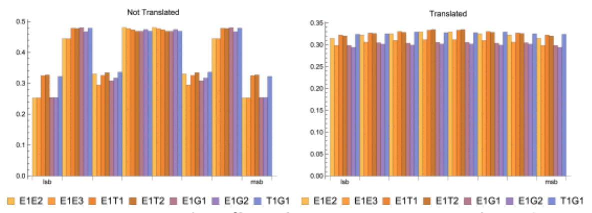

Fig. 5. Average bit flips between original and reversed arrangement for the raw and the translated corpora.

4.6 Results

Translation reduces the number of bit flips per byte by slightly more than 0.5 as long as we overwrite low-entropy data (HTML, UTF-8 text files) with other low-entropy data. Overwriting high entropy data (MPEG, JPG, or compressed files) with each other or with low-entropy data shows bit flips per byte numbers close to the theoretical value of 4. The benefit of translation arise independent of natural language. Translation even has marginal benefits for some high entropy data (using the English translation table). Flip-N-Write is outperformed by Translation. Using Flip-N-Write with translation has very little effect. It would appear that Flip-N-Write only has a place with high-entropy data.

SECTION 5 Conclusions

New and evolving technologies for persistent memory such as PCM have costs associated with each bit flip. In the case of PCM, the number of bit flips over the life-time of the device is limited and the energy consumption of a PCM device is almost completely dependent on the number of bit flips per second.

We have presented a scheme based on content dependent encoding of bytes that saves bit flips (from about 3 bit flips per byte to about 2.5 bit flips per byte) when overwriting low-entropy files with other low-entropy files, independent of their original language. The translation is byte for byte and therefore faster and simpler than compression, the alternative technology for saving bit flips based on content. Our scheme outperforms Flip-N-Write to render it ineffective for this type of data.

References

1. A. Alameldeen and D. Wood, "Adaptive cache compression for high-performance processors", Proc. 31st Int. Symp. Comput. Archit., pp. 212-223, 2004.
2. D. Bittman et al., "Designing data structures to minimize bit flips on NVM", Proc. IEEE 7th Non-Volatile Memory Syst. Appl. Symp., pp. 85-90, 2018.
3. D. Bittman, D. Long, P. Alvaro and E. Miller, "Optimizing systems for byte-addressable NVM by reducing bit flipping", Proc. 17th USENIX Conf. File Storage Technol., pp. 17-30, 2019.
4. S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance energy and endurance", Proc. IEEE/ACM Intern. Symp. Microarchit., pp. 347-357, 2009.
5. Y. Fang, H. Li and X. Li, "Lifetime enhancement techniques for PCM-based image buffer in multimedia applications", IEEE Trans. Very Large Scale Integr. Syst., vol. 22, no. 6, pp. 1450-1455, Jun. 2014.
6. M. Jalili and H. Sarbazi-Azad, "Captopril: Reducing the pressure of bit flips on hot locations in non-volatile main memories", Proc. Conf. Des. Automat. Test Europe, pp. 1116-1119, 2016.
7. L. Jiang, B. Zhao, Y. Zhang, J. Yang and B. Childers, "Improving write operations in MLC phase change memory", Proc. IEEE Int. Symp. High-Perform. Comput. Archit., pp. 1-10, 2012.
8. P. Palangappa and K. Mohanram, "Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories", Proc. 25th Ed. Great Lakes Symp. VLSI, pp. 221-224, 2015.
9. S. Rashidi, M. Jalili and H. Sarbazi-Azad, "A survey on PCM lifetime enhancement schemes", ACM Comput. Surv., vol. 52, no. 4, pp. 1-38, 2019.
10. 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", Proc. IEEE Int. Symp. Circuits Syst., pp. 3014-3017, 2007.