An Offline Frequent Value Encoding for Energy-Efficient MLC/TLC Non-volatile Memories

Ali Alsuwaiyan Kartik Mohanram

Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh

asa78@pitt.edu kartik.mohanram@gmail.com

ABSTRACT

This paper describes a low overhead, offline frequent value encoding (FVE) solution to reduce the write energy in multi-level/triple-level cell (MLC/TLC) non-volatile memories (NVMs). The proposed solution, which does not require any runtime software support, clusters a set of general-purpose applications according to their data frequency profiles and generates a dedicated offline FVE that minimizes write energy for each cluster. Results show that the write energy reduction of evaluation sets—using FVEs generated for training sets—are close (equal) to the best known solution for MLC (TLC) NVM encoding; however, our solution incurs a memory overhead that is $16\times$ ($5.7\times$) less than the best comparable scheme in the literature for MLC (TLC) NVMs.

Categories and Subject Descriptors: B.3.2 [Hardware]: Primary Memory

Keywords: Non-volatile Memories, Encoding, Energy

1. Introduction

It is acknowledged that DRAM scalability will slow beyond the 22nm feature size due to its considerable static and dynamic power requirements [1]. Non-volatile memory (NVM) technologies, such as phase change memory (PCM) [2] and resistive RAM (RRAM) [3], are being actively investigated to replace DRAM. Such technologies are highly scalable, have extended data retention times, and consume negligible static power. NVM technologies, however, have their own limitations, e.g., higher write energy, higher write latency, and limited endurance in comparison to DRAM [2, 4, 5].

Data encoding solutions (e.g., [6–8]) explicitly reduce write energy in NVM technologies. They also implicitly improve NVM endurance [9], and (in some cases) improve access latency [10]. Frequent value encoding (FVE) was originally introduced to encode data and address buses [11–13]. FVE can achieve high reductions in NVM write energy by mapping frequent values (words) into low energy codewords. Recently, FVE has been proposed to encode single-level cell (SLC) memory words in PCM [14]. The authors investigated both static (offline) and dynamic (online) encoding. They report that although offline encoding (originally introduced as “find-once for a given program” in [11]) achieves higher energy reductions on average, it requires compiler and operating system support, which is not desirable in practice. In contrast, online encoding may be better, but it requires a non-trivial online profiling effort that may impact system performance. However, to the best of our knowledge, FVE has not been applied to MLC/TLC NVMs, where

a single physical cell can store more than one logical bit to realize density and cost advantages [15, 16]. Since MLC/TLC NVMs have higher write energy and access latency as well as lower endurance in comparison to SLC NVMs, there is strong motivation to develop data encoding and wear-leveling techniques for MLC/TLC NVMs [9, 10, 17–19].

In this paper, we present an offline FVE for MLC/TLC NVMs that achieves an average write energy reduction approaching, and sometimes exceeding, that of optimal offline encoding [11]. The proposed method, which does not require compiler or operating system support, is based on finding multiple optimal FVEs. Each FVE is derived by aggregating the data frequency profiles of a group of compatible applications. To the best of our knowledge, this is the first work that presents a feasible version of an offline FVE approaching the average energy reductions of optimal offline FVE. This work makes the following contributions:

- It describes the use of k -medoids algorithm [20] to cluster a set of general-purpose benchmark applications (SPEC CPU2006 [21]) into k compatible groups. The applications are represented as an observation matrix consisting of rows of feature sets that are essential inputs to the k -medoids algorithm (Section 3.1).
- We process the output of the k -medoids algorithm to produce offline, low overhead, energy-efficient FVEs. The objective is to reach the write energy of optimal offline FVE while avoiding its disadvantages (Section 3.2). The proposed codec (coder/decoder) architecture uses read-only memories (ROMs) that are known to consume low power/energy in comparison to lookup tables (LUTs) and content-addressable memories (CAMs) (Section 3.3).

We divide the set of SPEC CPU2006 benchmarks into training and evaluation sets. The training set is used to derive the k offline FVE mappings, which are then used to encode the applications in the evaluation set. We evaluate the proposed solution for the MLC PCM prototype in [22] and the TLC RRAM prototype in [16]. Results (Section 4) indicate that average write energy in the MLC case is only 5% more than that of optimal offline FVE. The result is even better in the TLC case, where the average write energy of the proposed technique is 1% less than that of optimal offline FVE. In comparison to the write energies of MLC and TLC data comparison write (DCW) [23], the proposed solution achieves 39% and 35% energy savings, respectively, while the memory overhead in both cases does not exceed 3.5%. We report results for different values of k (the number of clusters), and for some values, our method results in average write energy that is only 4% more than that of the state-of-the-art MLC PCM encoding technique in [24]. However, [24] requires 50% NVM overhead, which is $16\times$ in comparison to the proposed solution. Whereas the 50% overhead of [24] can be reduced, we argue that such a reduction will result in exponentially larger sizes of LUTs in comparison to the codec ROMs required by our solution. For TLC RRAM, our method results in the same average write energy as incomplete data mapping (IDM) [10]. However, IDM has a memory overhead of 20%, i.e., $5.7\times$ the overhead of the solution proposed in this paper.

This research was supported by King Fahd University of Petroleum and Minerals and NSF Award CCF-1217738.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GLSVLSI '16, May 18–20, 2016, Boston, MA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4274-2/16/05...\$15.00

DOI: <http://dx.doi.org/10.1145/2902961.2902979>

Figure 1: In this plot, the x -axis is all the possible values (bytes) and the y -axis is the frequency of the corresponding value in logarithmic scale. Clearly, the frequency profile of benchmark 437.leslie3d exhibits good skew that supports the locality of frequent values. But aggregating the frequency profiles of 32 benchmarks results in a relatively flat plot, suggesting that more than 98% of the values are almost equally likely.

2. Background and Motivation

This section provides an illustrative overview of offline FVE [11] and highlights its potential advantages. Motivated by the fact that most computer applications have high data locality, FVE encodes frequent values such that they are assigned to the least energy codewords. It has been reported that these frequent values may account for 32% of all the values written to memory by a specific application [12]. Consider an example where an application has only four possible values: A, B, C, and D:

Value	A	B	C	D
Frequency	20	5	10	4
Energy (pJ)	36	307	547	20

In this table, the 1st row is the possible value, 2nd is the corresponding frequency of occurrence of that value, and 3rd is the corresponding write energy for each value. Therefore, the total energy consumed by this application is 7,805 pJ. The following value-codeword assignment:

Value	A	C	B	D
Codeword	D	A	B	C

reduces the total energy to 4,483 pJ, i.e., 43%. The assignment ranks the values in decreasing frequencies and codewords in increasing energies, and assigns values with high frequency to codewords with low energy in the order of the ranking. For example, A has the highest frequency and D has the lowest energy, we assign value A the codeword D. This optimal offline code assignment will only be good if used for this specific application (or a similar one), and therefore it is referred to as *optimal perfect knowledge* assignment, i.e., find-once for a given program [11]. Henceforth, we simply refer to this method as the perfect knowledge method.

In practice, if we have more than one application, we aggregate their frequency profiles and generate a combined code assignment resulting in lower energy. However, combining frequencies of incompatible applications may eliminate skew that is essential to achieve high energy reductions, as illustrated in Fig. 1. The plots show (1) the aggregate frequency plot of 32 distinct benchmark applications (28 SPEC CPU2006 [21] and four Splash-2 [25]) and (2) the frequency plot of the leslie3d benchmark. Even though most

offline frequency profiles of individual applications are skewed in favor of FVE, aggregating frequency profiles may eliminate skew and plateaus the graph for almost 98% of possible values. This is in comparison to the individual frequency plot of the benchmark leslie3d, in which the last elbow in the plot appears after 23% of possible values, implying high energy reduction.

Most applications have skewed frequency profiles in favor of perfect knowledge FVE. A primary motivation for this work is the careful aggregation of the frequency profiles of applications to derive multiple encodings, in which each encoding is still a result of a skewed aggregate frequency profile.

The authors in [14] evaluated perfect knowledge FVE on SLC PCM. While they reported that average write energy is greatly reduced, perfect knowledge FVE requires profiling support during the compilation process, implying a long compilation time for most applications. Another disadvantage of perfect knowledge FVE is that it requires operating system support, since for every application, we need to store the code assignment specific to that application in its executable. Perfect knowledge FVE is therefore expensive in practice.

On the other hand, online profiling occurs during application execution, e.g., [14]. Frequent values are identified on the fly and may also be replaced by other more frequent values. Although online profiling provides adaptive write energy reduction, it requires significant hardware and may affect overall system performance.

3. Contributions

The main contribution of this paper is the integration of an offline FVE solution for the purpose of MLC/TLC NVM write energy reduction approaching the reductions of perfect knowledge FVE without its disadvantages: the need for operating system and compiler support. Specifically, this work makes the following contributions:

- We use the k -medoids algorithm to cluster a set of applications into k compatible subsets. Precisely, our contribution is encoding the input applications as an observation matrix that consists of rows of feature sets, which is an essential input to the k -medoids algorithm (Section 3.1).
- We process the output of the k -medoids algorithm to derive a set of offline frequent value encodings. The objective is to achieve an average write energy that is as close as possible to the average write energy of perfect knowledge FVE (Section 3.2). We propose a codec architecture that utilizes ROMs that are known to consume low dynamic power in comparison to LUTs and CAMs (Section 3.3).

The proposed solution requires a slight modification to the traditional NVM memory word structure. This modified word structure is only internal to our codec and does not require any architectural changes outside the boundaries of the codec. As in Fig. 2a, an NVM word is composed of one or two tag cells, followed by a number of fixed length data slices. Each slice stores a frequent value in one of k encoded forms, and the tag cell(s) helps in the decoding process. The logical width of a data slice is referred to as frequent value length (FVL), and is equal to $\lceil \log_2 v_{\max} \rceil$, where $v_{\max}$ is the maximum possible frequent value. The FVL parameter and the number of possible frequent values to be encoded (FVN) characterize FVE. In this work, we fix FVL to 8 logical bits in the MLC case (Fig. 2b) and 9 logical bits in the TLC case (Fig. 2c). For FVN, however, the obvious choice would be 2^{FVL} , and this is used in this work. It is possible, however, that $\text{FVN} < 2^{\text{FVL}}$, and this can be combined with compression to lower the number of bit writes,

Figure 2: (a) A memory word in the proposed scheme consists of one or two tag cells and a number of data slices, which varies for MLC and TLC as shown in Table 1. Throughout this paper: (b) an MLC data slice consists of 4 physical MLC cells or 8 logical bits and (c) a TLC data slice consists of 3 physical TLC cells or 9 logical bits.

e.g., [14]. However, the cells in which the compressed values are written will potentially wear out faster. Hence, in this work, FVN is fixed to 2^{FVL} , i.e., 256 for MLC and 512 for TLC, to avoid biased wear of the memory cells.

3.1 Memory Trace Clustering

Finding data clusters is an unsupervised learning process in which data is divided into a given number of subgroups. Classical clustering algorithms include k -means [26] and k -medoids [20]. Clustering has played an important role in many areas, including artificial intelligence, pattern recognition, medical research, business intelligence, psychology, and political science [26, 27].

We utilize the k -medoids algorithm to cluster 32 memory traces of SPEC CPU2006 [21] and Splash-2 [25] benchmarks. These benchmarks represent a wide range of real user applications, and it is expected that most applications will have similar workloads. The k -medoids algorithm takes the following inputs: the number of clusters k , an observation matrix, and a similarity distance metric. For this work, the observation matrix consists of 32 rows (one row per benchmark), while the number of columns equals the feature set size, which is 256 for MLC and 512 for TLC.

Feature set extraction: The first step in generating the feature set of a given application is to extract its frequency information. Let $f(v)$ be a function returning the frequency of value v in the given application, where $0 \leq v < \text{FVN}$. Using this function, we form a set of pairs $\{(v, f(v)) : 0 \leq v < \text{FVN}\}$ and sort it in decreasing order with respect to $f(v)$ to generate an ordered list of these pairs. Each pair in the ordered list is composed of two entries: a value v and its frequency $f(v)$. Augmenting all the first entries of the pairs in the ordered list results in the feature set row vector $\vec{v}_d$. This feature set vector uniquely characterizes the given application. Two applications have the same vector *iff* they share similar, if not identical, frequency profiles. The rows of the observation matrix are constructed from these feature set vectors for all the 32 benchmarks. Note that a feature set vector is a permutation of all the values from 0 to FVN-1.

The similarity metric provides a means of measuring the distance between two applications, or more precisely, two feature sets or observations. It helps assign an observation to a cluster. In this work, since the rows of the observation matrix are permutations of the same set, a rank-based metric (Spearman rank correlation) was adopted. However, in most cases (see Section 4), other classic metrics, which are not rank correlated, perform equally well, i.e., the cosine and square Euclidean metrics.

As illustrated in Fig. 3, given k , an observation matrix, and a similarity metric, the k -medoids algorithm assigns a cluster ID (ranging from 0 to $k-1$) to every observation. Obviously, the size of the clusters may not be equal. Also, some of the clusters may have

Figure 3: Given the number of clusters k , the required similarity metric, and the observation matrix, the k -medoids algorithm divides the 32 observations (applications) into k clusters ($0 \leq C_i < k$ and $0 \leq i < 32$, where C_i is the cluster ID of observation i .)

a single observation. For the sake of separating training data from evaluation data, we split the output of the k -medoids algorithm into a training set and an evaluation set. This strategy is widely used in the literature and its purpose is to measure and compare the quality of clustering in both sets. A good clustering results in relatively close qualities, provided that the right number of clusters is chosen.

Since the size of the clusters may be different, and some clusters may contain an odd number of observations, we cannot always choose half the observations in each cluster to construct the training set. Instead, we create k lists, $L[i]$, where each list is initially identical to the corresponding cluster. We remove one observation at a time from the list containing the largest number of observations until the number of observations in the lists is halved. We assume that L is available as input to the code generation algorithm. The observations in the cluster lists represent the training set and the remaining observations represent the evaluation set.

3.2 Code Generation

Consider a function $e(v)$ that calculates the energy of frequent value v as the sum of the write energies of the cells composing v . For example, with respect to the PCM MLC prototype in [22], if $v = 0$ and FVL=8, then $e(0) = 36 \times 4 = 144$ pJ, since the value 0 is composed of 4 MLCs whose values are all 00. Another example is $e(0) = 2 \times 3 = 6$ pJ in case of RRAM TLC [16].

Next, we form a set of pairs $\{(v, e(v)) : 0 \leq v < \text{FVN}\}$ and sort it in increasing order by $e(v)$ to generate an ordered list of these pairs. Augmenting the first entries of all these ordered pairs, i.e., the v portion of the pairs, results in the vector $\vec{v}_e$. Similar to $\vec{v}_d$, the elements of $\vec{v}_e$ form a permutation of the integers in the range 0 to FVN-1. Note that $\vec{v}_e$ depends on the underlying MLC/TLC NVM technology and not on the application. It is a list of all the code-words ordered in increasing order by their energies. In the code generation algorithm, this vector is referenced item-wise, i.e., $\vec{v}_e[i]$ refers to the i^{th} element in the vector. Also, in the same algorithm, we assume that the function $F_p(o, v)$ returns the perfect knowledge frequency of value v in observation (application) o . Further, $L[i]$ is the i^{th} cluster list introduced earlier.

Algorithm 1 outlines the process of code generation. The final output of the algorithm is a two-dimensional mapping $M(i, v)$, where $0 \leq i < k$ and $0 \leq v < \text{FVN}$ are the encoding number and the value-to-be-encoded, respectively. Each cluster list generates one code as follows. First, the perfect knowledge frequency profiles of the applications in the cluster are aggregated, resulting in a set of pairs $\{(v, f(v)) : \forall v\}$. This set is sorted in decreasing order by frequency. The sorted list is stored in structure P , where $P[j]$ is the j^{th} pair with respect to the sort order, and $P[j].v$ refers to the v entry of the pair. The final step to build the i^{th} dictionary, $M(i, v)$, for all v . This is achieved by associating $P[j].v$ with the j^{th} codeword, $\vec{v}_e[j]$. These steps are repeated to build the codes for each of the other cluster lists.

Algorithm 1: Generating the k codes for the k clusters

Output : $M(i, v), 0 \leq i < k, 0 \leq v < \text{FVN}$, the i^{th} codeword of value v

for $i = 0$ **to** $k - 1$ **do**
 Initialize $f(v) = 0, \forall v$
 for each observation o **in** $L[i]$ **do**
 for $v = 0$ **to** $\text{FVN} - 1$ **do**
 $f(v) \leftarrow f(v) + F_p(o, v)$

Sort the pairs $\{(v, f(v)) : \forall v\}$ in descending order by $f(v)$, and let $P[i]$ be the i^{th} pair of this list. Further, let $P[i].v$ is the value part of the pair
for $j = 0$ **to** $\text{FVN} - 1$ **do**
 $M(i, P[j].v) \leftarrow v_e[j]$

3.3 Hardware Realization

The resulting codes from algorithm 1 are used to program k ROMs, one ROM per cluster. ROM i is programmed with the portion of the mapping tables corresponding to cluster i . Each slice of the incoming memory word is associated with its own encoding ROM, and the incoming memory word is encoded k times, one time per code. Then, the encoded version that results in minimum energy is selected for writing, taking into account the current content of the target address. Fig. 4 shows the encoding path for $k = 2$. Since we have two clusters, the code assignment algorithm will result in two mappings. Tag cell T_0 (T_1) takes the value 0 (1) to indicate that the word is encoded using the 0th (1st) mapping. The encoding overhead in bits due to using ROMs equals the size of one encoding ROM $\times$ the number of slices per word $\times$ the number of words in the memory line $\times k$. Note that all encoding ROMs have the same size which is $\text{FVN} \times \text{FVL}$ bits.

For decoding, we invert the mapping tables into ROMs as shown in the decoding architecture in Fig. 5. To avoid extra delay in the read path, the same ROM is duplicated in each slice across the whole memory line. One or more tag cells store the cluster number to decode the encoded slices inside the NVM array. Given a tag cell and an encoded slice, a decoding ROM outputs the decoded frequent value. Despite the duplication, the overhead is low in practice. The decoding overhead equals the size of one ROM $\times$ the number of slices per word $\times$ the number of words in the memory line, and ROM size is $k \times \text{FVN} \times \text{FVL}$ bits.

In Fig. 4 and 5, encoders and decoders are duplicated across all the slices in the memory line for simplicity. If this design is not affordable, an alternative lower overhead design is also possible using multiplexers and pipelining.

Figure 4: An NVM array consists of a number of rows, and each row contains words that are organized as in Fig. 2a. An unencoded new word from the CPU is received along with a target address. Then k encoded versions are generated and the word with the lowest energy (with respect to the current word at the target address) is selected for writing. For decoding purposes, each encoded version is associated with a tag value that is stored with the lowest energy version.

Figure 5: Decoding ROMs are replicated for each slice across the memory line. ROMs are addressed by slice value augmented with tag value.

4. Evaluation and Results

In this section, we present the simulation results of the proposed solution for the 4-state MLC PCM prototype in [22] and the 8-state TLC RRAM prototype in [16]. For brevity, we refer to these prototypes in this section as MLC and TLC, respectively.

Simulation setup: We evaluated the proposed solution using a trace-driven memory simulator. We used the Pin binary instrumentation tool [28] to extract the memory traces of 32 distinct applications: 28 SPEC CPU2006 [21] benchmarks and 4 Splash-2 [25] benchmarks. Table 1 lists the simulation configuration and the memory overhead due to using the tag cell(s). The number of logical bits in the memory line is 512 bits for MLC and 513 bits in TLC due to padding. In MLC, a 64-bit logical word is constructed from 32 physical cells. To avoid unnecessary padding in the TLC case, we chose a word size of 57 physical cells (171 logical bits). Note that this word size is local to the memory controller and does not imply a change to the word size of the CPU.

	MLC	TLC
Logical bits per memory line	512	513
Slice width or FVL (in logical bits)	8	9
Slices per word	8 / 16	19
Words per memory line	8 / 4	3
Tag cell(s) per word	1 / 2	1 / 2
NVM overhead	3.1%	1.8% / 3.5%

Table 1: MLC/TLC memory line organization and NVM overhead

Memory overhead: As indicated in Table 1, NVM overhead due to tag cell(s) is always 3.1% in MLC, since the ratio between the number of tag and data cells in the word is fixed. In TLC, the number of slices per word is constant, but the number of tag cells is one or two, and therefore, the overhead is 1.8% or 3.5%, respectively. The number of tag cells in the MLC and TLC cases is $\lceil \log_2 k \rceil$, where k is the number of clusters. Further, Table 2 reports the overhead for the codec ROMs for different values of k . Clearly, the codec overhead is negligible in comparison to the sizes of state-of-the-art NVMS.

k	2	4	8	16
MLC (KBytes)	8	16	64	128
TLC (KBytes)	43	86	171	342

Table 2: Assuming one codec is shared among all the words in a memory line, we show the ROM overhead for MLC/TLC NVMS for different values of k .

Hardware and latency overheads: We implemented Verilog modules for our codec circuitry. In our design, we avoid replicating the codec across all the words of the memory line, since this results in higher area. Instead, a single codec instance is shared (and

pipelined to avoid latency penalty) among all the memory words across the memory line. Using Design Compiler, we synthesize these modules on a 45nm technology node [29]. Since $k = 8$ provides best results for MLC, we chose to synthesize this case to examine the overheads. We implement the codec ROMs using case statements and the cosine metric.

The area of the synthesized codec is 0.23 mm^2 , which is negligible in comparison to the area of the state-of-the-art PCM memories, e.g., less than 0.4% the area of the 8Gb PCM in [30]. The latencies (energies) of the decoder and encoder paths are 1.93 ns (1.1 pJ/cell) and 3.91 ns (2.3 pJ/cell), which is also negligible in comparison to the PCM program-and-verify (P&V) write latencies (energies) [22]. Assuming P&V writes 34 cells at once, we can reduce the total memory line latency to a single word latency, since by the time the P&V modules complete writing one PCM word, the encoder will have encoded the entire line. But the decoding latency of the complete memory line equals the number of words per line times the word latency, i.e., 15.44 ns. If this is high, the decoder can be replicated to achieve a read latency that is as low as 1.93 ns for the memory line. Although this replication results in tripling the codec area (0.62 mm^2), it also reduces the read latency by $8\times$.

Energy reductions: Fig. 6 compares state-of-the-art methods to the proposed solution for MLC PCM. The bars represent the geometric mean (GM) of the write energy for each method across all the 32 applications. GMs are normalized to the GM energy consumed by DCW [23]. The methods from left to right are: DCW, the 50% overhead method [24], MFNW [18], cell remapping [9], and perfect knowledge FVE [11]. This is followed by aggregate FVE without clustering (1, no metric), and variations of our method, broken down by the number of clusters and distance metric, e.g., (2, cos) to indicate two clusters with the cosine metric.

Although the GM of the write energy of the proposed solution is measured across all the 32 applications, the training set, which is used to generate the k FVE mappings, is only composed of 16 applications. Fig. 7 compares the GM energy of the proposed solution across all applications, training set applications, and evaluation set applications. Energy reduction across the evaluation set for $k = 1$ is almost equivalent to DCW. As k increases, the reduction across the evaluation set improves. Although energy reduction across all the applications for $k = 8$ is better than $k = 16$, the energy reduction across the evaluation sets is marginally better ($\approx 1\%$) for $k = 16$ in comparison to $k = 8$.

Fig. 8 is the TLC equivalent of Fig. 6. The methods from left to right are: DCW, MFNW for TLC, incomplete data mapping (IDM) [10], and perfect knowledge FVE. NVM overheads of these methods are 0%, 1.8%, 20%, and 0%, respectively. Similar to the MLC case, as k increases, the energy reduction also improves. To compare energy reductions across the training and evaluation sets, we also provide a break down of this result in Fig. 9. Similar to MLC, when no clustering is performed ($k = 1$), energy reduction is only 5% better than DCW across the evaluation set. As k increases, the energy reduction across evaluation set improves. NVM overheads of the proposed solution are 1.8% for $k \leq 4$, and 3.5% for $k = 8$ and 16.

In all the previous charts, we divide the 32 applications into two halves: training set and evaluation set. To show the robustness of the clustering, we re-run the simulations using a training set size of

Figure 6: The x -axis lists state-of-the-art MLC PCM encoding techniques followed by our proposed method. The y -axis is the geometric mean (GM) write energy normalized to data comparison write (DCW) [23]. From left to right, the encodings are: DCW, 50% overhead encoding [24], MFNW [18], cell remapping [9], and perfect knowledge FVE [11]. This is followed by aggregate FVE without clustering (1, no metric), and variations of our method, broken down by the number of clusters and distance metric, e.g., (2, cos) to indicate two clusters with the cosine metric.

Figure 7: Each group of three bars represent the overall geometric mean (GM), training set GM, and evaluation set GM of MLC write energies. Similar to Fig. 6, write energies are normalized to DCW. As the number of clusters increases, the gap between write energies of the training set and evaluation set reduces.

8 applications and evaluation set size of 24 applications. Since we use only 8 benchmarks for training, a cluster size of 16 is no longer possible, and therefore, we only show the result for $k = 2, 4$, and 8. Fig. 10 shows the robustness chart in the MLC case. Clearly, the clustering is robust and the energy of the 25% training set size case (8 applications) is only 3.34% more on average in comparison to the 50% training set size (16 applications). The result is similar in the TLC case (Fig. 11), as the average energy is only 3.5% more.

In Fig. 6, the 50% overhead method produces the lowest MLC write energy, which is almost identical to perfect knowledge, but costs 50% NVM overhead, which is about $16\times$ more than our overhead. Moreover, the average write energy of the proposed solution is only 5% more than that of perfect knowledge when using 8 clusters. In comparison to MFNW and cell remapping, the proposed method consumes 29% and 24% lower energy, respectively.

For the TLC result in Fig. 8, IDM results in marginally better energy reduction over perfect knowledge ($\approx 1\%$). Note that our proposed solution is comparable to IDM when $k = 16$ for the square Euclidean and Spearman metrics. However, IDM has NVM overhead of $20\% \approx 5.7\times$ the proposed solution NVM overhead.

5. Conclusions

This paper proposed an energy efficient, low overhead offline FVE for MLC/TLC NVMs that approaches the performance of perfect knowledge FVE. The main idea is to cluster the perfect knowledge frequency profiles of a broad set of general-purpose applications to generate combined FVE mappings for each cluster. Results show that the average write energy for MLC PCM of the proposed

Figure 8: **Results, organized similar to Fig. 6, for state-of-the-art TLC encoding solutions (left to right): MFNW for TLC, incomplete data mapping (IDM) [10], and perfect knowledge FVE. Again, write energies are normalized to DCW.**

Figure 9: **Results for TLC RRAM, organized similar to Fig. 7. The gap between write energies of the training and evaluation sets consistently decreases as k increases, except for the case from $k = 8$ to $k = 16$ for the square Euclidean and Spearman metrics (12% in both cases).**

method is only 5% more than the average write energy of perfect knowledge FVE. For TLC RRAM, the average write energy of the proposed solution is better by 1%. Furthermore, the low NVM and codec overheads makes the proposed method easy to implement and integrate into modern memory controllers.

6. References

- [1] "International Technology Roadmap for Semiconductors," 2011.
- [2] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Intl. Symposium on Computer Architecture*, 2009.
- [3] I. Baek *et al.*, "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proc. Intl. Electron Devices Meeting*, 2004.
- [4] H.-S. Wong *et al.*, "Metal-oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, 2012.
- [5] J. Yue and Y. Zhu, "Accelerating write by exploiting PCM asymmetries," in *Proc. Intl. Symposium on High Performance Computer Architecture*, 2013.
- [6] J. Li and K. Mohanram, "Write-once-memory-code phase change memory," in *Proc. Design, Automation and Test in Europe Conference*, 2014.
- [7] D. Dgien *et al.*, "Compression architecture for bit-write reduction in non-volatile memory technologies," in *Proc. Intl. Symposium Nanoscale Architectures*, 2014.
- [8] A. Mirhoseini *et al.*, "Coding-based energy minimization for phase change memory," in *Proc. Design Automation Conference*, 2012.
- [9] J. Wang *et al.*, "Energy-efficient multi-level cell phase-change memory system with data encoding," in *Proc. Intl. Conference on Computer Design*, 2011.
- [10] D. Niu *et al.*, "Low power multi-level-cell resistive memory design with incomplete data mapping," in *Proc. Intl. Conference on Computer Design*, 2013.
- [11] J. Yang and R. Gupta, "Frequent value locality and its applications," *ACM Trans. Embed. Comput. Syst.*, vol. 1, no. 1, 2002.
- [12] J. Yang *et al.*, "Frequent value encoding for low power data buses," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 3, 2004.
- [13] D. Suresh *et al.*, "Energy-efficient encoding techniques for off-chip data buses," *ACM Trans. on Embedded Computing Systems*, vol. 8, no. 2, 2009.
- [14] G. Sun *et al.*, "A frequent-value based PRAM memory architecture," in *Proc. Asia and South Pacific Design Automation Conference*, 2011.

Figure 10: **Geometric mean (GM) of MLC write energies, normalized to DCW. The two cases are training on 50% of the applications and evaluation on 50% versus training on 25% and evaluation on 75%. By training on 25% of the applications, write energy increases by only 3.34% on average.**

Figure 11: **TLC case similar to Fig. 10. By training on 25% of the applications, write energy increases by only 3.5% on average in comparison to training on 50% of the applications.**

- [15] D.-H. Kang *et al.*, "Two-bit cell operation in diode-switch phase change memory cells with 90nm technology," in *Proc. Symposium on VLSI Technology*, 2008.
- [16] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. Design Automation Conference*, 2013.
- [17] W. Wen *et al.*, "State-restrict MLC STT-RAM designs for high-reliable high-performance memory system," in *Proc. Design Automation Conference*, 2014.
- [18] A. Alsuwaiyan and K. Mohanram, "MFNW: A flip-n-write architecture for multi-level cell non-volatile memories," in *Proc. Intl. Symposium on Nanoscale Architectures*, 2015.
- [19] L. Jiang *et al.*, "ER: Elastic RESET for low power and long endurance MLC based phase change memory," in *Proc. Intl. Symposium on Low Power Electronics and Design*, 2012.
- [20] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for K-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, 2009.
- [21] "SPEC CPU2006," 2006.
- [22] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, 2009.
- [23] B.-D. Yang *et al.*, "A low power phase-change random access memory using a data-comparison write scheme," in *Proc. Intl. Symposium on Circuits and Systems*, 2007.
- [24] A. Mirhoseini *et al.*, "Phase change memory write cost minimization by data encoding," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 1, 2015.
- [25] J. M. Arnold *et al.*, "Splash 2," in *Proc. Parallel algorithms and architectures*, 1992.
- [26] J. Wu, *Cluster Analysis and K-means Clustering: An Introduction*, 2012.
- [27] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: An introduction to cluster analysis*. New York: Wiley, 1990.
- [28] C.-K. Luk *et al.*, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. Conference on Programming Language Design and Implementation*, 2005.
- [29] J. E. Stine *et al.*, "Freehdl: An open-source variation-aware design kit," in *Microelectronic Systems Education*, 2007.
- [30] Y. Choi *et al.*, "A 20nm 1.8v 8gb PRAM with 40mb/s program bandwidth," in *Proc. Intl. Solid-State Circuits Conference*, 2012.