Coding for Non-Volatile Memory Technologies: Theoretical Advances and Practical Considerations

Lara Dolecek, *Senior Member, IEEE,* and Yuval Cassuto, *Senior Member, IEEE*

***Abstract*—Every bit of information in a storage or memory device is bound by a multitude of performance specifications, and is subject to a variety of reliability impediments. At the other end, the physical processes tamed to remember our bits offer a constant source of risk to their reliability. These include a variety of noise sources, access restrictions, inter-cell interferences, cell variabilities, and many more issues. Tying together this vector of performance figures with that vector of reliability issues is a rich matrix of emerging coding tools and techniques. Channel coding schemes ensure target reliability and performance and have been at the core of memory systems since their nascent age.**

In this survey, we first overview the fundamentals of channel coding and summarize well-known codes that have been used in NVMs. Next, we demonstrate why the conventional coding approaches ubiquitously based on symmetric channel models and optimization for the Hamming metric fail to address the needs of modern memories. We then discuss several recently proposed innovative coding schemes. Behind each coding scheme lies an interesting theoretical framework, building on deep ideas from mathematics and the information sciences. We also survey some of the most fascinating bridges between deep theory and storage performance. While the focus of this survey is primarily on the pervasive multi-level NAND Flash, we envision that other benefiting memory technologies will include phase change memory, resistive memories, and others.

***Index Terms*—Flash memories, ECC, algebraic codes, BCH codes, graph codes, LDPC codes, re-write codes, WOM codes.**

I. Introduction

NOn-volatile memories (NVMs) are a class of computer memories that maintain the stored data even after being disconnected from a power supply. NVMs have many desirable properties that have made them frontrunners to replace conventional hard-disk drives: they are faster, less power hungry, more flexible in form factor, amenable to random access, and not prone to heat-induced damages. As a result, NVMs are now being actively considered and designed for use in a diverse set of applications, including personal electronics and smart devices, autonomous vehicles, enterprise storage, and data-intensive high performance computing. This surge in NVM development has not been without challenges; both established and emerging NVMs come with a unique set of operational issues that must be overcome before these technologies can be broadly deployed at low cost.

Memories intrinsically suffer from various impairments, which steeply get worse in the high-density, fast access regime.

L. Dolecek is with the Department of Electrical and Computer Engineering, University of California, Los Angeles (UCLA), Los Angeles, CA 90095, USA, e-mail: dolecek@ee.ucla.edu.

Y. Cassuto is with the Viterbi Department of Electrical Engineering, Technion – Israel Institute of Technology, Technion City, Haifa 3200003, Israel, email: ycassuto@ee.technion.ac.il.

Manuscript received November, 2016; revised xxx.

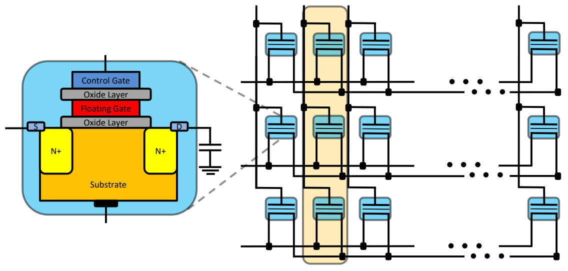


Fig. 1. NAND Flash array organization with a zoomed-in single cell. Read operation is performed by activating one word line (highlighted column).

It is indeed this stringent regime in which modern data-intensive applications operate – NVMs will have to be robust, fast, and affordable if they are to deliver on the promise of new information technologies.

In this article, we demonstrate the essential role channel coding techniques play in modern NVMs. We first summarize the key operational characteristics of NVMs, using Flash as the paragon. As we then show, understanding and appreciation of physical properties is mandatory for the proper development of new, mathematically deep yet practical coding solutions for future memories, in which physical impairments are bound to only get worse. By explicitly showcasing exciting recent advancements in coding theory specifically geared for the NVM applications, we highlight the enormous potential that tools from coding theory and related mathematical disciplines can have in the development of future, robust NVMs. These results build upon rich mathematical fields of combinatorics, abstract algebra, graph theory, and others, to offer rigorous yet elegant repertoire of both algebraic and graph codes.

In the next section, we describe the basic operating principles of Flash memories and summarize the main sources of physical impairments. The following Section III reviews the fundamentals of channel coding necessary to explain the concepts presented in subsequent sections. The next two sections are devoted to coding techniques for reliability. Section IV is devoted to algebraic coding techniques, both classical and recent. Section V considers graph codes, including conventional methods and recent Flash-tailored advances. Moving beyond reliability, in Section VI we discuss re-write codes for improved access. Section VII delivers conclusions.

II. Fundamentals of operations in Flash memories

An atomic unit of a Flash memory is one memory cell. A memory cell corresponds to a transistor that has a control gate and a floating gate, separated by insulating layers. The value of data stored in a memory cell corresponds to the amount of charge on the floating gate. Flash technologies are classified as NAND Flash and NOR Flash, corresponding to the logical NAND-like and NOR-like arrangement of the device, respectively. In NOR Flash, cells can be accessed individually. In contrast, in NAND Flash, cells are accessed at the much coarser granularity of pages. However, NAND Flash has a substantially lower cost than NOR Flash and is thus more pervasive; we will henceforth primarily focus on NAND Flash. See Figure 1 for a NAND cell illustration. In planar NAND Flash architectures, memory cells are organized into two dimensional arrays. Cells are organized into pages, which are further combined into NAND-blocks[1]. Thousands of NAND-blocks amount to one NAND device. For example, a 2Gb Flash device may consist of 2048 NAND-blocks with 64 pages per NAND-block, and 2112 bytes per page; other combinations of pages/NAND-block sizes are also possible and they yield different overall device capacities. More recently, 3D (vertical) NAND Flash has been developed, which has a more complex architecture due to the 3D structure.

Write and erase operations on the cells are performed by applying a sufficiently high voltage to the control gate to alter the amount of charge on the floating gate, which in turn sets the cell's *threshold voltage*. Depending on the direction in which the electrons flow, the cell is programmed (electrons flow towards the floating gate) or erased (electrons flow away from the floating gate). This process is known as Fowler-Nordheim tunneling [1]. The process of reading amounts to determining the amount of charge stored on the floating gate. During a read, an input voltage is applied to the control gate and the drain current is measured. If the drain current is below/above a certain threshold, as measured by a sense-amp comparator, we conclude that the input voltage is below/above the cell threshold voltage representing the information stored. Therefore, a *single threshold* scheme tells us whether the stored charge is below or above a certain level but does not tell us the exact amount of stored charge. The input voltage is applied to all cells in the page together, thus parallelizing the read operation. Placement of the threshold voltage need not be static; using tools from information and communications theory, recent work has demonstrated clear benefits of dynamically adjusted threshold voltages for improved lifetime [2].

Flash devices are commonly categorized by the number of bits memory cells can store. Single-level cell (SLC) devices store a single bit per cell; the SLC nomenclature comes from the fact that a single threshold is needed to distinguish the two ranges, corresponding to bit value '0' and bit value '1'. Multiple level cell (MLC) devices store multiple bits per cell, and multiple thresholds are needed to distinguish different levels. In the industry jargon, the MLC initials commonly refer specifically to two bits per cell. As somewhat of a misnomer, triple level cell (TLC) refers to a multilevel Flash device storing three bits per cell – here seven thresholds, not three, are needed to distinguish among eight ranges (one for each

[1]The common terminology refers to NAND-blocks simply as "blocks", but we reserve this term to denote a code block, the basic unit of coding.

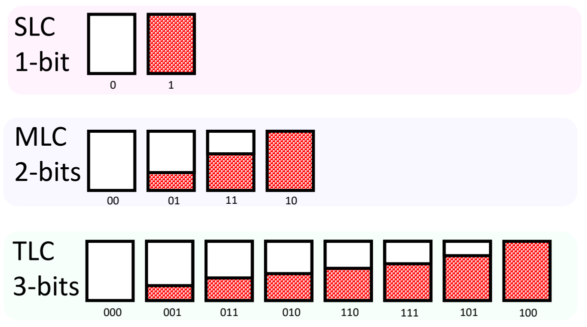


Fig. 2. SLC, MLC, and TLC cells, with digital information associated with the charge amount. Gray coding is used to label different levels thus minimizing the possibility of bit errors.

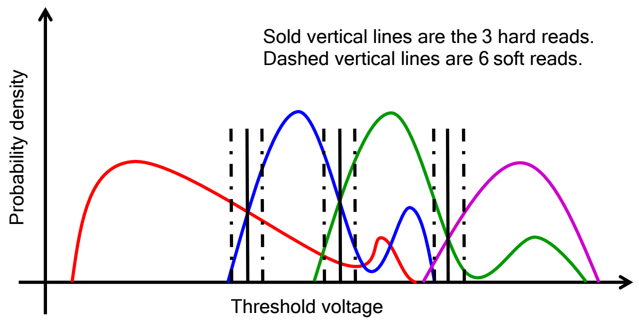


Fig. 3. MLC level distribution, with different reads.

combination of a pair of adjacent binary triplets). The lowest level is the fully erased state and the highest level is the fully programmed state.

See Figure 2 for an illustration of SLC, MLC, and TLC devices. Note that the differentiation amongst adjacent digital values shrinks as the density increases making it harder to distinguish amongst different values. The figure is also an idealized representation of memory cells, since the cells do not always behave in exactly the same way. The value of the threshold voltage reached when writing a certain information value varies across cells, due to variations in cell behavior. As a result, we observe a distribution associated with different cell levels. Modeling and parameterization of this difficult distribution requires a careful study that has been a subject of considerable recent research in industry and academia alike, including notable works [3], [4], [5], [6], [7], [8], among others. See Figure 3 for the illustration of the threshold voltage distribution in the MLC case.

A critical property of NAND Flash is that the write operation by way of adding charge is executed at the page level, but the erase operation is done at the much coarser NAND-block level. Thus, in principle, if a value of a single cell is to be decremented, the entire NAND-block of cells this cell belongs to would also be erased and re-written. Unfortunately, frequent erases wear out the device to the point that it can no longer be used with confidence. The usable lifetime is typically expressed in terms of the number of program and erase (P/E) operations that can be executed before the device is no longer considered reliable. It is well understood that the denser the device is, the fewer P/E cycles it can sustain before it is deemed unusable. For SLC devices, the expected lifetime is around 10^5 P/E cycles. For MLC devices storing two bits per cell, the lifetime drops to 10^4 P/E cycles, and with three bits per cell TLC memory, it drops even further to a mere 10^3 P/E cycles or even fewer [9], translating to only months of usable lifetime for frequently written devices. The P/E lifetime issue is further exacerbated by the reliability requirements: the raw bit error rate (BER) can be quite high, even as high as $10^{-2} - 10^{-1}$ [9], while the product specification requires the device to operate at the undetected BER (UBER) level of highly demanding 10^{-16} and even lower [10]. *Wear leveling* is employed in practice in order to carefully balance the number of P/E cycles across the NAND-blocks of the device to make the degradation more uniform. As the densities increase, managing wearout and impairments becomes an increasingly more daunting mission; fortunately, when equipped with proper ECC schemes, the task becomes much more manageable.

There are several intertwined causes of errors in read and write operations, which we summarize as follows.

Writing into cells is done by way of so-called incremental step pulse programming (ISPP) across a page of cells: a small amount of charge is repeatedly being added to each cell, cell values are read back to test if the target value is reached, and the next charge increment is added to those cells that still do not have the target values. ISPP is beneficial as it minimizes the detrimental effects of overshooting (inadvertent addition of too much charge) that would result in a costly NAND-block erase. However, very small ISPP steps also significantly slow down the write process, especially in the multi-level memories. An additional issue with this write process is that adding charge into one cell may unintentionally raise the charge on the adjacent cell, with which the first cell shares a line. The worst scenario of inter-cell interference with the ISPP is when a cell with low target level has its two neighbors programmed to high levels [11].

Even if a cell is correctly programmed, issues may arise during the read step. As the time passes since the initial programming, the electrons slowly leak out of the floating gate, and when the cell is eventually read, the observed value is lower than what had originally been written. Additionally, as the device ages, the total amount of charge that can be stored in cell gates gets reduced because of worsening defects.

An intriguing and design-critical observation is that these write and read characteristics are highly *asymmetric*. For instance, charge leakage only causes errors in the downward direction, and overshooting and inter-cell disturbs only cause errors in the upward direction. Additionally, inter-cell coupling affects low levels more profoundly than high levels. Push for smaller geometries and increased densities has progressively worsened these impairments. Fortunately, new innovations in coding hold promise to help reverse this negative trend. Before we describe these advanced methods, we first review the basic terminology of channel coding.

III. Error-Correction Codes (ECC): Introduction

In this section we review some coding terminology necessary to understand the approaches described later in the paper. Error-correction codes (ECC) are widely used in memories. ECC methods add a certain amount of redundancy to the input data prior to storing it on the medium in order to combat adverse effects of noise and other device impairments. These methods can be broadly divided into algebraic and graph methods, depending on the key mathematical principles that underpin the given code construction.

The collection of possible codewords (stored words that bear redundancy) is called a *code*. Practical coding methods used in memories and storage devices store data in equally sized blocks: input message of length k symbols is mapped to a codeword of length n symbols, with $n > k$. A code can be binary or non-binary, depending on whether one symbol corresponds to just one bit or to multiple bits. Non-binary codes are defined over finite fields that have cardinality that is a power of a prime. The representation with the number of different symbols, say q, that is a power of 2 is particularly useful for multi-level Flash devices that store multiple bits per cell (and thus have the number of levels that are a power of 2).

The *rate* of a code is k/n. In NVMs, as is the case with other storage devices, the rate should be close to 1 in order to minimize the storage overhead associated with redundancy. The codes are typically linear, so that the code forms a k-dimensional linear subspace in this n-dimensional space, where n is the codeword length. Linearity allows for a compact representation of the code. A linear code can be represented both via a *generator matrix* and via a *parity-check matrix*. The former has rows that span the range space of the code and the latter has rows that span the null space of the code. Alternatively, the rows of the parity check matrix can be viewed as parity-check equations that each codeword in the code must satisfy. As we describe in more details later, the parity-check matrix viewpoint is especially well suited for graph codes. In the case of algebraic codes, depending on the details of the construction, interpretation in terms of one of the two matrices can be more convenient.

The product of a codeword with the parity-check matrix always produces the all-zeros vector. The product of any other word that is not a codeword with the parity check matrix produces a vector which is strictly non-zero. We refer to the output of the product of a word of length n and the parity check matrix as the *syndrome* of the word. In principle, exponentially many words have the same syndrome. Syndromes are often used in the decoding of algebraic codes.

The *minimum Hamming distance* d_{min} of a code is the smallest number of positions in which two distinct codewords differ. In the canonical setting, the parameter $t = \lfloor (d_{min} - 1)/2 \rfloor$ is the measure of how many errors can be corrected. Classical coding techniques are typically characterized in terms of how many errors t can be corrected [12] – for given code parameters k and n, one typically seeks to maximize the minimum distance of a code. It is an oft-overlooked fact that these well studied techniques implicitly assume that the errors are equally likely and symmetric. As we described, modern NVMs possess a large amount of asymmetry; shoe-horning an existing channel code into the NVM model is bound to be grossly inefficient. We discuss many of the recent coding proposals that explicitly depart from this ineffective approach, but first we summarize early coding solutions for older NVMs wherein conventional coding tools were deemed adequate.

IV. Error Correction with Algebraic Codes

A. Classical codes: from Hamming to BCH and Reed-Solomon

Early NVM technologies only required mild error correction capabilities for which Hamming codes were sufficient [13]. Hamming codes are one of the simplest coding methods, characterized by a parity-check matrix whose columns are all the non-zero binary tuples of a particular length. Hamming codes are single-error correcting codes since any error pattern with exactly one non-zero symbol can be corrected. However, as the devices scaled down and area density increased, reliability constraints became more stringent, and the need for more sophisticated coding methods soon followed.

BCH (Bose-Chaudhuri-Hocquenghem) codes are a well-known class of linear algebraic codes that emerged as the coding solution of choice for early Flash memories [14]. BCH codes – and Reed-Solomon codes as their special case – were already popular in commercial data storage technologies (e.g., hard-disk drives) and were well understood by memory designers.

BCH codes can be viewed as a generalization of Hamming codes. Like Hamming codes, they are linear block codes with a well-defined structure. Unlike Hamming codes, they can be constructed to correct multiple errors. This is done by simply constructing a parity-check matrix of a code as an array of elements from an appropriately chosen finite field, using well established rules from conventional algebraic coding theory [12]. BCH codes have the guaranteed error-correction property that memory designers favor: one can explicitly design a code capable of correcting all patterns with up to a prescribed number of errors.

BCH code is also an instance of a cyclic code, wherein a cyclic shift of a codeword produces another codeword. This viewpoint is helpful for encoding, as each codeword is then represented as a product of the message polynomial (a polynomial with message symbols as coefficients) with the generator polynomial defining the code. The coefficients of the resulting polynomial are then symbols of the produced codeword.

Decoding is a more difficult task as it amounts to several non-trivial steps. First, one computes the syndrome associated with the given retrieved word. Based on this syndrome, one then seeks to find the most likely error pattern that has this syndrome. Exhaustively searching for the most likely error pattern with a given syndrome is completely impractical. What is done instead for BCH codes is the construction of an auxiliary polynomial, called *error-locator polynomial*, whose coefficients are linearly related to the coefficients of the symbol polynomial. The roots of this error-locator polynomial are then precisely the locations of the erroneous symbols. Construction of the polynomial is routinely done via the Berlekamp-Massey algortihm, and computation of the roots of the polynomial is done using Chien search [12].

Practical decoder implementations must always strike a careful balance between additional coding gain enabled by more powerful codes and the increased resource consumption caused by additional decoding circuitry. In the common regime of using high-rate codes, the complexity is in general manageable because the number of corrected errors is not too large. Several recent works have specifically addressed this question in the context of BCH decoders for Flash memories, relying on the techniques of partial parallelization and pipelining, e.g., [15]. In general, the slowest step in BCH decoding is the Chien search, which is typically done in parallel to improve the decoding throughput. However, a parallelized solution also incurs additional hardware complexity and energy consumption – recent architectural approaches geared towards NVM applications have been developed to reduce the area consumption of the parallel Chien search by removing redundant operations [16], further combined with more informative scheduling [17], and by formulating Chien search as a matrix multiplication for faster search [18], [19].

We also remark that an additional benefit of BCH codes in the context of NVMs is that they intrinsically have a rate-compatibility feature: a parity-check matrix of a BCH code correcting t_1 errors is a sub-matrix of a BCH code correcting t_2 errors, for $t_1 < t_2$, In other words, for the same code length n, a t_2-error correcting BCH code $\mathcal{C}_2$ is a subcode of a t_1-error correcting BCH code $\mathcal{C}_1$. Alternatively, from the encoder's perspective, since BCH codes are cyclic codes, $\mathcal{C}_2$ can be constructed from $\mathcal{C}_1$ by adding monomial terms to the generator polynomial of $\mathcal{C}_1$. (For theoretical details, see [12].) As discussed in Section II, NVMs are highly susceptible to wear out. The noise worsens over time, requiring more redundancy in the code to deal with higher rate of errors. One way of addressing this issue is by using rate-compatible codes, with a high-rate code deployed in the early part of the lifetime, and a lower-rate code in the latter part. Seamlessly switching to progressively more powerful codes is relatively easy with the BCH set-up because it simply amounts to introducing additional parity-check symbols over what had already been stored with respect to a codeword of the initial code.

BCH codes are a prime exemplar of what the conventional coding theory offers: powerful error-correction schemes intrinsically designed to deal with *symmetric* errors wherein the ability to correct an error pattern only depends on the number of symbol errors in it, and not on how the symbols change by the errors. However, as we discussed in the previous section, error patterns arising in modern NVMs are far from symmetric! This observation has motivated intense recent research activity that explicitly departs from the conventional code design for symmetric errors. We now discuss how several recently proposed coding approaches have addressed the operational properties of NVMs, and have also led to a new chapter of fundamental advances in coding theory.

We choose to survey two algebraic coding schemes that are the most convenient to deploy, because they can use existing coding modules (e.g., from BCH codes) as their main building block complexity-wise.

B. Algebraic codes for NVM error models

A central characteristic of multi-level NVM channels is that the incident errors are structured rather than symmetric. The structure of the errors stems from the electric and algorithmic features of the write and read processes. For example, the representation of data as q discrete charge levels makes an error more likely between adjacent levels than between far-apart levels. Such error structure is not addressed by classical codes like BCH and Reed-Solomon, which are designed for symmetric errors. It is still possible to use symmetric error correcting codes for non-symmetric errors, but this use is highly sub-optimal because the codes need to cover error events strictly worse than actually needed at likely operation. For example, a common technique in practice is to implement a *gray mapping* between q-ary charge levels and tuples of $\log_2 q$ bits (cf. Figure 2). With this mapping, a q-ary error between nearby levels translates to a binary error in a small number of bits. But even with this desired property, a binary code correcting the resulting bit errors is required to correct more errors than really needed, thus unjustly adding to the redundancy cost. For example, consider the simple binary reflected gray code on 3 bits, corresponding to $q = 8$ levels. In this mapping we map the levels $(0, 1, 2, 3, 4, 5, 6, 7)$ to the bit tuples $(000, 001, 011, 010, 110, 111, 110, 100)$. To see that this is a sub-optimal mapping, we observe that the transitions $0 \leftrightarrow 3$ are single-bit errors exactly like the transitions $0 \leftrightarrow 1$, even though the former are much less likely in a realistic memory channel. In the rest of this section we describe two coding schemes that better capture the structure of multi-level NVM channels. In these promising alternatives we still use known symmetric error-correcting algebraic codes, but in a clever way to maximize the coverage of the error patterns of interest. The schemes rely on the celebrated concept of *code concatenation* [20] developed for communication applications, while specializing and refining to best match deployment in Flash and other NVMs. Code concatenation is a powerful technique that combines two codes, wherein codewords of the inner code are symbols in the alphabet over which the outer code is defined. The two schemes in discussion are distinct and complementary: the first one is especially tailored to NVMs where coding is done directly over the non-binary cells; the second one better fits gray-mapped memories composing a cell level as multiple bits.

1) Codes for errors with magnitude limit: Suppose that our memory has $q = 8$ levels, and that a common error mechanism changes a desired level x to level $x - 1$. This error type is called *asymmetric errors with magnitude* 1. One possible source for such errors is *retention errors* [21], whereby charges gradually escape the cells when they are not re-written or refreshed for a long period of time. A complete analog of this error model takes effect when level x changes to $x + 1$, which can happen for example when the programmed level (irreversibly) overshoots above the level requested in a cell write, or due to disturbs from other cells' writes. We note that it is not required that *all* errors will be asymmetric errors with magnitude 1; treating this error model is beneficial even when other secondary error sources are active alongside of it. Moreover, codes similar to what we next describe can also be constructed for two-directional errors $x \pm 1$, and with error magnitudes greater than 1.

To start the discussion on coding for asymmetric errors of magnitude 1, it will be instructive to consider the extreme case where *all cells* in the word may experience an $x + 1$ error. It is clear that the best solution to this case is to "give up" one of the three bits in each cell, and use only half of the $q = 8$ levels, for example all the even levels $\{0, 2, 4, 6\}$, [22]. When the errors are less intensive we do not want to lose an entire bit per cell, and instead do the following [23]. We write a page of n cells as $3n$ bits with the restriction that the n bits of the lower significance belong to the binary code BCH1 that corrects t bit errors. This procedure is depicted in Figure 4a, where the shaded area represents the parity bits of BCH1. Note that the other two rows in the n cells of Figure 4a are stored uncoded. At read time, we obtain the bits of the coded n-bit row, and use the decoder of BCH1 to *locate* the errors, but not to correct them as bit errors. Instead, in each error location we reverse the error by subtracting 1 from the read 8-ary level. It is clear that this scheme can correct up to t asymmetric errors of magnitude 1. It borrows all the good properties of BCH codes for symmetric errors, while exhibiting several optimality features for the target error model [23].

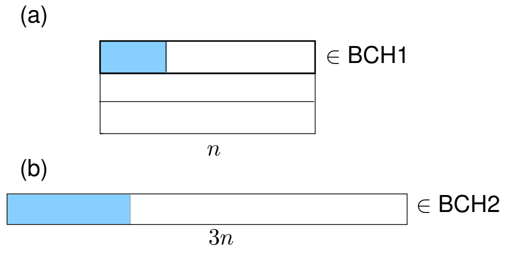


Fig. 4. Encoding strategies for correcting asymmetric magnitude-1 errors in a memory with $q = 8$ levels. (a) a code designed for asymmetric magnitude-1 errors, and (b) a binary gray-mapped code. Both codes are based on BCH codes, but (a) requires fewer parity bits than (b).

In contrast, an alternative scheme for the same error model maps to each 8-ary cell level 3 bits using a gray code. Applying a t-error binary code BCH2 to the $3n$ bits, shown in Figure 4b, also guarantees correction of t asymmetric errors of magnitude 1. However, the number of parity bits required for this alternative scheme is larger by roughly $t \log_2(3) \approx 1.58t$, which amounts to significantly increased redundancy when t is moderate to high. Beyond this specific example, the presented scheme can use any code for symmetric errors, not necessarily a BCH code. In addition, it can be extended to any q, any error magnitude l, and other error models with structure [23], [24], [25], [26].

As the memory technology scales in density, we expect the low-magnitude errors to become more frequent and dominant. In the regime of moderate to high rates of low-magnitude errors, the scheme detailed previously in the section may not be the most efficient, because symmetric-error codes for large

t are expensive to implement. We show that in this case the best approach works quite differently than previous coding schemes for such errors. Given a block of n cells with q levels, where n may be smaller than the memory page size, we encode the data such that the block *does not contain cells with consecutive levels* in $\{0, \ldots, q-1\}$. For example, if the block has a cell with level 3, then it cannot have any cell with levels 2 or 4. Another block may have a cell with level 4, but then it cannot have cells with 3 or 5. The key in this encoding is that its knowledge by the decoder can help to efficiently correct a large number of asymmetric errors with magnitude 1. Not less importantly, for finite block lengths n this encoding is much less redundant than the encoding that uses only half of the levels $\{0, 2, 4, \ldots\}$. A coding scheme based on this idea was suggested with the name *NCC* (non-consecutive constraint) [27], and was shown to have the best error correction given the expended redundancy. This can be seen in Figure 5 showing the output symbol-error rate (output SER) as a function of the input symbol-error rate (input SER). It is seen that other coding alternatives with the same code rate have inferior performance. The plot with + markers shows the performance of the coding scheme depicted in Figure 4a using a constituent BCH code, and the one with $\diamond$ markers shows it for the even/odd code that restricts the n levels to be all even or all odd. Conveniently, the NCC can control the trade-

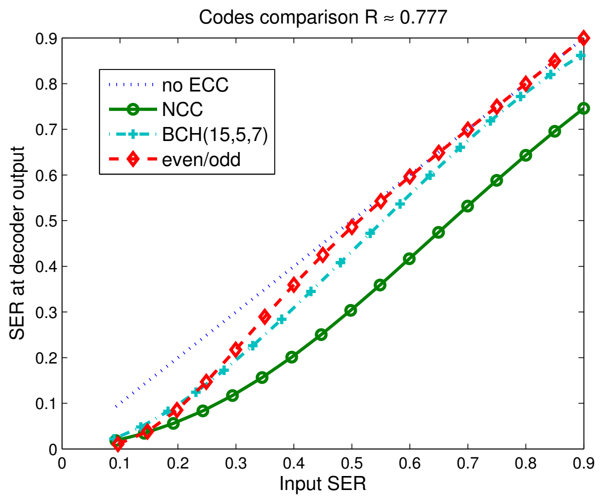


Fig. 5. Correcting many asymmetric errors of magnitude 1: comparison between three coding schemes with equal code rate, set to 0.77.

off of rate vs. correction capability by merely changing the codeword length n: a small n gives high rate and weak error correction, and as n grows the error correction improves and the rate decreases. Overall the significant reduction in SER from input to output by the NCC code allows cleaning up the remaining symbol errors (together with errors from other types) by a reasonable-strength outer code, e.g. a binary LDPC code.

2) Tensor-product codes (TPC) for heterogeneous errors: Another promising example of carefully exploiting BCH-like codes for errors of special characteristics reported in the literature [28], [29] is given by the tensor-product codes [30], which have the intrinsic capability of incorporating fine-grained knowledge of the error patterns.

To motivate this discussion, let us consider the following example for $q = 8$ levels representing 3 bits (TLC Flash): Data is stored in triplets (each triplet corresponds to one triple-level cell.) For 5 TLC cells, let us say that $(100, 110, 000, 000, 010)$ is stored. We read back the block $(100, 100, 000, 011, 011)$. The binary difference (XOR) is $(000, 010, 000, 011, 001)$.

We could consider each triplet a symbol, observe that there are 3 symbol errors, and thus seek a code correcting at least 3 symbol errors. However, this is not a sufficiently refined definition: note that the majority of the erroneous triplets only contain 1 bit in error (as indeed would be the case in Flash). This observation is not considered by symmetric error-correcting codes, such as non-binary BCH codes. A more efficient code must exploit this notion, correcting a certain number of erroneous triplets with few bits in error and a much smaller number of erroneous triplets with many bits in error. TPC codes offer precisely this added efficiency.

Mathematically speaking, the baseline TPCs (from [30]) are expressed as a particular type of concatenation: their parity-check matrix H is itself a tensor product (hence the name) of a parity-check matrix A of a non-binary code $\mathcal{C}_A$ with a parity-check matrix B of a binary code $\mathcal{C}_B$, denoted $H = [A \otimes B]$. In the context of multilevel Flash memories, the code length of $\mathcal{C}_A$ corresponds to the number of memory cells and the code length of $\mathcal{C}_B$ corresponds to the number of bits per each cell. This construction then allows for controlling the error-correction capability simultaneously over cells and over bits per each cell, that is, we aim to correct a certain number of erroneous cells, and for each erroneous cell, we correct a certain (small) number of erroneous bits.

Flash-motivated extension of this construction was developed in [31], where the parity-check matrix H was built out of 4 constituent binary/non-binary parity-check matrices, in order to also accommodate rarer, larger weight errors, which the original construction is too rigid to handle. The resultant parity check matrix is then $H = \begin{bmatrix} A \otimes B \\ C \otimes D \end{bmatrix}$. Binary matrices B and D control the number of correctable erroneous bits per erroneous cell, and non-binary matrices A and C (over appropriately defined finite fields) control the number of correctable symbol errors that have a prescribed number of bit errors; for example we can construct a code that corrects t_1 symbol errors each flipping at most ℓ_1 bits and t_2 symbol errors each flipping at most ℓ_2 bits; the regime with $t_1 >> t_2$ and $\ell_1 < \ell_2$ is of interest in Flash. Observe that this specification defines the error-correction capability of a code in a much more precise way than what is allowed by the conventional t-error correcting moniker. In the context of the example above, we could parameterize the error pattern $(000, 010, 000, 011, 001)$ via $t_1 = 2, t_2 = 1, \ell_1 = 1$, and $\ell_2 = 3$.

It was shown in [31] on real experimental data, that this tensor-product construction with 4 constituent matrices chosen in a way that mimics the Flash behavior, leads to lifetime increase of at least 40%. Even further, the resultant tensor-product code has two highly desirable features from the implementation standpoint: the guaranteed error correction (with respect to the more finely specified error patterns), and low-complexity encoding and decoding algorithms. The latter property is a consequence of the fact that the proposed tensor-product codes are built from simple symmetric error correcting codes. Figure 6 shows performance results of applying TPC codes from [31] to TLC Flash. The codes are all of length 4096 and have rate 0.86. It is especially interesting to point out that the TPC construction outperforms not only good non-binary and binary BCH codes, the latter derived by using the same code over the three pages, but that it also outperforms the best combination of three BCH codes, one for each page, where BCH codes with different error correction capabilities are assigned to different pages – while the other three codes hit error rates of 10^-6 and higher much earlier, TPC codes offers excellent reliability the longest (i.e., no errors were observed). This is precisely because TPC code can correct certain error patterns spanning multiple bits per cell with less redundancy than three parallel codes can. Tensor-product constructions can be further customized for Flash. For example, a simple transformation of the tensor-product operation allows for limited programming into certain cells [32]. This operation is particularly beneficial for Flash memories that have a small number of defective cells, which can needlessly consume a disproportionate amount of error correction.

The codes we showed for limited-magnitude errors earlier in this section can actually be combined into the flexible tensor-product construction of the latter part. A similar graded error-correction profile can be obtained where ℓ_1 and ℓ_2 represent other types of errors besides symmetric bit errors.

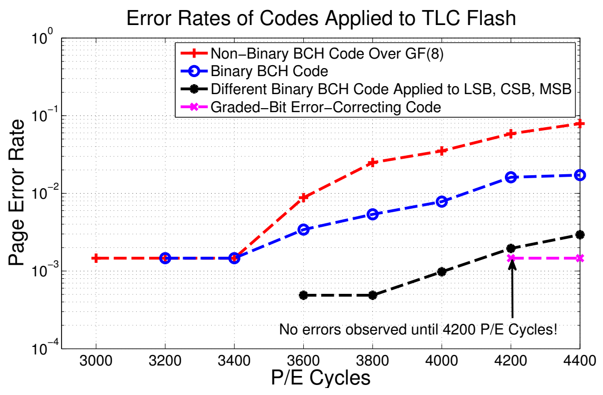


Fig. 6. Simulation results showing the benefits of using graded-bit error correcting codes in Flash. MSB/CSB/LSB refers to most significant/center significant/least significant bits. Figure derived based on results from [31].

C. Additional promising algebraic-coding schemes

We now briefly comment on other algebraic methods that have shown promise over the baseline BCH codes. The majority of these works are on the exploration of appropriate combinations of known coding tools, using to a large extent idealized channel assumptions, and not explicitly focusing on the code design tailored specifically for asymmetric Flash.

Code concatenation is a powerful technique exploited in the two memory codes we previously described. Other approaches of a related flavor include the following. Product code refers to a construction based on two constituent codes $\mathcal{C}_1$ and $\mathcal{C}_2$ such that each row of the resultant code is a codeword of $\mathcal{C}_1$ and each column of the resultant code is a codeword of $\mathcal{C}_2$. An attractive feature of a product code is that its minimum distance is the product of the minimum distances of the two constituent codes, and that it has an efficient iterative row/column decoder that can correct with high probability many more errors than half the minimum distance [12]. One of the first results on the product codes for Flash is the work in [33], which demonstrated via simulations on a synthetic channel the potential gains over a plain BCH-coded scheme when a 2-step coding is employed: a BCH code is used across rows and a simple Hamming code is used across columns in a way that more error patterns of interest can be corrected relative to the uninformed BCH code. The architecture proposed in [33] also allows for parallel processing of multiple codewords, thus reducing the overall latency. Additional progress on concatenated BCH codes was made in [34], which also exploited the property that a combination of weaker, shorter (and hence cheaper) BCH codes codes is competitive with one stronger, longer (and thus more expensive) BCH code. Another interesting twist on product codes was recently explored in [35], where it was shown that so-called half product codes have better minimum distance properties than their (full) product counterparts.

Complementing theoretical investigations on BCH-enhanced designs, several recent works and industry patents have explored performance benefits and implementation issues of concatenated/product codes in the context of Flash [36], [37], [38], although likely primarily in the idealized settings.

Intra-cell variability can also be exploited by trellis coded modulation (TCM) [39], another idea from classical communications theory – TCM limits the magnitude of errors in a way that is relevant in Flash [40]. Benefits of the BCH-TCM concatenated schemes over the baseline Reed-Solomon/BCH-coded system was demonstrated in [41], [42], [43], [44]. Since TCM requires some amount of redundancy, concatenated schemes with a TCOM component could be of interest in Flash architectures that permit additional threshold levels and can tolerate rate loss incurred by the TCM component.

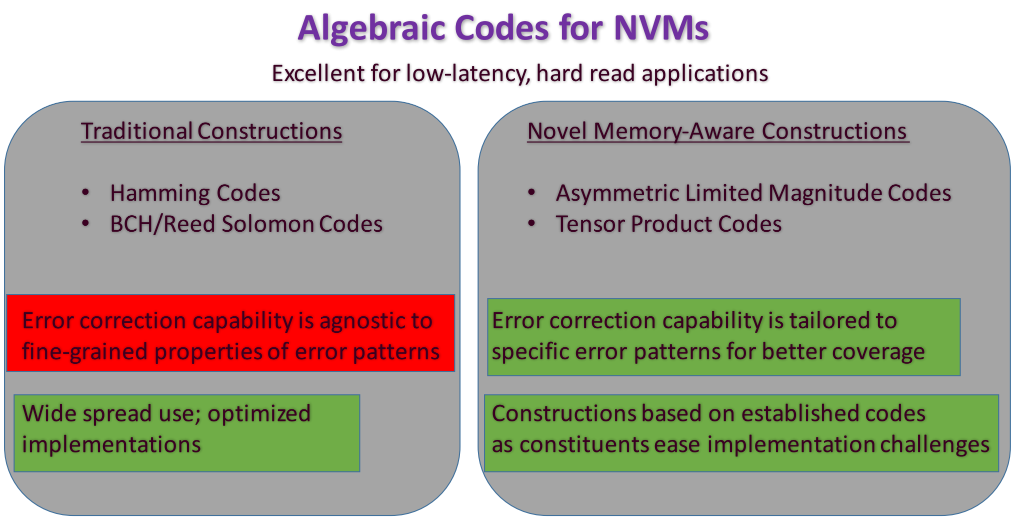
Even with the implicit emphasis on the symmetric noise model, these coding techniques already demonstrate potential in NVM applications; a compelling open research question is how to best utilize them in the channel-aware way.

Building upon the results presented in this Section, we summarize the properties of classical and modern algebraic codes in Figure 7.

V. Error Correction with Graph Codes

A. Classical graph codes: LDPC codes and iterative decoding

Like previously discussed algebraic codes, low-density parity-check (LDPC) codes are also linear block codes. They can also be binary or non-binary, depending on whether the information is organized in bits or in symbols. LDPC codes are described by a sparse parity-check matrix, hence the “low-density” adjective. It is especially convenient to view an LDPC code as a bipartite graph where one set of nodes, called *variable nodes*, corresponds to the columns of the parity-check matrix, and the other set of nodes, called *check nodes*,



Algebraic Codes for NVMs

Excellent for low-latency, hard read applications

Traditional Constructions

- Hamming Codes
- BCH/Reed Solomon Codes

Error correction capability is agnostic to fine-grained properties of error patterns

Wide spread use; optimized implementations

Novel Memory-Aware Constructions

- Asymmetric Limited Magnitude Codes
- Tensor Product Codes

Error correction capability is tailored to specific error patterns for better coverage

Constructions based on established codes as constituents ease implementation challenges

Fig. 7. Summary of main algebraic codes and their key properties in the context of NVMs. Advantages are highlighted in green and disadvantages in red.

corresponds to the rows of the parity-check matrix. An edge between a variable node and a check node exists if and only if the corresponding entry in the parity-check matrix is non zero. Concretely, an edge between variable-node i and check-node j marks that the i-th code symbol participates in the j-th linear check equation of the code.

In the binary case, the parity-check matrix is the adjacency matrix of this bipartite graph. In the non-binary (or q-ary) case, each non-zero entry in the parity-check matrix is a non-zero element of a Galois finite field GF(q), and the corresponding edge in the bipartite representation of the code has this non-zero value as its label. The sparse graphical representation of the code enables low-complexity iterative decoding algorithms, executed as a series of message-passing steps alternating between the set of variable and the set of check nodes. The exchanged messages are proxies of the likelihoods of the values of the variable nodes; in practice, computations are performed in a transformed domain and the messages represent log likelihood ratios (LLRs). Message exchange terminates when all the checks are satisfied in the sense that the linear equations associated with them hold. NVM channels are especially natural for representing the code symbols as q-ary symbols, which calls for the use of q-ary LDPC codes. It is generally understood that q-ary LDPC codes offer significant performance benefits over their binary counterparts, at the expense of substantially increased decoder complexity.

Owing to their excellent performance, LDPC codes have already found phenomenal success in many modern data transmission applications. It is thus not a surprise that LDPC codes are actively being considered in modern NVMs as well, with a number of industry-based patents recently issued on this topic, see e.g., [45], [46], [47], [48], [49], wherein the focus has mostly been on binary LDPC codes.

LDPC codes offer most benefits when decoded using real-valued LLRs, i.e., with the initialization and the messages

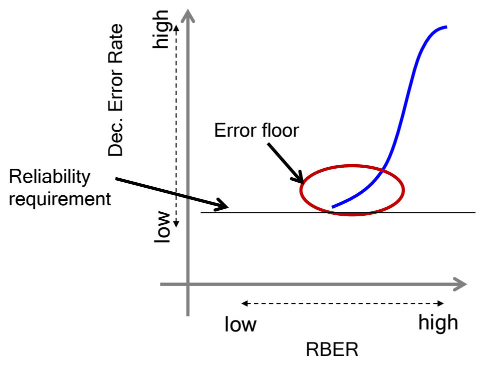


Fig. 8. Typical performance plot of unoptimized LDPC codes.

expressed in full precision. However, read information about Flash channels is obtained through a sense amp that can only report whether the threshold voltage of a cell is below or above some value, information that is intrinsically discrete (see also Section II). As a result, the channel that the LDPC decoder sees is inevitably discrete. A non-trivial question then is where to place threshold voltages as a function of the number of available reads in order to maximize the utility of memory devices, see also Figure 3. One mathematically precise yet intuitive idea is to assign threshold voltages exactly in the way that would maximize mutual information between the input and the output of the induced discretized channel [50]. Placement of threshold voltages is also important for the code design. As we discuss next, code design and optimization critically depends on proper channel modeling.

B. Graph codes for NVM error models

LDPC codes are very powerful error-correcting codes because they mimic Shannon-optimal random codes, with the added feature of low-complexity decoding. They have also been around for sufficient time so that their design for classical channels has been nicely perfected by a massive body of research. Despite this favorable state of matters, the application of LDPC codes to NVMs motivates interesting new fundamental research. NVM distinctive error models and unique operation modes necessitate the enrichment of the constructive toolbox for LDPC codes, and also their analysis. Non-binary LDPC codes are ideally suited for multi-level memories. We thus focus in this sub-section on two promising directions for NVM LDPC codes: one is their finite-length design of non-binary codes optimized for common error types, and another is the design of non-binary codes optimized to the multi-bit structure of the Flash MLC/TLC architectures. Other interesting avenues are discussed in the next sub-section.

1) Finite-length code design for NVM errors: It is well known that practical LDPC codes, both binary and non-binary, suffer from the so-called "error floor", manifested as a failure of the code to lower the output error rate sufficiently when the input error rate is very low [51]. This undesirable behavior is especially problematic for modern Flash devices as the flooring effect prevents the system from meeting target reliability constraints, see the schematic Figure 8. Here RBER denotes raw bit error rate and Dec. Error Rate denotes residual errors after LDPC decoding. The unwanted error-floor effect is due to the fact that the low-complexity iterative decoding algorithm operates on the LDPC bipartite graph which inevitably has cycles. (We quickly remark that this issue vanishes in the infinite block-length regime where one assumes that the bipartite graph is essentially cycle-free. In this regime, the elegant theory of density evolution offers crisp code performance characterization [52]. This theory critically depends on the cycle-free assumption and is not directly useful in the finite-length setting.)

The issue of the error floor is particularly problematic for applications that need to operate under stringent constraints on reliability, including modern NVMs. Extensive prior work was performed on the analysis of the LDPC error floor, implicitly assuming the transmission over a symmetric channel. *Trapping/absorbing sets* is the terminology (e.g., [51], [53], [54], [55]) adopted in the coding literature used to refer to combinatorial objects that exist in the bipartite representation of the code that trick the iterative decoder into making decoding errors. Trapping sets encompass convergence to non-codewords and oscillations among different configurations [51]. Typically, oscillation errors can be suppressed with a more informed quantization scheme [56]. The definition of the absorbing sets [55] is purely combinatorial and it refers to objects that are fixed points of certain practical decoders, notably including detrimental non-codewords. These configurations are locally consistent (from a vantage point of an individual node) but are not necessarily globally consistent in the sense that they need not produce a codeword. As a result, during the decoding, some of the checks remain unsatisfied despite repeated iterations of the message-passing decoder. The configurations are typically characterized by a certain number of variable nodes a connected to a certain number b of unsatisfied checks; a codeword is a special case of such a configuration with $b = 0$.

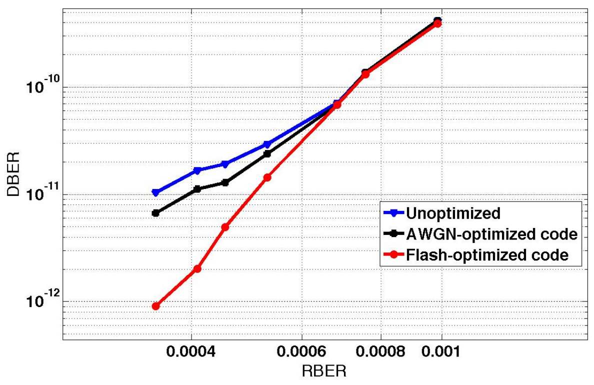


Fig. 9. Benefits of channel-aware LDPC code optimization on a realistic MLC Flash channel model.

Intriguingly, it is often the case that absorbing sets with small a and with $b \neq 0$ cause decoding errors in iteratively-decoded LDPC codes – that is, there exist problematic configurations with weight less than the code minimum distance. In other words, in contrast to traditional coding theory principles, quantifying the goodness of a code in terms of distances between codewords is insufficient in the case of iteratively-decoded graph codes.

As argued before, NVM channels fundamentally differ from their oft-utilized symmetric counterparts, further complicating LDPC code optimization techniques. Despite a common practice of using AWGN-optimized LDPC codes on a Flash channel, the approach is grossly inadequate. The reason for this rests with a closer investigation of problematic objects for the two types of channels. The type of absorbing sets causing decoding errors and in turn the pesky error floor is significantly different for the two channels. For example, for AWGN-like channels, due to noise symmetry, dominant absorbing sets are those that have a small number of variable nodes a, and for each such variable node, there are more satisfied than unsatisfied neighboring check nodes. In contrast, for Flash-like channels, due to asymmetry, categorization of absorbing sets into problematic and non-problematic is topologically more subtle, [57]. As a result, code optimization focused on the removal of AWGN-detrimental structures is essentially useless if the code is to be used over the highly asymmetric Flash channel. Fortunately, as in the case of algebraic codes, substantial gains can be made once the code is designed in a way that is cognizant of the channel characteristics, as we illustrate in Figure 9, where we plot raw BER (RBER) against "decoded" BER (DBER); the latter is the error rate after the decoding step.

In this example, as an instance of widely popular quasi-cyclic designs, codes are designed according to [58], which offers implementation-friendly, circulant-based structure of the parity check matrix. Since the target application is MLC Flash, the codes are non-binary and are defined over the finite field of size 4. Additionally, they have rate 0.9, length 4000 bits, and variable-node degree equal to 3. Without any further channel-aware optimization, the performance is as shown by the top curve (in blue).

A promising technique for non-binary code optimization consists of two steps. First, choose the ordering of the constituent circulants in the parity-check matrix to minimize the number of possible detrimental configurations in the bipartite representation of the code. Second, assign edge labels (from the finite field of size 4 in this example) to ensure the non-existence of detrimental configurations. This technique is attractive as it results in a design that preserves all desirable code properties (length, rate, circulant organization of the parity check matrix, node degree regularity), and moreover can be described in crisp combinatorial terms – recently developed frameworks [59] and [57] are based on succinct linear-algebraic description of the absorbing set, so that the provable elimination of possibly numerous instances of the detrimental structure can be achieved by controlling the null space of one simple matrix. As a result, the optimization protocol is highly computationally efficient, systematic, and can at once produce a whole family of parity check matrices with the desired properties. In fact, mathematical characterization of absorbing set/trapping set topologies is more tractable for codes with lower variable node degree [60], [61]. The codes that have high rate – the rate regime in which NVMs need to operate – imply low variable node degree, thereby making combinatorial optimization of graph codes aimed at handling bad configurations especially well suited for NVM applications.

Even with a fast optimization algorithm in place, the key question to answer is what configurations one should optimize for. The answer is highly channel-dependent and the more the channel differs from the AWGN setting, the more diverse the problematic objects are relative to their AWGN counterparts. In the context of our example, optimizing this code by only removing absorbing sets that are problematic in the AWGN setting results in the middle curve of Figure 9 (in black), which roughly corresponds to the elimination of sets with (a,b) parameters being (4,2) or (4,4). This offers only modest improvements on the Flash channel (modeling akin to [4]), whereas optimization that removes a broader collection of objects that are truly problematic in the Flash domain gives the lowest curve (in red), which reflects order of magnitude improvement while maintaining all other structural code properties. This optimization targets absorbing sets with (a,b) parameters with $4 \leq a \leq 7$ and $1 \leq b \leq 4$. Combinatorial strategies for the removal of problematic configurations in the non-binary domain are substantially more involved than in the binary case; they are discussed in [57].

Beyond effective performance-improvement tools for accepted code constructions, NVM coding performance can greatly benefit from tools that illuminate the underlying constructive considerations. We next show, using a new theoretical framework, how q-ary LDPC codes should be designed when the *multi-bit structure* of a q-ary channel is explicitly taken into account.

2) Non-binary codes with multi-bit structure: Deeply ingrained in the Flash architecture is the duality of *binary logical* pages stored on *q-ary physical* pages. A common choice by SSD vendors is to map $\log_2 q$ binary logical pages – for example 3 pages in $q = 8$ TLC – to a single page of q-ary cells. The main motivation is access benefits: allowing lower-latency access to a logical page before the physical cells are fully read. This is possible because the unit of *bit* is naturally expressed in the physical processes, for example a read primitive that returns a bit of information comparing a cell threshold level to a reference value. However, even in the presence of smart gray-like mappings, we lose in error-correction efficiency when employing a binary code for each logical page individually. As was the case in Section IV-B for algebraic codes, ignoring the features of the q-ary channel in code design is sub-optimal and inefficient. We instead want to deploy the code on the q-ary physical page, but in a way that considers the underlying bit structure of the physical processes. In other words, we want to design LDPC codes that are defined over q-ary alphabets, but designed for channels preserving the bit-structure of the read/write processes. This will offer improvement over the known approaches of either 1) use a q-ary LDPC code designed for symmetric errors, or 2) use a hierarchy of binary LDPC codes through the concept of multi-level coding [62]. The key is that the new approach gets the best of both worlds: it enjoys the inherent advantage of q-ary LDPC codes, and it optimizes the code design to the true underlying channel.

Making progress with design of LDPC codes for NVM channel models is most promising by first defining new *erasure* models corresponding to the channel errors. This has been the case with binary LDPC codes, for which performance analysis over the *binary erasure channel (BEC)* contributed most insights and design practices [63]. The analog of an erasure in our case is a *partial erasure*, which represents a read where the cell level is not fully resolved but also not completely unknown [64]. Given that a cell level can be any symbol in the set $\mathcal{Q} = \{0, 1, \ldots, q-1\}$, a partial erasure is a subset of $\mathcal{Q}$ whose contents are the possible levels for that cell after the read. A subset of size q represents the standard q-ary (full) erasure, and a subset of size 1 represents the no-erasure case where the cell level is perfectly known. All subset sizes in between those two extremes are the partial erasures we find useful in our code design. Note that a partial erasure is a useful proxy for a structured q-ary error, similarly to a full q-ary erasure being a good proxy for symmetric q-ary errors. To model error channels with a multi-bit structure, we consider the following definition of a partial-erasure channel, which we call here *q-ary multi-bit erasure channel* [65]. Let $q = 2^s$, where s is the number of bits mapped to each cell level. For convenience, let us take the special case of $q = 8$ and $s = 3$ (TLC). Suppose that a cell has the level $x = 0$ stored in it. Then the channel output y is either $\{0\}$ representing perfect readout, or $\{0, 1\}$ representing a partial erasure missing the least-significant bit of x, or $\{0, 1, 2, 3\}$ representing a partial erasure missing the two lower bits, or $\{0, 1, \ldots, 7\}$ representing a full erasure missing all three bits. We get $\{0, 1\}$ with probability ϵ_1, $\{0, 1, 2, 3\}$ with probability ϵ_2, $\{0, 1, \ldots, 7\}$ with probability ϵ_3, and $\{0\}$ with probability $1 - \epsilon_1 - \epsilon_2 - \epsilon_3$. Note that this is a generalization of the symmetric case that can accommodate the variable reliabilities among the three bits. In particular, it captures the property of the q-ary channel that a given error magnitude affects all bits from some significance level and downward. To combat real NVM errors we will set values of $\epsilon_1, \epsilon_2, \epsilon_3$ according to the media properties, and design a code that corrects such error events with high probability.

Some ingredients need to be developed to enable code design for q-ary multi-bit channels. The first is an iterative decoder that extends the efficient message-passing algorithms of symmetric channels to the new channels. For q-ary partial-erasure channels such an extension is provided in [64]. Secondly, we need an efficient analysis framework that can tell the performance of code ensembles over the new channels. In [65] such an analysis is developed based on density evolution [63], with a careful exploitation of the channel structure to reduce the analysis complexity that otherwise blows up quickly with q. Lastly, and most importantly, we need to find ways by which the analysis framework can be used to design better codes for the new channels. An interesting example for this is the following crisp design rule from [65]: for $q = 4$ ($s = 2$,MLC), if the multi-bit erasure channel has a dominant occurrence of single-bit erasures ($\epsilon_1 \gg \epsilon_2$), then the edge labels of the q-ary LDPC code must *not* be selected uniformly from the non-zero field elements $\{1, 2, 3\}$, but rather uniformly over two of the elements, e.g. $\{1, 2\}$, with no labels selected as the remaining element 3. It is not clear a priori why this rule should apply, but it is provably correct given the analysis framework. A more comprehensive design tool building on the new analysis framework optimizes the code degree distributions taking into account the parameters of the partial-erasure channel. It has been shown [64] that degree distributions obtained through this dedicated optimization have superior decoding thresholds and error rates compared to codes that were designed for the standard erasure channel.

Moving to finite block-length optimization of LDPC codes for multi-bit channels, we seek algorithms that improve the code specifically for the more common error types. In this part we build upon the erasure interpretation of the channel, and study how the well-defined configurations called *stopping sets* [66] can be mitigated in the case where we have additional knowledge on the erasure types.

A stopping set is defined as a subset of the variable nodes that collectively connect to a set of check nodes each of which has degree more than 1 to the variable-node subset. Stopping sets are detrimental for iterative decoding, because if all variable nodes in them are erased, the decoder cannot continue iteratively. Examining an iterative decoder operating over a q-ary partial-erasure channel, we observe that a stopping set existing in the graph can be neutralized by carefully setting the edge labels to not halt the iterative decoder. This is true only for partial erasures, and does not apply to codes for the q-ary (full) erasure channel (any stopping set for the binary erasure channel is also a stopping set for the q-ary erasure channel, for any edge-label combination). Following a detailed characterization of the label sets that resolve stopping sets for the multi-bit channel, we have developed an algorithm that sets edge labels in a specific code graph to remove stopping sets of small sizes. Note that this label optimization can be done *on top and beyond* other known stopping-set reduction techniques applied to the code graph (e.g. [67]). In Figure 10 we show sample results showing the potential advantage of this coding scheme. We take a regular LDPC code with check-node degree 18 and variable-node degree 2 (rate 8/9), and run our labeling algorithm removing partial-erasure stopping sets for $q = 8$. We plot the symbol-erasure rate (SER) at the decoder output as a function of the probability ϵ_1 that a symbol undergoes a 1-bit erasure. The performance after the label optimization improves significantly in most of the ϵ_1 range. We also compare the performance to a binary code drawn from the same ensemble, but with triple length (to get the same number of bits), and third the erasure rate (to get the same expected number of bit erasures). It is shown that the binary code is not competitive to the q-ary option, even though the errors we considered here are single-bit errors. This motivates further research constructing stronger and more practical LDPC codes that perform well over multi-bit channels. An interesting open research question is how to best combine the combinatorial framework and the new bit-level interpretation introduced in the last two subsections, respectively, for the ultimate LDPC solutions.

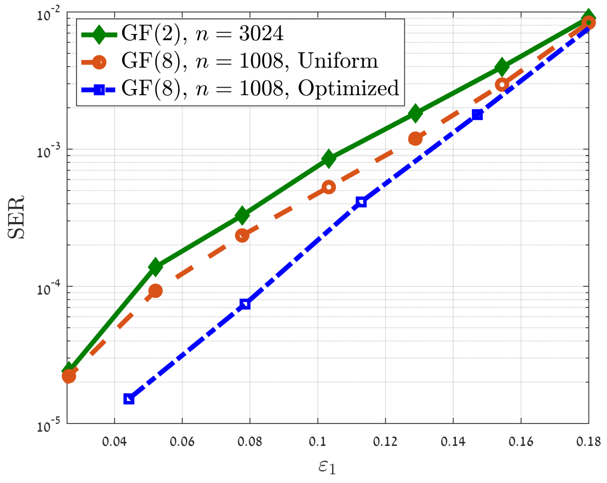


Fig. 10. Performance of 8-ary LDPC codes before (dashed) and after (dash-dotted) removal of stopping sets affecting the multi-bit erasure channel. In comparison, a binary LDPC code that is 3 times longer (solid) has much worse correction performance.

C. Additional considerations and future directions for graph codes

Complementing proper channel code design is the implementation of associated decoders. In the unoptimized case, excellent LDPC code performance comes at the prohibitively high implementation complexity of the decoder that exceeds latency, hardware footprint, and power consumption allowed in modern NVM applications. One way to reap the benefits of LDPC codes while maintaining target latency is by use of coarse decoding and lookahead computations when channel conditions permit, as recently proposed in [68]. The idea is that since the hard-decision decoder is successful most of the time, one would only invoke additional soft-information LLRs when the hard-decision decoder fails to decode. To minimize the latency, this additional information is computed concurrently with the baseline decoder, and used only as necessary. The work in [68] also quantifies the impact of progressive sensing on the overall power consumption of the LDPC decoder. A recently proposed technique for reducing decoder latency in TLC Flash operates directly on the soft decisions: soft information is generated by only using center read references. By interleaving the three pages, errors are effectively evenly spread across the pages [69].

Further architectural solutions for LDPC decoders in NVMs include a circumspect combination of throughput-enhancing techniques, such as strategic message update, dynamic scheduling, use of lookup tables, code structure-aware parallelized structure, and local error correlations, among others, [70], [71], [72], [73]. Additionally, new formulations of iterative decoders that are well suited for limited precision implementations, such as finite precision decoders proposed in [74], [75], will further help in the broader adoption of LDPC-coded systems. Another fruitful research direction would be to optimize LDPC decoders specifically from the point of view of recovery from non-codeword errors dominant in the NVM error models analyzed in the previous section [57]. This could be done, for example, by opportunistically pruning computations in high-performance but costly non-binary LDPC decoders [76], or by using the intra-cell variations in the LLR scalings [77].

Spatially-coupled (SC) codes (also known as LDPC convolutional codes) are the newest exemplar of graph designs; they offer excellent performance in a variety of settings. These codes are obtained by chaining together bipartite graphs each corresponding to a smaller LDPC code. This concatenation results in structured irregularity that has led to capacity-approaching performance in the asymptotic setting [78]. Moreover, SC codes are amenable to low complexity window decoding with message passing decoder operating on the block constituents [79]. Initial results on the optimization of SC codes in NVM and related applications already show promise [80], [81], and a thorough study will likely lead to significant results.

Additionally, the power of recently invented polar codes has not yet been fully explored in the context of NVMs. Another recent work has proposed the use of non-linear polar codes for asymmetric channels potentially suitable for Flash memories [82] and the work in [83] offered the first study on using polar codes as the error correction technique in Flash memories. Comprehensive analysis of polar codes and polarization principles in the context of NVMs could be another interesting open research direction, provided issues stemming from the higher complexity and decoding latency of polar codes can be adequately addressed.

As an analog to the summary of the algebraic codes given in Figure 7 and based on the discussion in this section, we sum up the key features of graph codes for NVM applications in Figure 11.

VI. Re-write codes for the in-place update feature

Since the early days of data storage, density scaling has always meant challenges to data reliability. But in modern storage media starting from Flash, competitive density also means significant compromise to *access performance*. The best known access-performance issue in Flash storage is the inability to perform *erase* operations (remove charges from cells) at the same small granularity of the *program* operation (add charges to cells). While helpful for storage density, this restriction is extremely limiting for access performance, because data cannot be updated in-place. In fact, a vast amount of research in the storage-systems field is devoted to circumventing this restriction in applications where it prohibits adequate performance. Coding enables a more direct solution for this restriction, through the use of *re-write codes*.

A. Re-write and WOM codes

To solve the write-access problem stemming from restricted erase operations, coding needs to bridge between the restricted physical media and the unrestricted user data written to the storage. The user may want to update data arbitrarily by re-writing a data unit, and the code provides a representation for the data that adheres to the restriction to only add charges to the physical cells. It turns out that a model known since the 1980's called *write-once memory* (WOM) [84] coding is highly applicable to the problem of update-restricted Flash storage. In the WOM model, information can be written t times on a block of n binary physical cells, such that physical cell levels change from 0 to 1, but not from 1 to 0. By applying the WOM model to Flash[2], the user can write t times to the same physical cells without requiring a slow and costly erase operation. Thus such codes hold great potential to improve the performance and life span of storage devices. The design objective for a t-write WOM code is to maximize the *sum rate*, which is the total amount of information (in bits) written to the n cells in t writes, divided by the number of cells n. Toward this objective, several new theoretical constructions with good sum rates have been proposed. For example, recent works in [85], [86], [87], [88], [89] provided high sum-rate WOM codes based on careful adaptation of powerful coding theoretic constructions, and by clever compositions of simple WOM codes into stronger ones, e.g., constructing multiple-write WOM codes from two-write WOM codes, constructing non-binary WOM codes from binary WOM codes, and others.

For application in multi-level memories, q-ary WOM codes are of interest. The q-ary generalization of WOM[3] was defined in [90], where cell levels are restricted to only change in the *upward direction*. Note that when q is a power of 2, for example $q = 8$ in the TLC technology, it is possible to use the q-ary cell as multiple bits in a binary WOM code (3 bits in TLC) without violating the update restrictions. However,

[2]We adopt the convention that an erase operation *decreases* the cell level, which may be different from the convention in the memory-devices community (but fully equivalent to it).

[3]Note that WOM is a misnomer for non-binary codes, because the physical cells are no longer limited to be written only once.

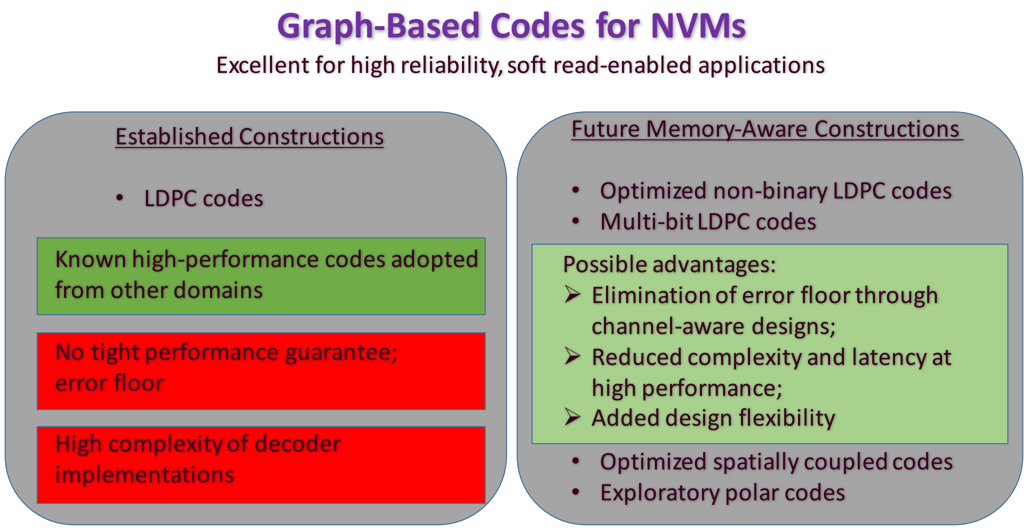

Graph-Based Codes for NVMs
Excellent for high reliability, soft read-enabled applications
Established Constructions
LDPC codes
Known high-performance codes adopted from other domains
No tight performance guarantee; error floor
High complexity of decoder implementations
Future Memory-Aware Constructions
Optimized non-binary LDPC codes
Multi-bit LDPC codes
Possible advantages:
Elimination of error floor through channel-aware designs;
Reduced complexity and latency at high performance;
Added design flexibility
Optimized spatially coupled codes
Exploratory polar codes

Fig. 11. Summary of main graph codes and their key properties in the context of NVMs. We highlight advantages in green and disadvantages in red. We reserve "light green" for the possible advantages of emerging LDPC designs as these theoretical constructs need to also be validated in practice.

this is inefficient because it is well known that using larger alphabet sizes improves the re-write sum rate for a given amount of physical storage [91]. Coding results for the q-ary model appeared in [91], and later in [92], [93]. A theory based on lattices aiding the construction of q-ary codes was developed in [94]. In addition to q-ary WOM, there are other re-write coding models applicable to multi-level memories. In the model of *floating codes* [95], the code supports t writes, but in each write only a *single bit* out of k information bits is updated. Other re-write models for multi-level memories were studied in [96], [97], [98]. Most recently, [99] developed codes based on the coset coding idea to improve memory lifetime.

A re-write code in the q-ary WOM model is defined by the parameters q, n, t and $\boldsymbol{M} = [M_1, \ldots, M_t]$. Parameter n is the number of physical q-ary cells in the memory word used by the code. Parameter t is number of times the memory word can be written to, and the vector $\boldsymbol{M}$ specifies for each of the t writes the number of possible values of the input information. In the sequel we focus on the practical case where in all t writes we have the same input size, that is, $M_1 = M_2 = \cdots = M_t = M$. With this restriction the code is called *fixed rate*, and its parameters are denoted with the tuple (q, n, t, M), where M is a scalar integer. We also define $k = \log_2 M$ and say that k is the number of input information bits. In practical use, once a (q, n, t, M) code exhausts its t writes, the n cells may not be further reused without an external erase operation, which is not an explicit part of the model (but does happen in practical use in Flash).

The motives to use WOM codes in Flash are compelling: it has been demonstrated in the literature that with a clever use of WOM codes in an SSD, the *write amplification* can be reduced significantly [100]. In addition, implementing WOM codes in an SSD simulator has shown significant advantage in *write throughput* [101]. With this promise come some non-trivial challenges. Probably the greatest concern in deploying WOM codes is the impact on data reliability. Operating a WOM code implies re-writing data *in place* and no longer in a pure sequential order, and this may introduce new issues of disturbs and inter-cell interference. Adding to that concern is the fact that constructing error-tolerant WOM codes is not an easy task. Coding schemes that combine re-write and error-correction capabilities exist in the theoretical literature [102], but are not practical enough for implementation. Combining the two features by concatenating a WOM code and an ECC is also problematic: an outer WOM code means that ECC parity bits computed from the WOM codeword will violate the WOM constraints; an inner WOM code means that small channel errors can propagate to massive error events by the WOM decoder. A good potential solution around these issues is to use *short q-ary WOM codes*. If we use a short inner WOM code, then channel errors cannot propagate beyond the small WOM block length, and concatenation with a long outer ECC can work well. It turns out that q-ary WOM codes can have attractive re-write capabilities even if they use as few as $n = 2$ cells. We demonstrate this next.

To specify a WOM code, one needs to provide a pair of functions: the *decoding* and *update* functions. We define the decoding function as $\psi : \{0, \ldots, q-1\}^n \to \{0, \ldots, M-1\}$, which maps the current levels of the n cells to one of the M possible information values. The update function is defined as $\mu : \{0, \ldots, q-1\}^n \times \{0, \ldots, M-1\} \to \{0, \ldots, q-1\}^n$, specifying how the cell levels need to change as a function of the current cell levels and the new information value at the input (here again the input is taken from a set of M possible values). The update function needs to satisfy the WOM constraints of not moving a cell to a lower level. Let us consider the special case of $q = 8$ (TLC), $n = 2$ (2 cells), and $M = 8$ ($k = 3$ information bits per write).

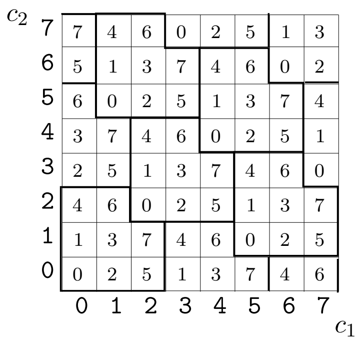


Fig. 12. Decoding function of a $(q = 8, n = 2, t = 4, M = 8)$ code. 4 guaranteed writes is optimal for the code parameters.

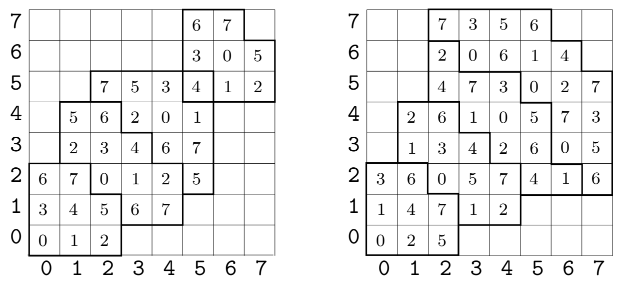


Fig. 13. Decoding functions of two more $(q = 8, n = 2, t = 4, M = 8)$ codes. The left code is designed to reduce ICI, and the right one can give $t = 5$ if q grows to 9.

Note that n is the block length for coding purposes only, and a page with $N \gg n$ cells can be used with multiple WOM blocks in parallel. A convenient way to represent a decoding function $\psi : \{0, \ldots, 7\}^2 \rightarrow \{0, \ldots, 7\}$ is by a two-dimensional matrix where a position (c_1, c_2) represents the physical levels of the two cells, and the numbers in the matrix are the information values returned by the decoding function. For example, Figure 12 shows a decoding function obtained by tiling the $q \times q = 8 \times 8$ matrix with a polygonal shape with area $M = 8$. To make it a decoding function of a WOM code, we need to define on it an update function that only moves upward and to the right in the matrix. Given a current matrix position, the encoding function takes an input value and needs to find it in a position neither below nor to the left of its current position. In [103] an update function was given for the decoding function in Figure 12 that guarantees $t = 4$ writes with any sequence of input values. For example, a sequence of 4 writes with the input values $6 \rightarrow 4 \rightarrow 7 \rightarrow 3$ will be written by updating the cell levels with the sequence $(1, 2) \rightarrow (2, 4) \rightarrow (3, 6) \rightarrow (7, 7)$. Hence this is a $(q = 8, n = 2, t = 4, M = 8)$ code. It was also shown in [103] that $t = 4$ is the maximum possible number of writes given the other code parameters, hence this is an optimal code. Despite the extremely short length of this code, the 6 bits that it consumes (2 cells, 3 bits each) are within 0.65 bit from the information-theoretic fundamental limit of *binary* fixed-rate WOM codes [104], which is only attainable with very long and high complexity codes.

Interestingly, the code shown in Figure 12 is not the only option for getting a $(q = 8, n = 2, t = 4, M = 8)$ code. Without losing anything in the number of writes, we can construct other codes that offer additional useful features. As two examples we take the codes depicted in Figure 13. The code on the left guarantees in addition that the two cells will be balanced to be *at most* 3 *levels apart* throughout the write sequence [105]. This feature reduces inter-cell interference (ICI) between the cells, which is known to be more significant when the two cells have large level differences [11]. The code on the right is designed with the feature that increasing the number of cell levels to $q = 9$ can add a fifth guaranteed write (the previous two codes cannot add a write with one more level). This shows that the use of WOM codes opens the way to using cells with numbers of levels that are not necessarily powers of two, and such uses can actually give good performance with a simple implementation. Moreover, this example motivates considering WOM codes with additional features, for example maximizing the data reliability when an outer error-correcting code is employed.

We end this section with the remark that a related technique called “flip-N-write” was successfully proposed for phase change memories [106]. In this simple but powerful scheme, either a desired word is written or its complement, depending on which one would be faster to write. One bit of redundancy is used to indicate whether the intended word or its complement are being written.

B. Other coding schemes to watch for the future

We now briefly discuss two additional coding mechanisms that are of interest in NVMs: constrained coding and rank modulation. Constrained coding for NVMs is strongly motivated by the pronounced amount of ICI. ICI is caused by parasitic capacitances between physically adjacent cells in the Flash chip. As a result, when charge is added to a cell (during programming), the charge levels of neighboring cells may inadvertently increase as well. The amount of this unintended charge is a function of device parameters and design but has a roughly inversely proportional relationship to the physical distances between the cells [107]. As a result, as Flash technology is scaled down, the ICI becomes increasingly more pronounced. One way to overcome the adverse effects of ICI is by preemptively preventing certain patterns to be written in the first place. Constrained coding is a branch of information theory that precisely answers the question of maximizing data transmission/storage while ensuring that undesirable subsequences are never stored. Constrained coding techniques have already been successfully deployed in other more conventional data storage technologies, such as HDDs [108], and as with other existing methods mentioned before the challenge is to design constrained coding methods that accurately address technology-specific particularities. In the context of Flash, one seeks to avoid ”High-Low-High” patterns. This has led to the development of elegant mathematical theory of constrained systems, as in [109], where the focus was on characterizing the set of sequences that are free of detrimental patterns. Recent results on construction of constrained codes for NVMs are presented in [110], [111], [112]. A challenging open question is to transfer the results from the asymptotic domain to the practical finite-length setting while offering codes with minimal rate penalty and easy encoding/decoding.

The special physical properties of the Flash channel have recently motivated an exploration of a different type of data representation: rank modulation [113]. The idea in rank modulation is to represent information as the relative ranking of a cell with respect to the entire block, rather than as the absolute amount of charge in a particular cell. Information is stored in permutations, and is read by comparing the values of different cells in blocks. Ordering-based representation has many advantages including the fact that charge leakage, which affects all cells at roughly equal rates, will not change the relative ranks of cells, only their absolute values. One distinct concern regarding the implementation of rank modulation techniques is the need to have very finely grained comparators, which are currently impractical. If this key issue is resolved, many fascinating recent theoretical results on rank-modulation codes [114], [115], [116] could then be used in practice.

VII. Conclusion and Perspectives

In this survey paper, we reviewed several recent exciting developments in coding methods for non-volatile memories. The need for novel coding schemes is by now clear to the memory industry, which has already advanced research and development in this area considerably, including commercialization of BCH codes, LDPC codes, constrained codes, and various concatenated codes. While specific details of code constructions remain carefully guarded trade secrets, numerous industry patents on this topic offer a glimpse into practical deployment and importance of various ECC methods: for LDPC, among many others, these patents include [45] (LSI corp.), [46] (Intel Corp.), [47] Marvell Ltd., [48] sTec Inc. (acquired by Western Digital Corp.); for algebraic and concatenated codes, these include [38], [37], [117] (Marvell Ltd.), [118] (Qualcomm Inc.), and [119] (SK Hynix Inc.), and for constrained codes these include [120] (IBM Corp.), [121], [122] (Marvell Ltd.), [123] (Intel Corp.). Our goal in this survey is to present the mathematical concepts underpinning these trends in industry, and show how the same concepts lead to more advanced coding schemes that were recently proposed in the literature.

We advocate that the departure from channel codes previously made popular in traditional data communications and storage systems is fundamental for the future advances in NVM reliability and performance: a flourishing mathematical repertoire exists beyond the conventional coding that implicitly assumes symmetric errors. What is more, several of these techniques are also amenable to code combining in the sense that the most dominant error patterns could be first cleaned up by customized codes, followed by another perhaps more generic code for the remaining errors, which would be done in a way that is more efficient than directly applying a code that is agnostic to error patterns. As evidenced by presented examples, NVM channel-aware code design offers significant opportunities for deep theoretical explorations while simultaneously furthering the reach of memory technologies.

We presented in detail two representative classes of codes: algebraic codes and graph codes. As discussed earlier, the two approaches by design offer fundamentally different tradeoffs in terms of performance guarantee and error correction capability. Which one is ultimately chosen for deployment is a function of system-architecture considerations and the demands of the end applications – for larger page sizes, in terms of performance alone, LDPC codes are bound to be superior to BCH codes. On the other hand, well-designed code concatenation and associated algebraic codes can correct a very refined set of error patterns and can also offer backward compatibility with legacy BCH codes.

At the same time, a large body of work on coding for NVMs has still largely remained of solely theoretical interest. Moving forward, we envision that the best advancements and facilitated efforts in practice and theory alike will be achieved through a more open dialogue between industry leaders and academia. Towards that goal, in the context of different coding tools, we have outlined several (what we believe are promising) research directions. For example, best utilization of these new powerful algebraic and graph codes may require multi-page read architectures, which is in contrast to current practice of single page reads. This new approach may already be feasible as recent evidence suggests the benefits of multi-page reads when used in conjunction with simple code interleaving [69]. How to best balance read operations and coding benefits is an interesting system design problem.

We envision that several of the code design principles developed for multi-level Flash will also have a positive impact on alternative NVM technologies, including phase change memories (PCM) and resistive RAM (RRAM). These technologies also possess certain domain-specific asymmetries e.g., in PCM thermal cross-talk and thermal accumulation cause significant spatio-temporal variations in cell reliability, and in RRAM errors are data-dependent and with strong spatial correlations. New innovative coding schemes that are appropriately device-aware could play a critical role in transitioning these and other technologies into the mainstream. As the non-volatile technologies further evolve and diversify, it is not unforeseeable that different NVMs will have different types of dominant error patterns. In each case, appropriately chosen coding methods (stand alone or a combination of multiple pattern-specific methods) would yield a winning combination. Building on the fundamental coding concepts covered in this survey will help tailor the best solution to the specific design setup. Moving beyond coding for reliability lies an interesting trade-off between coding performance (that needs long blocks) and access performance (that prefers short blocks). Developing codes that operate at the desirable points of this trade-off is another fruitful avenue of future research.

Acknowledgment

The authors would like to thank A. Hareedy, F. Sala, C. Schoeny, R. Cohen, and E. Hemo for preparing and helping with some figures. Work of L. Dolecek is supported in part by NSF grants CAREER CCF 1150212 and CCF 1029030. Work of Y. Cassuto is supported in part by the Israel Science Foundation and by the Intel Center for Computing Intelligence.

References

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of IEEE*, vol. 91, no. 4, pp. 489–502, Apr. 2003.

[2] H. Wang, N. Wong, T.-Y. Chen, and R. D. Wesel, "Using dynamic allocation of write voltage to extend flash memory lifetime," *IEEE Transactions on Communications*, vol. 64, no. 11, pp. 4474–4486, Nov. 2016.

[3] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis, "Modelling of the threshold voltage distributions of sub-20nm NAND flash memory," in *IEEE Global Communications Conference (GLOBECOM)*, London, UK, December 2014.

[4] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling," in *Design Automation and Test in Europe (DATE)*, Grenoble, France, March 2013.

[5] H. Wang, T.-Y. Chen, and R. D. Wesel, "Histogram-based flash channel estimation," in *IEEE International Conference on Communications*, London, UK, June 2015.

[6] D. H. Lee and W. Sung, "Estimation of NAND flash memory threshold voltage distribution for optimum soft-decision error correction," *IEEE Transactions on Signal Processing*, vol. 61, no. 1, pp. 440–449, Jan. 2013.

[7] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Channel models for multi-level cell flash memories based on empirical error analysis," *IEEE Transactions on Communications*, vol. 64, no. 8, pp. 3169–3181, Aug. 2016.

[8] ——, "On the capacity of the beta-binomial channel model for multi-level cell flash memories," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2312–2324, Sep. 2016.

[9] L. Grupp, J. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, February 2012.

[10] "Solid state drive (SSD) requirements and endurance test method," http://www.jedec.org/standards-documents/results/jesd218b01, June 2016, accessed: 2016-11-01.

[11] A. Berman and Y. Birk, "Constrained flash memory programming," in *IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, July 2011.

[12] S. Lin and D. Costello, *Error Control Coding*. Prentice Hall, 2004.

[13] "TN-29-63: Error correction code (ECC) in Micron single-level cell (SLC) NAND," Micron Technologies, Tech. Rep., 2011.

[14] "TN-29-71: Enabling software BCH ECC on a linux platform," Micron Technologies, Tech. Rep., 2012.

[15] K. Lee, S. Lim, and J. Kim, "Low-cost, low-power and high-throughput BCH decoder for NAND flash memory," in *IEEE International Symposium on Circuits and Systems*, Seoul, Korea, August 2012.

[16] M. Zhang, F. Wu, Y. Zhou, and K. Zou, "A novel optimization algorithm for Chien search of BCH codes in NAND flash memory devices," in *2015 IEEE International Conference on Networking, Architecture and Storage*, Boston, MA, August 2015.

[17] C.-H. Yang, Y.-H. Chen, and H.-C. Chang, "An area-efficient BCH codec with echelon scheduling for NAND flash applications," in *IEEE International Conference on Communications*, Budapest, Hungary, June 2013.

[18] Y. Lee, H. Yoo, I. Yoo, and I.-C. Park, "6.4Gb/s multi-threaded BCH encoder and decoder for multi-channel SSD controllers," in *IEEE International Solid-State Circuits Conference*, San Francisco, CA, February 2012.

[19] ——, "High-throughput and low-complexity BCH decoding architecture for solid-state drives," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 5, pp. 1183–1187, May 2014.

[20] G. Forney, "Concatenated codes," Ph.D. dissertation, Massachusetts Institute of Technology, 1965.

[21] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis," in *Design Automation and Test in Europe (DATE)*, Dresden, Germany, March 2012.

[22] R. Ahlswede, H. Aydinian, and L. Khachatrian, "Unidirectional error control codes and related combinatorial problems," in *Eighth International Workshop on Algebraic and Combinatorial Coding Theory*, St. Petersburg, Russia, September 2002.

[23] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multilevel flash memories," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1582–1595, Apr. 2010.

[24] T. Klove, J. Luo, I. Naydenova, and S. Yari, "Some codes correcting asymmetric errors of limited magnitude," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7459–7472, Nov. 2011.

[25] M. Schwartz, "Quasi-cross lattice tilings with applications to flash memory," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2397–2405, Apr. 2012.

[26] L. Tallini and B. Bose, "On L1-distance error control codes," in *IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, July 2011.

[27] E. Hemo and Y. Cassuto, "A constraint scheme for correcting massive asymmetric magnitude-1 errors in multi-level NVMs," in *IEEE International Symposium on Information Theory (ISIT)*, Hong Kong, HK, June 2015.

[28] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *IEEE/ACM MICRO*, New York, NY, December 2009.

[29] E. Yaakobi, L. Grupp, P. H. Siegel, S. Swanson, and J. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *ICNC*, Maui, HI, January 2012.

[30] J. K. Wolf, "On codes derivable from the tensor product of check matrices," *IEEE Transactions on Information Theory*, vol. 11, no. 2, pp. 281–284, Apr. 1965.

[31] R. Gabrys, E. Yaakobi, and L. Dolecek, "Graded bit error correcting codes with applications to flash memory," *IEEE Transactions on Information theory*, vol. 59, no. 4, pp. 2315–2327, Apr. 2013.

[32] R. Gabrys, F. Sala, and L. Dolecek, "Coding for unreliable flash memory cells," *IEEE Communication Letters*, vol. 18, no. 9, pp. 1491–1494, Sep. 2014.

[33] C. Yang, Y. Emre, and C. Chakrabarti, "Product code schemes for error correction in MLC NAND flash memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 12, pp. 2302–2314, Dec. 2012.

[34] S. Cho, D. Kim, J. Choi, and J. Ha, "Block-wise concatenated BCH codes for NAND flash memories," *IEEE Transactions on Communications*, vol. 62, no. 4, pp. 1164–1177, Apr. 2014.

[35] S. Emmadi, K. R. Narayanan, and H. Pfister, "Half-product codes for flash memory," in *Non-Volatile Memories Workshop (NVMW)*, San Diego, CA, March 2015.

[36] D. Kim and J. Ha, "Quasi-primitive block-wise concatenated BCH codes with collaborative decoding for NAND flash memories," *IEEE Transactions on Communications*, vol. 63, no. 10, pp. 3482–3496, Oct. 2015.

[37] J. Xu, P. Chaichanavong, G. Burd, and Z. Wu, Marvell International Ltd., "Tensor product codes containing an iterative code," Patent US 7 861 131 B1, December, 2010.

[38] X. Yang, Marvell International Ltd., "Tensor product codes for flash," Patent US 8 732 543 B1, May, 2014.

[39] G. Ungerboeck, "Tellis-coded modulation with redundant signal sets, part I: introduction, and part II: state of the art," *IEEE Communications Magazine*, vol. 25, no. 2, pp. 5–21, Feb. 1987.

[40] A. Ramamoorthy, A. Wu, and P. Sutardja, Marvell International Ltd., "Method and system for error correction in flash memory," Patent US 7 844 879 B2, November, 2010.

[41] S. Li and T. Zhang, "Improving multi-level NAND flash memory storage reliability using concatenated BCH-TCM coding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 10, pp. 1412–1420, Oct. 2010.

[42] J. Oh, J. Ha, J. Moon, and G. Ungerboeck, "RS-enhanced TCM for multilevel flash memories," *IEEE Transactions on Communications*, vol. 61, no. 5, pp. 1674–1683, May 2013.

[43] T. Luo and B. Peleato, "Spreading modulation for multilevel nonvolatile memories," *IEEE Transactions on Communications*, vol. 64, no. 3, pp. 1110–1119, Mar. 2016.

[44] B. Kurkoski, "Coded modulation using lattices and Reed-Solomon codes, with applications to flash memories," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 900–908, May 2014.

[45] H. Zhong, Y. Li, R. Danilak, and E. T. Cohen, LSI Corp., "LDPC erasure decoding for flash memories," Patent US 8 935 595 B2, January, 2015.

[46] R. Motwani, Intel Corp., "Storage drive with LDPC coding," Patent US 8 549 382 B2, October, 2013.

[47] A. Ramamoorthy, Marvell International Ltd., "Multi-level signal memory with LDPC and interleaving," Patent US 7 971 130 B2, June, 2011.

[48] A. D. Weathers, R. D. Barndt, and X. Hu, sTec, Inc., "Optimal programming levels for LDPC," Patent US 8 484 519 B2, July, 2013.

[49] E. Sharon, I. Alrod, A. Navon, and O. Lieber, Sandisk Ltd., "Low density parity code (LDPC) decoding for memory with multiple log likelihood ratio (LLR) decoders," Patent US 8 301 979 B2, October, 2012.

[50] J. Wang, K. Vakilinia, T.-Y. Chen, T. A. Courtade, G. Dong, T. Zhang, H. Shankar, and R. D. Wesel, "Enhanced precision through multiple reads for LDPC decoding in flash memories," *IEEE Journal on Selected Areas of Communications*, vol. 32, no. 5, pp. 880–891, May 2014.

[51] T. Richardson, "Error floors of LDPC codes," in *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2003.

[52] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[53] B. Vasic, S. K. Chilappagari, D. V. Nguyen, and D. Declercq, "Trapping set ontology," in *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2009.

[54] S. Landner and O. Milenkovic, "Algorithmic and combinatorial analysis of trapping sets in structured LDPC codes," in *IEEE International Conference on Wireless Networks, Communications and Mobile Computing*, Honolulu, HI, June 2005.

[55] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, "Analysis of absorbing sets and fully absorbing sets for array-based LDPC codes," *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.

[56] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Design of LDPC decoders for low bit error rate performance: Quantization and algorithm choices," *IEEE Transactions on Communications*, vol. 57, no. 11, pp. 3258–3268, November 2009.

[57] A. Hareedy, C. Lanka, and L. Dolecek, "A general non-binary LDPC code optimization framework suitable for dense flash memory and magnetic storage," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2402–2415, Sep. 2016.

[58] A. Bazarsky, N. Presman, and S. Litsyn, "Design of non-binary quasi-cyclic LDPC codes by ACE optimization," in *IEEE Information Theory Workshop (ITW)*, Sevilla, Spain, September 2013.

[59] J. Wang, L. Dolecek, and R. Wesel, "The cycle consistency matrix approach to absorbing sets in separable circulant-based LDPC codes," *IEEE Transactions on Information Theory*, vol. 59, no. 4, pp. 2293 – 2314, Apr. 2013.

[60] B.Vasic, S.K.Chilappagari, D.V.Nguyen, and S.K.Planjery, "Trapping set ontology," in *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep.-Oct. 2009.

[61] L. Dolecek, "On absorbing sets of structured sparse graph codes," in *Information Theory and Applications Workshop*, San Diego, CA, February 2010.

[62] H. Imai and S. Hirakawa, "A new multilevel coding method using error-correcting codes," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 371–377, Mar. 1977.

[63] T. Richardson and R. Urbanke, *Modern Coding Theory*. Cambridge University Press, 2008.

[64] R. Cohen and Y. Cassuto, "Iterative decoding of LDPC codes over the q-ary partial erasure channel," *IEEE Transactions on Information Theory*, vol. 62, no. 5, pp. 2658–2672, May 2016.

[65] ——, "LDPC codes for the q-ary bit-measurement channel," in *9th International Symposium on Turbo Codes & Iterative Information Processing*, Brest, France, August 2016.

[66] C. Di, D. Proietti, I. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[67] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular $(2, d_c)$-LDPC codes over GF(q) using their binary images," *IEEE Transactions on Communications*, vol. 56, no. 10, pp. 1626–1635, Oct. 2008.

[68] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng, "LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives," in *USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, February 2013.

[69] S.-H. Song, K. Gunnam, M. Qin, L. Franca-Neto, R. Mateescu, C. Zhang, R. Barndt, and Z. Bandic, "High speed soft decision decoding architecture for triple level cell NAND Flash memory," in *Nonvolatile Memories Workshop*, San Diego, CA, March 2016.

[70] Y. Zhang, C. Zhang, Z. Yan, S. Chen, and H. Jiang, "A high-throughput multi-rate LDPC decoder for error correction of solid-state drives," in *IEEE Workshop on Signal Processing Systems*, Hangzhou, China, October 2015.

[71] J. Kim and W. Sung, "Rate-0.96 LDPC decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 5, pp. 1004–1015, May 2014.

[72] W. Zhao, G. Dong, H. Sun, N. Zheng, and T. Zhang, "Reducing latency overhead caused by using LDPC codes in NAND flash memory," *EURASIP Journal on Advances in Signal Processing, Special Issue on Coding and Signal Processing for Non-Volatile Memories*, vol. 203, pp. 1–9, Sep. 2012.

[73] R.-S. Liu, M.-Y. Chuang, C.-L. Yang, C.-H. Li, K.-C. Ho, and H.-P. Li, "Improving read performance of NAND flash SSDs by exploiting error locality," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1090–1102, Apr. 2016.

[74] F. Cai, X. Zhang, D. Declercq, B. Vasic, and S. K. Planjery, "Finite alphabet iterative decoders for LDPC codes: Optimization, architecture and analysis," *IEEE Transactions on Circuits and Systems - Part I: Regular Papers*, vol. 61, no. 5, pp. 1366–1375, May 2014.

[75] S. K. Planjery, D. Declercq, L. Danjean, and B. Vasic, "Finite alphabet iterative decoders, part I: Decoding beyond belief propagation on the binary symmetric channel," *IEEE Transactions on Communications*, vol. 61, no. 10, pp. 4033–4045, Oct. 2013.

[76] Y. Toriyama, B. Amiri, L. Dolecek, and D. Markovic, "Logarithmic quantization scheme for reduced hardware cost and improved error floor in non-binary LDPC decoders," in *IEEE Global Communications Conference (GLOBECOM)*, London, UK, December 2014.

[77] H. Sun, W. Zhao, M. Lv, G. Dong, N. Zheng, and T. Zhang, "Exploiting intracell bit-error characteristics to improve min-sum LDPC decoding for MLC NAND flash-based storage in mobile device," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2654–2664, Aug. 2016.

[78] M. Lentmaier, A. Sridharan, D. J. Costello Jr., and K. S. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 5274–5289, Oct. 2010.

[79] A. R. Iyengar, P. H. Siegel, R. L. Urbanke, and J. K. Wolf, "Windowed decoding of spatially coupled codes," *IEEE Transactions on Information Theory*, vol. 59, no. 4, pp. 2277–2292, Apr. 2013.

[80] B. Amiri, A. Reisizadehmobarakeh, H. Esfahanizadeh, J. Kliewer, and L. Dolecek, "Optimized design of finite-length separable circulant-based spatially-coupled codes: An absorbing set-based analysis," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4029–4043, Oct. 2016.

[81] H. Esfahanizadeh, A. Hareedy, and L. Dolecek, "Optimized graph-based codes for Flash memories," in *Flash Memory Summit*, San Jose, CA, August 2016.

[82] E. En Gad, Y. Li, J. Kliewer, M. Langberg, A. Jiang, and J. Bruck, "Asymmetric error correction and flash-memory rewriting using polar codes," *IEEE Transactions on Information Theory (submitted)*, 2014.

[83] Y. Li, H. Alhussien, E. F. Haratsch, and A. Jiang, "A study of polar codes for MLC NAND flash memories," in *IEEE International Conference on Computing, Networking and Communications*, Anaheim, CA, February 2015.

[84] R. L. Rivest and A. Shamir, "How to reuse a write-once memory," *Information and Control*, vol. 55, no. 1, pp. 1–19, Dec. 1982.

[85] D. Burshtein and A. Strugatski, "Polar write once memory codes," *IEEE Transactions on Information Theory*, vol. 59, no. 8, pp. 5088–5101, Aug. 2013.

[86] E. Yaakobi and A. Shpilka, "High sum-rate three-write and non binary WOM codes," *IEEE Transactions on Information Theory*, vol. 60, no. 11, pp. 7006–7015, Nov. 2014.

[87] A. Shpilka, "New constructions of WOM codes using the Wozencraft ensemble," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4520–4529, Jul. 2013.

[88] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Transactions on Information Theory*, vol. 58, no. 9, pp. 5985–5999, Sep. 2012.

[89] R. Gabrys and L. Dolecek, "Constructions of non-binary WOM codes for multilevel flash memories," *IEEE Transactions on Information Theory*, vol. 61, no. 4, pp. 1905–1919, Apr. 2015.

[90] A. Fiat and A. Shamir, "Generalized "write-once" memories," *IEEE Transactions on Information Theory*, vol. 30, no. 3, pp. 470–480, May 1984.

[91] F. Fu and A. J. H. Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Transactions on Information Theory*, vol. 45, no. 9, pp. 308–313, Sep. 1999.

[92] R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel flash memories," in *IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, July 2011.

[93] K. Haymaker and C. Kelley, "Geometric wom codes and coding strategies for multilevel flash memories," *Designs, Codes, and Cryptography*, vol. 70, no. 1-2, pp. 91–104, May 2012.

[94] A. Bhatia, M. Qin, A. Iyengar, B. M. Kurkoski, and P. H. Siegel, "Lattice-based WOM codes for multilevel flash memories," *IEEE Journal on Selected Areas on Communications*, vol. 32, no. 5, pp. 933–945, May 2014.

[95] A. Jiang, V. Bohossian, and J. Bruck, "Rewriting codes for joint information storage in flash memories," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 5300–5313, Oct. 2010.

[96] E. Yaakobi, A. Vardy, P. H. Siegel, and J. K. Wolf, "Multidimensional flash codes," in *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2008.

[97] H. Finucane, Z. Liu, and M. Mitzenmaher, "Designing floating codes for expected performance," in *Allerton Conference on Communication, Control, and Computing*, Monticello, IL, October 2008.

[98] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Trajectory codes for flash memory," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4530–4541, Jul. 2013.

[99] G. Mappouras, A. Vahid, A. R. Calderbank, and D. J. Sorin, "Methuselah Flash: Rewriting codes for extra long storage lifetime," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, Toulouse, France, June 2016.

[100] S. Odeh and Y. Cassuto, "NAND flash architectures reducing write amplification through multi-write codes," in *IEEE International Conference on Massive Storage Systems and Technology (MSST)*, Santa Clara, CA, June 2014.

[101] G. Yadgar, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving SSD erase costs using WOM codes." in *USENIX Conference on File and Storage Technologies (FAST)*, Santa Clara, CA, February 2015.

[102] A. Jiang, Y. Li, E. E. Gad, M. Langberg, and J. Bruck, "Joint rewriting and error correction in write-once memories," in *IEEE International Symposium on Information Theory (ISIT)*, Istanbul, Turkey, July 2013.

[103] Y. Cassuto and E. Yaakobi, "Short q-ary fixed-rate WOM codes for guaranteed rewrites and with hot/cold write differentiation," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3942–3958, Jul. 2014.

[104] C. Heegard, "On the capacity of permanent memory," *IEEE Transactions on Information Theory*, vol. 31, no. 1, pp. 34–42, Jan. 1985.

[105] E. Hemo and Y. Cassuto, "d-imbalance WOM codes for reduced intercell interference in multi-level NVMs," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 9, pp. 2378–2390, Sep. 2016.

[106] S. Cho and H. Lee, "Flip-N-write: a simple deterministic technique to improve PRAM write performance, energy and endurance," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, Dec. 2009.

[107] J. D. Lee, S. H. Hur, and J. D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Letters*, vol. 23, no. 5, pp. 264–266, May 2002.

[108] B. H. Marcus, R. M. Roth, and P. H. Siegel, *Constrained Systems and Coding for Recording Channels,Handbook of Coding Theory, V.S. Pless and W.C. Huffman (Editors)*. Elsevier, Amsterdam, 1998.

[109] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 836–846, May 2014.

[110] S. Buzaglo, E. Yaakobi, and P. H. Siegel, "Coding schemes for intercell interference in flash memory," in *IEEE International Symposium on Information Theory (ISIT)*, Hong Kong, HK, June 2015.

[111] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Error analysis and intercell interference mitigation in multi-level cell flash memories," in *IEEE International Conference on Communications (ICC)*, London, UK, June 2015.

[112] Y. Kim, R. Mateescu, S.-H. Song, Z. Bandic, and B. V. K. V. Kumar, "Coding scheme for 3D vertical flash memory," in *IEEE International Conference on Communications (ICC)*, London, UK, June 2015.

[113] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[114] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3158–3165, Jul. 2010.

[115] A. Mazumdar, A. Barg, and G. Zemor, "Constructions of rank modulation codes," *IEEE Transactions on Information Theory*, vol. 59, no. 2, pp. 1018–1029, Feb. 2013.

[116] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck, "Constant-weight Gray codes for local rank modulation," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7431–7442, Nov. 2011.

[117] N. Varnica, G. Burd, S. Low, L. Sun, and Z. Wu, Marvell World Trade Ltd., "Concatenated codes for holographic storage," Patent US 8 583 981 B2, November, 2013.

[118] J. K. Wolf, Qualcomm Incorporated, "Method and apparatus for transmitting and receiving concatenated code data," Patent US 5 983 383, November, 1999.

[119] M. Marrow, SK Hynix Memory Solutions Inc., "Coding architecture for multi-level NAND flash memory with stuck cells," Patent US 8 719 670 B1, May, 2014.

[120] M. M. Franceschini, A. Jagmohan, L. A. Lastras-Montaño, and M. Sharma, International Business Machines Corporation, "Constrained coding to reduce floating gate coupling in non-volatile memories," Patent US 8 463 985 B2, June, 2013.

[121] P. Sutardja and Z. Wu, Marvell World Trade Ltd, "Flash memory with coding and signal processing," Patent WO 2 007 084 751 A2, July, 2007.

[122] P. Chaichanavong, Marvell International Ltd., "Systems and methods for constructing high-rate constrained codes," Patent US 7 714 748 B1, May, 2010.

[123] R. Motwani, Intel Corp., "Maximum-likelihood decoder in a memory controller for synchronization," Patent WO 2 013 048 385 A1, April, 2013.