Writing Cosets of a Convolutional Code to Increase the Lifetime of Flash Memory

Adam N. Jacobvitz, R. Calderbank, Daniel J. Sorin
Department of Electrical and Computer Engineering
Duke University, Durham, North Carolina, USA, 27708
adam.jacobvitz@duke.edu, robert.calderbank@duke.edu, sorin@ee.duke.edu

***Abstract*—The goal of this paper is to extend the lifetime of Flash memory by reducing the frequency with which a given page of memory is erased. This is accomplished by increasing the number of writes that are possible before erasure is necessary. Redundancy is introduced into the write process to decrease the number of memory cells that are impacted by a given write, and to even out the impact of writing across an entire page of memory. Improvements are expressed in terms of write efficiency and lifetime gain. Write efficiency is the ratio of cells written to cells available, and lifetime gain is the ratio of coded writes to the baseline of uncoded writing.**

We use a physical model that allows multiple writes to a given region of memory. This can be realized with single level cells or with multi-level cells. Data is written to memory in the form of a coset of a convolutional code. The coset is represented by a binary vector that is selected by the Viterbi algorithm to minimize the number of cells impacted by the write (Hamming weight) and to even out the number of writes to each cell within a given page. Several different Viterbi metrics are evaluated. It is shown that page write efficiencies of over 85% and lifetime gains of over 500% are possible with only modest encoding and decoding complexity.

It is also straightforward to integrate lifetime extension with standard methods of error correction by requiring that the coset representative be drawn from an error correcting code. An example is provided where single error correction is provided using a Hamming code.

I. Introduction and Background

Current computing systems usually include hard disk drives (HDDs). However HDDs are losing ground because they do not support random access, their moving parts consume significant power (3-8W) [1], and they have relatively slow read and write speeds. HDDs are also susceptible to early failure [2], requiring an expensive burn-in process to avoid significant return costs. Furthermore, prices are falling for alternative storage devices, encouraging the transition from HDDs.

One alternative to HDDs are Flash Electronically Erasable Programmable Read Only Memory (EEPROM) Solid State Drives (SSDs). Flash memory was patented by Fujio Masuoka at Toshiba in 1980 [3]. The first patent on Flash SSDs was released in 1989 [4] and had support for handling defective cells, a problem at the time, and still a problem today. In 1991 Sandisk developed and sold one of the first Flash SSDs: a 20MB model for $1000. In 1987, Masouka developed a new variation of Flash known as *NAND Flash*. NAND Flash can be manufactured much more densely than prior Flash technologies. The vast majority of Flash SSDs today use NAND Flash. When we refer to Flash memory in this paper, we will be referring only to NAND Flash. NAND Flash and the development of multi-level cells (MLCs), which allow multiple states to be stored in a single Flash cell[1], caused prices to fall dramatically. Further advances in miniaturization and reliable manufacturing have resulted in the SSDs we see today.

Only recently have SSDs achieved the data densities required to replace HDDs. There is still a difference in density, but as transistor minimum feature size shrinks the gap is closing. SSDs are becoming more attractive despite being more expensive than HDDs. In 2012 prices, HDDs cost around $0.06/GB [5] for consumer grade, and around $0.15/GB [6] for enterprise grade. SSDs cost around $1.00/GB [7] for consumer grade, and $5.50/GB [8] for enterprise grade. Therefore a typical SSD in 2012 costs around 15x - 40x per GB more than an equivalently sized HDD.

The adoption of SSDs by large data centers [9] has made SSDs' shortcomings more visible. Our primary focus is on *wear out*, the fact that an SSD can only support a limited number of writes before cells fail. On average, Flash cells support around 10^4 *erases* (1-to-0 transitions) before failing. This number is projected to decrease as the minimum feature size of Flash transistors shrinks [10]. This shortcoming is magnified by patterns of writing data that concentrate writes on a small number of cells in a given device [11].

We introduce two metrics to evaluate the performance of codes that extend Flash SSD lifetime by introducing redundancy into the write process. Both metrics are with respect to the minimum addressable unit of a Flash SSD often referred to as a *page*. Typically a page is composed of 2KB, 4KB, 8KB of Flash cells or more. The page is the smallest region of memory that can be read or written to at a given time. The first of the two metrics is lifetime

[1]Single Level Cells (SLCs) are still used in enterprise applications due to their longer lifetimes.

gain, which measures how many additional writes to a page are possible compared to writing without a code. This is our primary metric, as our goal is to be able to write many times before erasing. Our secondary metric is write efficiency. Write efficiency indicates when more writes might be possible. It is not a perfect metric because it would report 100% write efficiency even if we were to exhaust a page by writing to every cell. We refer to codes designed to optimize these metrics as *endurance codes*.

$$\text{Lifetime Gain} \triangleq \left(\frac{\text{\# of Writes w/ Code Before Erase is Req}}{\text{\# of Writes w/o Code Before Erase is Req}} - 1 \right) \times 100\%$$

$$\text{Write Efficiency} \triangleq \left(\frac{\text{\# of Writes to all Cells Before Erase is Req}}{\text{Max \# of Writes Possible to all Cells}} \right) \leq 1$$

We design endurance codes that make use of the Program Without Erase (PWE) write mechanism, a technique that enables incremental programming of Flash cells without first requiring an erase [12]. In practice we must be able to increase the charge in MLC cells accurately, and this is complicated by issues such as *overshoot*, [13] where the programmed charge exceeds the given target level, and *Program and Read Disturbs* (PRD), where reading/writing proximate cells causes changes in threshold voltage levels [14]. Both overshoot and PRD may compromise direct use of a technique like waterfall coding [15] with PWE. In order to provide an interface for PWE endurance codes it is necessary to combine waterfall coding with improvements to programming accuracy [16] and coding techniques that reduce the tolerances required for MLC programming [17][18].

Not only is Flash memory prone to wear out, Flash memory must also be resilient to manufacturing defects and to transient errors. This includes those due to alpha particle strikes [19], neutron particle strikes [20], and finally what are known as *retention errors* [21] where charge slowly leaks out of the Flash cell causing it to change value. Previous work has shown that, on average, the number of bits with incorrect values present at a given time in the memory can be as high as 1 out of 10^6 due to retention errors for 63nm Flash cells [21] and 2 out of 10^8 due to alpha particle strikes on 25nm Flash cells [19]. These studies guide our choice of ECC parameters when choosing examples to demonstrate the integration of standard ECC with coding to extend the lifetime of Flash memory.

In this paper we will primarily use binary convolutional codes to construct endurance codes. Data is then written to memory as a coset representative, and this representative is chosen to minimize the number of writes and to even out wear across the memory (see [22][23] for background material on coset coding). Integration with standard ECC is accomplished by requiring that the representative be chosen from a binary Hamming code. We achieve write efficiencies of 53% and lifetime extension of 300% for SLCs, and write efficiencies of over 95% and lifetime extension of over 700% for MLCs.

Sections II and III provide a brief description of the physics of Flash memory and the system model. Section IV compares the performance of our endurance codes with prior work on Write Once Memory (WOM), floating codes and enumerative codes. Coset coding with block and convolutional codes are described in Sections VI and VII respectively. The integration of coset coding with standard error correction is described in Section V. A technique for gaining a few additional writes for very low overhead is introduced in Section VIII. The modification of the Viterbi algorithm to promote wearleveling is described in Section IX.

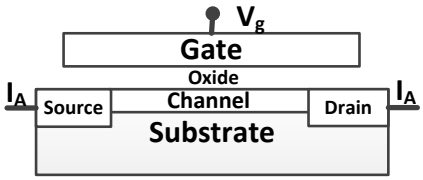
II. The Physics of Flash Memory

In traditional circuit design, a planar MOSFET[2] (Metal Oxide Semiconductor Field Effect Transistor) has a single gate. When a voltage larger than the threshold voltage (V_{th}) is applied to the gate, the transistor turns on. When a voltage below V_{th} is applied, the transistor is off. A Flash MOSFET has two gates: a *floating gate* and a *control gate*. When charge is present between the floating and control gates, a voltage above V_{th} applied to the control gate will turn on the transistor. If no charge is present between the two gates, the same voltage applied to the control gate will not turn on the transistor. Figure 1 abstracts the difference in structure between standard and Flash MOSFETs.

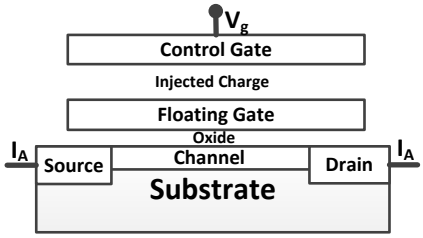
Devices store state in Flash memory through two operations: PROGRAM and ERASE. Programming is the act of injecting charge between a cell's control and floating gate through either Hot-Electron Injection (HEI) or Fowler-Nordheim (FN) tunneling [24]. Erasure of a block requires removing the charge from the gate, typically through Fowler - Nordheim tunneling. HEI uses strong lateral and transversal electric fields to inject carriers through the oxide quickly and into the gap between the floating and control gates. FN tunneling uses a strong electric field across a thin oxide to induce quantum tunneling. For more information on the physics of Flash MOSFETs as well as a detailed model, we refer the reader to [24].

In Flash memory cells it is possible to create multiple ON states by varying the amount of charge between the floating and control gate. Single-Level Cells (SLCs) contain two states. Cells which are programmed to contain more than two states are known as Multi-Level Cells (MLCs). Figure 2 shows how the output current of a multi-level cell depends

[2]These transistors are still used in logic and classical memory configurations such as SRAM and DRAM.



(a) Traditional Planar MOSFET



(b) Floating Gate MOSFET used in Flash Memory

Figure 1: Traditional Planar and Flash MOSFETs

on the control gate voltage (V_g) and on the level of charge between the floating and control gates. It is customary to fabricate MLCs to store multiple bits in a single Flash cell, but it is not strictly necessary that the number of states be a power of 2. The most important property of MLCs is that it is possible to increment their state without first requiring an erase by increasing the charge between the floating and control gates. This capability is called *program without erase* [12] and it is the key to increasing the lifetime of NAND Flash memory.

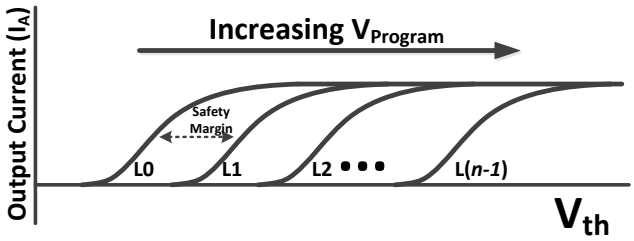


Figure 2: Output current (I_A) as a function of voltage level Li, i = 0, 1, ... , n-1 for an n-level cell. The safety margin assures that one level is not read as another.

III. System Model

Flash SSDs are composed primarily of two components: the NAND Flash memory chips in which data is stored, and a controller called the Flash Translation Layer (FTL). A component on the host computer known as the *Host Bus Adapter* (HBA) uses a protocol (usually SCSI, Infiniband, Fibre Channel or SATA/PATA based) to communicate with the SSD.

The Flash Translation Layer (FTL) shown in Figure 3 is the brain of the SSD. The FTL translates memory read/write requests into the sequence of operations that are necessary to change the state of the Flash memory. The FTL reads and writes SLC or MLC Flash memory cells at granularity of a page. Flash memory is erased at the more coarser granularity of a *block*, typically 256KB or more.

Erasures stress Flash cells, with Flash transistors expected to fail after 10^4 erasures [10]. This number is projected to decrease as the minimum feature size of flash transistors shrinks. Therefore it is important to increase the number of writes before an erasure is necessary and to even out wear across the page.

Note also that without error correction, a page will fail when a single cell fails within that page. The industry standard for the number of cells with erroneous data in a HDD is 1 in 10^{13} - 10^{15} [25]. For SSDs to be competitive with HDDs they must meet this standard. Setting aside manufacturing defects, the number of cells with erroneous data for SSDs can be as high as 1 in 10^6 (see Section I). Some level of error correction is necessary if SSDs are to match the data integrity of HDDs.

Our systems objective then is to delay the wear out of Flash cells by reducing the frequency with which blocks are erased while still tolerating errors at a level required to match that of HDDs.

IV. Previous Work

We now discuss prior work on endurance codes and on error correction for SSDs.

A. Endurance Coding for SSDs

The Write Once Memory (WOM) model exhibits the same behavior as implementing PWE in a multi-level cell. The value of WOM coding in terms of extending the number of writes (before failure in their paper, before erase in ours) is explored in the original paper by Rivest and Shamir [26] and in subsequent work [27][28]. WOM codes can be applied to multiple alphabet sizes, but we will assume an alphabet of $\{0, 1\}$ for our input. We will also assume writing to q-level MLCs (q-ary cells).

Flash memory can be presented either as a Write Once Memory (WOM) or Write Asymmetric Memory (WAM). We define a WOM (and/or WAM) code by defining an encoding function from k data bits $d = (d_1, ..., d_k)$ to n code bits $c = (c_1, ..., c_n)$ and a decoding function that recovers the data bits d from the codeword c. Given the exact properties

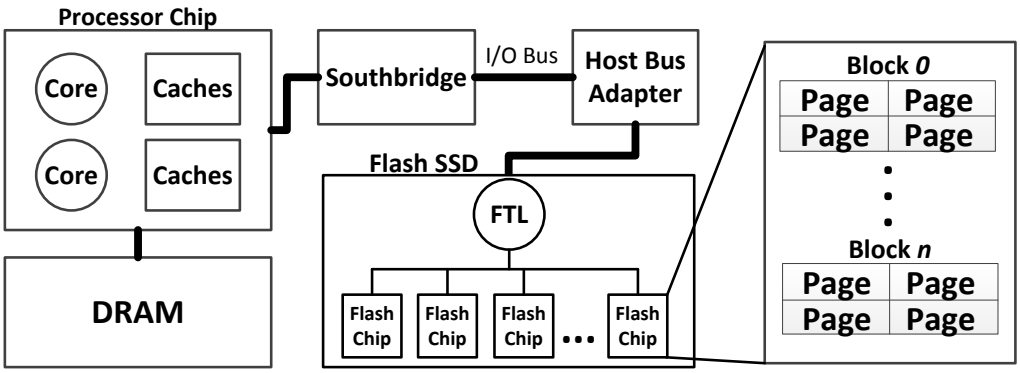


Figure 3: High level architecture showing integration of computer processor and Flash SSD

of the underlying storage, WOM/WAM codes can be created directly [27][28] or using enumerative coding [29]. There are also frameworks for implementing WOM/WAM codes in a systematic manner such as by using Floating Codes [30].

The codewords in a floating code are organized in a multi-level directed graph where updating a single input variable d_i causes the memory state to change from one level to the next. The number of levels in the graph is more than the number of states in an individual cell, and this translates to extending the life of the memory. For example, given a 3-state MLC, it is possible with 3 cells to guarantee that 2 data bits can be rewritten 5 times (see [30] for details). With no coding, data is written directly to memory ($k = n$) and only 3 rewrites are possible if the same bit is updated three times. In this example, the price of resilience to updates that are focused on a single input bit is 50% area overhead. The initial focus on worst case performance of floating codes has been expanded to average case performance [31] by viewing rewriting as a Markov process on the multilevel graph. One of the advantages of the coset coding approach described in Section V is that information carried by an individual input bit is spread across a larger number of memory cells. It is immaterial whether updates are focused on a small number of bits or distributed more evenly.

Enumerative coding is a generalized technique for mapping between a set of vectors and their relative indexes when lexographically enumerated. Jagmohan et. al. [32] uses enumerative coding to guarantee two writes to SLC cells before erasing is required. Jagmohan et al. does this by representing the input data as a lexographic index of a set of vectors V. The vectors in V all have the same symbol frequency distribution. Vectors are then selected from V to be written to memory. A second write is accomplished by indexing the feasible vectors that remain. Note however that the rate of the second write depends on what was written initially. Decoding consists of determining the lexographic order of the written vector which is then read out as the data.

B. Error Correction for SSDs

Commercial SSDs require Flash memory that can accommodate large numbers of writes and withstand a mixture of permanent faults and transient errors. Alpha particle and neutron strikes may cause the threshold voltage in an MLC to shift which may in turn cause the state to change to a higher value. Conversely the state of an MLC might change to a lower value because a gate fails to retain charge.

Error Correcting Codes (ECC) are the most common means of recovery from these types of error, but there has also been prior work that takes advantage of the unique characteristics of Flash memory. One such method is to detect invalid threshold values by expanding the buffers between different threshold voltages [33]. However this method reduces the number of states that can be supported by a single MLC cell, creating additional area overhead. Another technique breaks up input data into sub-blocks, encodes each sub-block with ECC, and then uses coset coding to write the data to disk [34]. There has also been work on asymmetric error correction that leverages the fact that cell defects during manufacture usually have a known value even though the cell cannot be rewritten [35][36][37][38].

V. Integration of Coset Coding and Error Correction

This section describes how coset coding [22][23][39] may be integrated with standard error correction. We provide complete details for a simple illustrative example, the single error correcting Hamming code, and leave it to the reader to extend the method to an arbitrary linear block code. We employ an $[n,k]$ binary linear code C for error correction and an $[n,m]$ subcode C' for coset coding. Input data is encoded as a coset of C' in C and the coset representative is chosen to reduce the number of bits written to memory and

to promote wearleveling. By varying the objective function we can tradeoff lifetime gain and write efficiency within the same optimization framework. The area overhead is the ratio of the number of input data bits to the length of the code minus one, that is $\frac{n}{k-m} - 1$.

A. Encoding

When uncoded data is written directly to Flash memory cells, uneven wear can significantly reduce the lifetime of the SSD [11]. Wear can be reduced by reading the state of the memory and using this information to reduce the number of writes. For example we might choose between a data word and its complement as in Flip-N-Write [40]. As an illustrative example consider the integration of coset coding using the repetition code with error correction using the [8,4] Hamming code C. Here $n = 8$, $k = 4$, $m = 1$ and the area overhead is $5/3 = 167\%$.

The first row ($m = 1$) of the generator matrix G given below generates the repetition code C' and the next three rows ($k - m = 3$) generate distinct coset representatives for C' in C. Three input bits generate a coset of C' in C and the representative is chosen to minimize Hamming weight.

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ - & - & - & - & - & - & - & - \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{B} \\ - \\ \mathbf{D} \end{bmatrix}$$

In general we may select a $k \times n$ generator matrix $\mathbf{G}$ for C where the first m rows form a matrix $\mathbf{B}$ that generates the coset code C' and the remaining $k - m$ rows form a matrix $\mathbf{D}$ that generates representatives for 2^{k-m} distinct cosets of C' in C. If d is the dataword, then we select the coset representative $d \cdot \mathbf{D} + b \cdot \mathbf{B}$ that minimizes our objective function. The input vector b is used to balance lifetime gain and write efficiency.

At small block lengths, the area overhead incurred by error correction and coset coding is prohibitive. This suggests making block length and page length commensurate, but if the dimension of the coset code C' scales linearly with the block length n then the number of potential coset representatives scales exponentially with n and exhaustive search is infeasible. Since the search metric is non-negative and computed entry by entry it is natural to use the Viterbi algorithm to select the coset representative. The role of the coset code is then to provide an ensemble of vectors that can be searched efficiently by this algorithm. We have chosen to implement coset coding primarily using convolutional codes, though we note that many traditional block codes have low complexity trellis representations.

We generate an initial coset representative $d \cdot \mathbf{D}$ and use the Viterbi algorithm to find an approximation $b \cdot \mathbf{B}$ to $d \cdot \mathbf{D}$. If wearleveling were not a concern this would simply reduce to data compression with respect to the Hamming metric. It is the error vector $e = d \cdot \mathbf{D} \oplus b \cdot \mathbf{B}$ that is then written to memory. If the current state of the memory is S then we need to write the coset $S \oplus d \cdot \mathbf{D}$ and we approximate this vector rather than $d \cdot \mathbf{D}$.

B. Decoding

We encode a dataword d as

$$c = d \cdot \mathbf{D} \oplus b \cdot \mathbf{B} = [d|b]\mathbf{G}$$

where $\mathbf{G}$ is the generator matrix of the error correcting code C. We decode c by forming $c \cdot \mathbf{G^*}$ where $\mathbf{G^*}$ is the right inverse of $\mathbf{G}$ (that is $\mathbf{GG^* = I}k$). One method of calculating $\mathbf{G^*}$ is to invert the Smith or Hermite Normal Form of $\mathbf{G}$.

VI. Block Codes

This section explores the use of block codes for both error correction and coset coding. We begin by calculating the Bit Flip Gain of the repetition code R_L of length L in the absence of error correction, when the input is a random binary vector of length L.

$$\text{Bit Flip Gain} \triangleq \left(\frac{\text{\# of Bits Flipped Writing a Dataword}}{\text{\# of Bits Flipped Writing a Codeword}} \right)$$

Theorem 1: When L is even, the Bit Flip Gain of the repetition code R_L of length L is given by

$$\frac{L/2}{\frac{L}{4(2^{L-1})}\left(2^L - \binom{L}{L/2}\right)} \tag{1}$$

and when L is odd, the Bit Flip Gain is given by

$$\frac{L/2}{\frac{L}{2^L}\left(2^{L-1} - \binom{L-1}{(L-1)/2}\right)} \tag{2}$$

Proof: When L is even, the expected number of bit flips E_L is given by

$$\begin{aligned} E_L &= \frac{L + \sum_{j=2}^{L/2} j\binom{L}{j} - \binom{L}{L/2}\frac{L}{4}}{2^{L-1}} \\ &= \frac{L}{2^{L-1}}\left(1 + \sum_{j=2}^{L/2} \frac{(L-1)!}{(j-1)!(L-j)!} - \frac{1}{4}\binom{L}{L/2}\right) \\ &= \frac{L}{4(2^{L-1})}\left(2^L - \binom{L}{L/2}\right) \end{aligned}$$

The Bit Flip Gain is the ratio of $\frac{L}{2}$ to E_L.

When L is odd the expected number of bit flips E_L is given by

$$E_L = \frac{1}{2^L}\sum_{j=0}^{L} j\binom{L}{j} = \frac{2}{2^L}\sum_{j=1}^{(L-1)/2} j\binom{L}{j}$$

Applying the identity $k\binom{n}{k} = n\binom{n-1}{k-1}$ we obtain

$$E_L = \frac{L}{2^L}\left(2^{L-1} - \binom{L-1}{(L-1)/2}\right)$$

The Bit Flip Gain is the ratio of $\frac{L}{2}$ to E_L. ■

Remark If L is even then

$$\text{Bit Flip Gain} = \frac{L/2}{\frac{L}{2^{L+1}}\left(2^L - \binom{L}{L/2}\right)} = \frac{(L+1)/2}{\frac{L+1}{2^{L+1}}\left(2^L - \binom{L}{L/2}\right)}$$

The Bit Flip Gains for L and L+1 coincide.

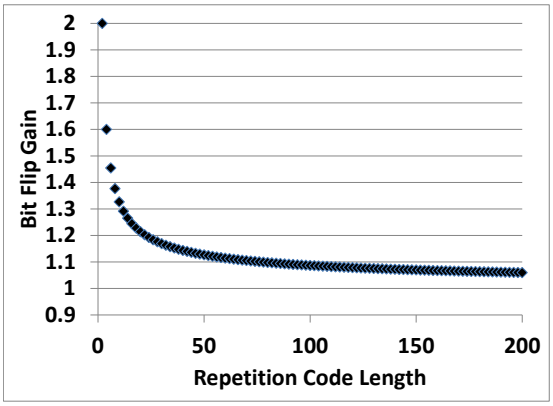


Figure 4: Bit flip gains for Flip-N-Write as a function of the block length L. The area overhead is $L/(L-1)$. Shorter codes are less efficient but provide larger bit flip gains.

Figure 4 shows that the Bit Flip Gain $\left(\frac{L/2}{E_L}\right)$ decreases with the block length L. This is unsurprising since the most likely input vector weight is $L/2$, and therefore the longer the input vector the less likely it is that there will be bit flip gains.

Next we combine error correction, using an extended Hamming code of length mL, with a coset code formed by concatenating m Repetition codes of length L. We permute the entries of the extended Hamming code so that it contains every codeword in the coset code. A generator matrix for the case $m = 2$, $L = 8$ is shown below; the first two rows form a matrix **B** that generates the coset code C' and the remaining 9 rows form a matrix **D** that generates representatives for 512 distinct cosets of C' in C.

$$\mathbf{K} = \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1\\ -&-&-&-&-&-&-&-\\ 1&1&1&1&1&1&1&1\\ 0&1&1&1&1&0&1&1\\ 0&1&1&0&1&0&0&1\\ 1&0&0&0&0&1&1&1\\ 0&1&0&1&0&0&1&1\\ 1&1&1&0&0&0&0&1\\ 0&1&1&1&0&0&0&1\\ 1&1&1&0&0&1&0&0\\ 0&0&0&1&0&0&1&0 \end{bmatrix}$$

$$\mathbf{L} = \begin{bmatrix} 1&1&1&1&1&1&1&1\\ 0&0&0&0&0&0&0&0\\ -&-&-&-&-&-&-&-\\ 1&1&0&1&1&0&0&0\\ 1&1&1&1&0&0&0&0\\ 1&0&0&1&0&1&1&0\\ 1&1&1&0&1&1&1&0\\ 1&0&1&0&1&1&0&0\\ 0&0&1&1&1&0&0&1\\ 0&1&1&1&0&0&0&1\\ 1&0&0&1&0&0&0&0\\ 0&0&1&1&0&1&0&1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix}\mathbf{K} & \mathbf{L}\end{bmatrix} = \begin{bmatrix}\mathbf{B}\\ --\\ \mathbf{D}\end{bmatrix}$$

The area overhead for this code is $16/9-1 = 77\%$, the bit flip gain is 1.38 (calculated using Theorem 1). When $m = 2i$ we have measured the bit flip gains of coset codes C' contained in the extended Hamming code of length mL that are the product of m r-dimensional subcodes C'' of the Hamming code of length L. Flip-N-Write corresponds to the special case of the Repetition code where $r = 1$.

The bit flip gains shown in Table I were obtained by numerical simulation. Larger values of r and smaller values of L provide higher Bit Flip Gains at the cost of more significant area overhead.

Bit flip gains are only a means to an end and we extend memory lifetime not only by writing fewer bits but by distributing those writes evenly over the memory cells. Section IX describes methods for selecting coset representatives that promote wearleveling. These methods provided only modest lifetime extension when applied to Flip-N-Write. For example, assuming 8-level MLC cells, it was possible to use a Repetition code of length $L = 8$ to write to a 4KB page 8 times rather than 7. In this example, the write efficiency is relatively high (80% after 7 writes and 92% after 8 writes). It

dim(C')= r	Block Length L	ECC Block Length mL	Bit Flip Gain	Area Overhead (%)
3	32	64	1.07	25.49
3	32	128	1.14	18.52
3	32	256	1.17	14.80
3	32	512	1.19	12.78
3	64	128	1.09	12.28
3	64	256	1.12	8.94
3	64	512	1.14	7.11
3	64	1024	1.15	6.11
3	128	256	1.08	6.22
3	128	512	1.10	4.49
3	128	1024	1.11	3.54
3	128	2048	1.11	3.01
4	8	16	0.43	433.33
4	8	32	0.91	220
4	8	64	1.14	156
4	8	128	1.27	128.57
4	16	32	0.99	77.78
4	16	64	1.13	56.10
4	16	128	1.21	45.45
4	16	256	1.26	39.89

Table I: Numerical evaluation of bit flip gains provided by coset codes that are m-fold direct products of r-dimensional seeds.

is the combination of small overhead and small bit flip gains that limits performance. The search for long block codes with significant bit flip gains is a subject for future study.

VII. Convolutional Codes

This section presents results showing that very simple convolutional codes are remarkably effective as coset codes. Consider the problem of lossy compression of equiprobable binary data using the rate 1/2 convolutional code with 2 states that appears in Table II. It is possible to analyze the Viterbi algorithm via a Markov process on three decoder states and to show that on average it is only necessary to change one bit in six to convert a random binary vector to a codeword.

Constraint Length	Bit Flip Gain	Area Overhead
1	1.49	100%
2	1.79	100%
3	1.85	100%
4	1.90	100%
5	1.94	100%
6	1.97	100%
7	2.03	100%
8	2.04	100%

Table II: Bit flip gains associated with coset codes that are rate 1/2 convolutional codes. The outer error correcting code is an extended Hamming code. The generator polynomials are are taken from Table 12.1 (c) of [41]. The convolutional codes are allowed to start and terminate at any state.

Constraint Length	Bit Flip Gain	Area Overhead
1	1.31	33%
2	1.40	33%
3	1.51	33%
4	1.57	33%
5	1.60	33%
6	1.62	33%
7	1.63	33%
8	1.64	33%

Table III: Bit flip gains associated with coset codes that are rate 1/4 convolutional codes. The outer error correcting code is an extended Hamming code. The generator polynomials are are taken from Table 12.1 (c) of [41]. The convolutional codes are allowed to start and terminate at any state.

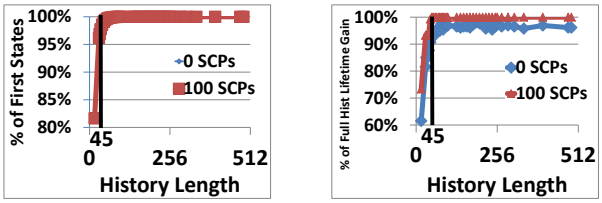
The cost of writing N data bits as a codeword of length $2N$ is then $N/3$, the cost of writing N uncoded bits is $N/2$, so the bit flip gain is $3/2$. This is slightly different from the value reported in Table II because the input to the coset code is a random Hamming codeword and not a random vector. Tables II and III show that the bit flip gains provided by convolutional codes are significantly higher than those provided by Repetition codes. The Viterbi algorithm is used to select the coset representative. Bit flip gains increase with constraint length and significant gains are possible with modest complexity (16 trellis states).

Note however that gains are considerably more modest when the decoding window for the convolutional code is shorter than the full codeword length. The advantage of using convolutional codes is that it is possible to optimize pattern matching by delaying decoding decisions. Table IV lists bit flips gains for convolutional codes using a sliding window from size 16 to the full codeword length. These gains are calculated by numerical simulation.

	Bit Flip Gains		
History Depth	8-State Code	512-State Code	Area Overhead
16	1.55	1	100%
32	1.80	1.11	100%
64	1.84	1.37	100%
128	1.84	1.78	100%
256	1.84	2.05	100%
512	1.84	2.1	100%

Table IV: Bit flip gain as a function of decoding depth for coset codes that are rate 1/2 convolutional codes appearing in Table II

In the application of convolutional codes to digital communication the input sequence is a codeword perturbed by noise rather than a random vector, so quantization is less of a challenge. Experience and analysis have shown that if



(a) First State Convergence

(b) % of Full Lifetime Gain

Figure 5: The Effects of using a Sliding Window on Lifetime Gain for a Rate 1/2 8-State Code, Codeword Length 1024-bits

the number of stages t in the decoder window is on the order Fof 5 times the constraint length then with probability approaching 1 all survivors stem from the same information block t stages back (see [41] for more details). When random vector inputs are used with our modified Viterbi algorithm, reduction to a single survivor is slower. Figure 5 shows that convergence for a code of constraint length 3 occurs only after approximately 45 stages.

VIII. Saturated Cell Pointers

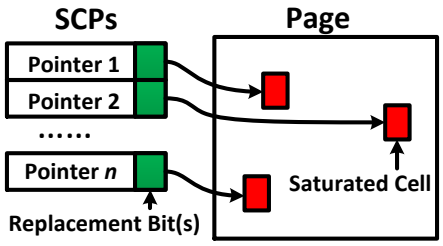


Figure 6: An Illustration of Saturated Cell Pointers

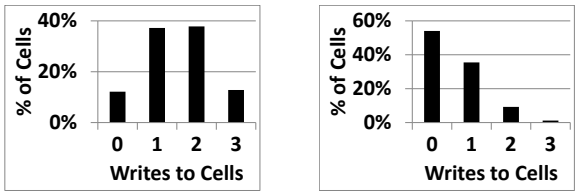
Saturated Cell Pointers (SCPs) are resources that can be combined with coset coding to extend the lifetime of memory. A SCP points at a saturated cell and delays the need to erase the page by providing one or more replacement bits as shown in Figure 6. When the page is erased the pointer is also erased. Table V relates the number of writes to the number of SCPs. As the SCPs become active the quantization problem approaches that of finding a convolutional codeword that matches the SCP entries (see [42]. These entries constitute a fraction of the total entries and there is a threshold above which there are diminishing returns.

IX. Wearleveling

Our systems objective is not bit flip reduction, rather it is the maximize the number of writes to a page before erasure is necessary. We found that when the Viterbi algorithm simply selects the coset representative by minimizing the Hamming distortion the result is uneven wear and premature saturation of the page. We illustrate this in Figure 7 which shows the distribution of cell states immediately before an erase for both uncoded recording (Case (a)) and coset coding without wearleveling (Case (b)). In both cases the write efficiency is far from ideal. We therefore modify the edge metric in the Viterbi algorithm to promote wearleveling.

# of SCPs	# of Writes SLC	4-MLC	8-MLC	Area Overhead
0	3	15-16	45	0%
1	3	16	45	0%
10	3	16	46	0.5%
20	3	16-17	46	1%
50	4	17	47	2.4%
100	4	18	48	4.9%
200	4	18	49	10%
500	4	19	49	24%
1000	5	19	50	49%

Table V: Number of Writes Before Erasure for Different Numbers of SCPs When Writing to a 4KB Page. Area Overhead is Calculated for a Rate 1/2, 512-State Convolutional Code



(a) No Coding

(b) Coset Coding w/o Wearleveling

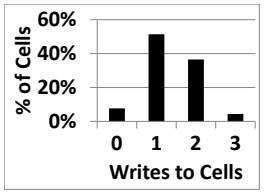
Figure 7: Terminal wear distribution across a page of 4-level MLCs. Case (a) shows the baseline of no coding which supports 3 writes before erasure. Case (b) shows coset coding without wearleveling for a convolutional code with constraint length 9. The combination of coset coding and 100 SCPs supports 5 writes before erasure.

Let $q-1$ be the number of writes that a cell can accommodate. We use the Viterbi algorithm to find a codeword in a rate $1/n$ convolutional code that best approximates the initial coset representative. The branch metric that minimizes Hamming distortion is simply the Hamming distance between the input $c = (c_1, ..., c_n)$ and the edge label $d = (d_1, ..., d_n)$.

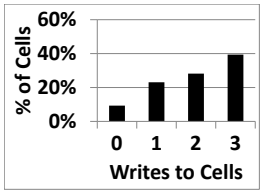
We promote wearleveling by incorporating the *Per Cell Previous Write Count* (PCPWC) into a new edge metric $M(c,d)$ given by

$$M(c,d) = \delta(c_1,d_1)W_1 + ... + \delta(c_n,d_n)W_n$$

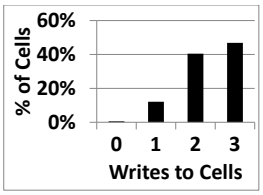
where $\delta(x,y)$ is 0 or 1 according as x are y are the same or different. If the number of prior writes to cell i, denoted here as e, is less than $q-1$ then the weight W_i is set to the number of prior writes. If $e = q-1$ then the weight is set to a large positive number (approximating ∞) to strongly discourage use of this edge.



(a) Viterbi edge metric incorporating PCPWC



(b) Viterbi edge metric forcing interpolation of saturated cell entries

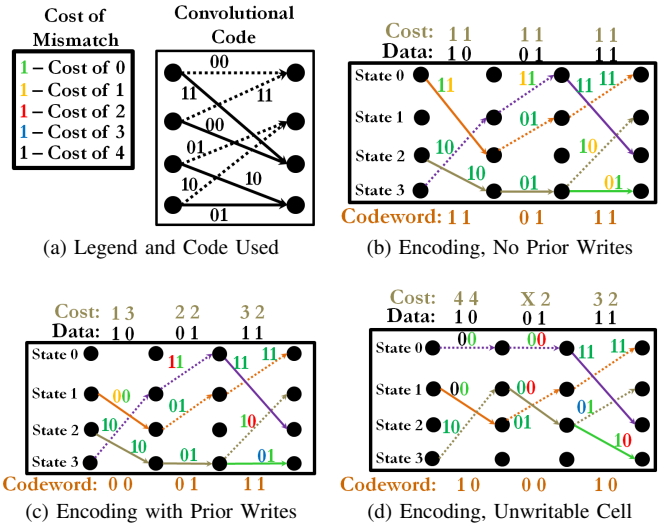


(c) Viterbi edge metric combining (a) and (b) above

Figure 8: Terminal wear distribution across a page of 4-level MLCs. Case (a) shows the effect of integrating the PCWC into the Viterbi edge metric. Case (b) shows the effect of modifying the Viterbi edge metric to force interpolation of entries in saturated cells. Case (c) shows the effect of both modifications in combination. The number of writes before erasure is 11 in Case (a), 16-17 in Case (b), and 18 in Case (c). All results are obtained for coset coding with a convolutional code with constraint length 9 in combination with 100 SCPs.

Figure 8 displays the effect of these modifications. Incorporating PCPWC into the Viterbi edge metric creates a wear distribution very similar to that of uncoded recording. Simply requiring the coset representative to match values stored at saturated cell locations also creates a very favorable wear distribution. In combination they provide a terminal wear distribution where about half the cells are fully saturated and almost every cell has been written to at least once.

Figure 9 provides examples of Viterbi path selection in three circumstances encountered during encoding. When there are no prior writes (Case (b)) the best path corresponds to the codeword that is closest in Hamming distance to the input string. With prior writes (Case (c)) the number of writes to a given cell enters the Viterbi edge metric. With saturated cells (Case (d)) the Viterbi edge metric is modified further to force survivor paths to match the entries in these cells.



(a) Legend and Code Used

(b) Encoding, No Prior Writes

(c) Encoding with Prior Writes

(d) Encoding, Unwritable Cell

Figure 9: Encoding at different stages in the life of a page. The edge metric in the Viterbi algorithm is determined by the cost of changing the entry in a given cell.

X. Lifetime Extension Comparison

A. Methodology

Our implementation of coset coding uses a convolutional code with 512-states in combination with 100 SCPs. Random inputs of length 501 choose the initial coset representative, which is a codeword in an extended Hamming code of length 1024. The Viterbi algorithm is then used to choose a coset representative which is then written to a 4kB page.

Since enumerative codes have a pre-determined number of rewrites, we used the numbers from their papers as their lifetime improvements.

Our evaluation of floating codes uses the mapping of 2 logical bits to 4 physical cells presented in [30]. Random data, viewed as a sequence of pairs of logical bits, is written to a 4kB page and each time a pair is rewritten a counter specific to that pair is decremented. Erasure is required when the counter associated with some pair of bits reaches zero. Floating codes were implemented in combination with 100 SCPs for fair comparison with coset coding. The comparison was in fact a little unfair to coset coding since each floating code SCP contained two replacement bits instead of one.

Coding Technique		Number of Writes Before Erasure is Required					
Scheme Name	**Encoding Gran**	**SLC**	**4-Level MLC**	**8-Level MLC**	**16-Level MLC**	**200-Level MLC**	**ECC**
Coset Coding + ECC + 100 SCPs	501-bits	4	18	48	112	1632	Yes
Floating Codes + 100 SCPs [30]	2-bits	1	6	17	41	651	N/A
Enumerative Coding [32]	3-bits	2	N/A	N/A	N/A	N/A	N/A

Table VI: Comparison of Different WOM Schemes for 4KB page, 100% Overhead, Random Inputs.

Coding Technique		Expected Lifetime Gain					
Scheme Name	**Encoding Gran**	**SLC**	**4-Level MLC**	**8-Level MLC**	**16-Level MLC**	**200-Level MLC**	**ECC**
Coset Coding + ECC + 100 SCPs	501-bits	300%	500%	586%	640%	720%	Yes
Floating Codes + 100 SCPs [30]	2-bits	0%	100%	143%	173%	227%	N/A
Enumerative Coding [32]	3-bits	100%	N/A	N/A	N/A	N/A	N/A

Table VII: Comparison of Different WOM Schemes for 4KB page, 100% Overhead, Random Inputs. Waterfall Coding (mod 2) is Used as a Baseline.

In the above schemes the algorithm and/or alphabet for writing to a given cell or group of cells is independent of the number of prior writes. When the writing scheme is allowed to be a function of the number of writes it is possible to obtain very significant gains. A representative example taken from [27][28] uses 200% overhead to achieve 11.4 writes before erasure with SLCs. If we view a WOM/WAM code as an inner code and we use a coset code to construct a concatenated code then the bit flip gains will be additive. A system implementation of WOM/WAM coding requires that the number of writes to a given cell or block of cells be available at the encoder. This is also required for implementation of the Viterbi algorithm in coset coding. In coset coding this information can be read off from the cell state but this is not the case in WOM/WAM coding. If it were necessary to record separately in Flash memory the number of writes to a given cell or block of cells, then the result would be a significant expansion of memory overhead. This cost is not considered in [27][28], so we have not included performance of WOM/WAM coding in Tables VI and VII.

B. Results

Table VI compares the effect of different coding techniques on the number of writes to a 4KB page before erasure is required. Table VII provides a different perspective on the same data. Binary data is written to memory using waterfall coding [15] in which the MLC value is interpreted modulo 2 (for example a physical value of 7 is read as a logical 1). The coset code results were obtained with 100 SCPs by applying the Viterbi edge metrics described in Section IX to a convolutional code with constraint length 9.

We conclude from Tables VI and VII that the lifetime gains associated with coset coding are superior to those associated with other techniques by a factor between 3 and 5. Coset coding gains increase with the number of levels in the cells. To approximate coset coding gains in the infinite case we simulated coset coding using 200-level cells, resulting in a lifetime gain of 720%. This shows how we can still get significant lifetime benefit using more than 16-level MLCs.

XI. Worst Case Performance

Tables VI and VII report results for random writes leaving open the possibility that a worst case pattern of writes might compromise the expected lifetime extension. Floating codes have the property that the mapping from input bits to recorded bits is local, and repeated writes to the same bits will then cause premature erasure. Coset codes have the property that there are many ways to represent every pattern of inputs, and that the different ways are distributed across the entire page. When using coset coding there is little or no difference between worst case and average case lifetime extension.

XII. Future Work

Storage systems can benefit from using coset coding. This will be the subject of future work. Since our code has positive lifetime extension, we believe it will render in-place updates feasible, reducing the need to migrate data to a new block when writing to memory. Improvements will be expressed in terms of *write amplification*–the ratio of data written to NAND Flash to the data written by the host computer.

XIII. Conclusion

We have presented a technique for increasing the number of writes we can perform to a page of data before it must then be erased using the Viterbi algorithm, linear codes, SCPs, and coset coding. We have also demonstrates that our technique translates directly into increased lifetime for flash devices by increasing the number of writes before a block needs to be erased. We have demonstrated lifetime gains for SLCs of 300%, for 4-level MLCs of 500%, for 8-level MLCs of 586%, and finally for 16-level MLCs of 640%. We have also shown that our technique can successfully be used with ECC to tolerate bit errors at an arbitrary granularity.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under grant CCF-111-5367.

REFERENCES

[1] Seagate, "Barracuda datasheet," Nov. 2011. [Online]. Available: http://www.seagate.com/files/staticfiles/docs/pdf/datasheet/disc/barracuda-ds1737-1-1111us.pdf

[2] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *5th Annual USENIX Conf. on File and Storage Tech.*, 2007, pp. 17–29.

[3] F. Masuoka, "Charge pumping semiconductor memory," U.S. Patent 4 223 333, 1980.

[4] E. Harari, "The Non-Volatile memory industry - a personal journey," in *3rd International Memory Workshop*, May 2011, pp. 1 –4.

[5] "Western digital AV-GP WD20EURS 2TB 64MB cache SATA 3.0Gb/s 3.5" internal hard drive -Bare drive." [Online]. Available: http://www.newegg.com/Product/Product.aspx?Item=N82E16822136783

[6] "Seagate constellation ES ST1000NM0001 1TB 7200 RPM 64MB cache SAS 6Gb/s 3.5" internal enterprise hard drive -Bare drive." [Online]. Available: http://www.newegg.com/Product/Product.aspx?Item=N82E16822148868

[7] "OCZ agility 4 AGT4-25SAT3-128G 2.5" 128GB SATA III MLC internal solid state drive (SSD)." [Online]. Available: http://www.newegg.com/Product/Product.aspx?Item=N82E16820227807

[8] "Intel 313 series hawley creek SSDSA2VP020G301 2.5" 20GB SATA II SLC internal solid state drive (SSD)." [Online]. Available: http://www.newegg.com/Product/Product.aspx?Item=N82E16820167118

[9] D. Andersen and S. Swanson, "Rethinking flash in the data center," *IEEE Micro*, vol. 30, no. 4, pp. 52 –54, July-Aug 2010.

[10] Semiconductor Industry Association, "International technology roadmap for semiconductors pids," Tech. Rep., 2011.

[11] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2007, pp. 1126–1130.

[12] A. Nozoe, H. Kotani, T. Tsujikawa, K. Yoshida, K. Furusawa, M. Kato, T. Nishimoto, H. Kume, H. Kurata, N. Miyamoto, S. Kubono, M. Kanamitsu, K. Koda, T. Nakayama, Y. Kouro, A. Hosogane, N. Ajika, and K. Koyashi, "A 256-Mb multilevel flash memory with 2-MB/s program rate for mass storage applications," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 11, pp. 1544 –1550, Nov. 1999.

[13] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for Multi-Level flash memories: Correcting asymmetric Limited-Magnitude errors," in *IEEE International Symposium on Information Theory*, Jun. 2007, pp. 1176 –1180.

[14] A. Modelli, A. Visconti, and R. Bez, "Advanced flash memory reliability," in *International Conference on Integrated Circuit Design and Technology*, 2004, pp. 211 – 218.

[15] L. Lastras-Montano, M. Franceschini, T. Mittelholzer, J. Karidis, and M. Wegman, "On the lifetime of multilevel memories," in *IEEE International Symposium on Information Theory*, 2009, pp. 1224 – 1228.

[16] A. Bandyopadhyay, G. Serrano, and P. Hasler, "Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method," in *IEEE International Symposium on Circuits and Systems*, 2005, pp. 2148 – 2151.

[17] 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.

[18] Mingahai Qin, Eitan Yaakobi, and Paul Siegel, "Optimized cell programming for flash memories with quantizers," in *IEEE International Symposium on Information Theory*. Cambridge, MA, USA: IEEE, 2012.

[19] M. Bagatin, S. Gerardin, and A. Paccagnella, "Alpha-Induced soft errors in floating gate flash memories," *IEEE International Reliability Physics Symposium*, 2012.

[20] S. Gerardin, M. Bagatin, A. Paccagnella, G. Cellere, A. Visconti, S. Beltrami, C. Andreani, G. Gorini, and C. D. Frost, "Scaling trends of neutron effects in MLC NAND flash memories," in *International Reliability Physics Symposium*. IEEE, May 2010, pp. 400–406.

[21] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in NAND flash memories," in *International Reliability Physics Symposium*. IEEE, May 2008, pp. 9–19.

[22] G. D. Forney, "Coset codes. i. introduction and geometrical classification," *Trans. on Info. Theory*, vol. 34, no. 5, pp. 1123–1151, 1988.

[23] ——, "Coset codes. II. binary lattices and related codes," *Trans. on Info. Theory*, vol. 34, no. 5, pp. 1152–1187, 1988.

[24] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1248–1271, Aug. 1997.

[25] J. Gray and C. van Ingen, "Empirical measurements of disk failure rates and error rates," MSR, Tech. Rep. MSR-TR-2005-166, Jan. 2007.

[26] R. L. Rivest and A. Shamir, "How to reuse a write-once memory," *Information and Control*, vol. 55, no. 1–3, pp. 1 – 19, 1982.

[27] E. Yaakobi, S. Kayser, P. Siegel, A. Vardy, and J. Wolf, "Efficient two-write WOM-codes," in *Info. Theory Wrkshp*, 2010, pp. 1 –5.

[28] S. Kayser, E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf, "Multiple-write WOM-codes," in *Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing*, Oct. 2010, pp. 1062 – 1068.

[29] T. Cover, "Enumerative source encoding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, pp. 73 – 77, Jan 1973.

[30] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *International Symposium on Information Theory*. IEEE, Jun. 2007, pp. 1166–1170.

[31] F. Chierichetti, H. Finucane, Z. Liu, and M. Mitzenmacher, "Designing floating codes for expected performance," *IEEE Transactions on Information Theory*, vol. 56, no. 3, pp. 968–978, Mar. 2010.

[32] A. Jagmohan, M. Franceschini, and L. Lastras, "Write amplification reduction in NAND flash through multi-write coding," in *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies*. IEEE, May 2010, pp. 1–6.

[33] H. C. So and S. C. Wong, "Multibit-per-cell non-volatile memory with error detection and correction," U.S. Patent 5 909 449, Jun., 1999.

[34] Michele M. Franceschini and Ashish Jagmohan, "Multi-Write endurance and error control coding of Non-Volatile memories," Apr. 2012.

[35] A. V. Kuznetsov and B. S. Tsybakov, "Coding in a memory with defective cells," *Probl. Peredachi Inf.*, vol. 10, pp. 52–60, 1974.

[36] C. Heegard and A. Gamal, "On the capacity of computer memory with defects," *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 731 – 739, Sept 1983.

[37] E. Hwang, S. Jeon, R. Negi, B. V. K. V. Kumar, and M. Cheng, "Coding with side information for radiation-tolerant memory devices," *IPN Progress Report*, vol. 42, no. 187, Nov. 2011.

[38] E. Hwang, R. Negi, and B. Kumar, "Additive encoding low-density parity-check (ae-ldpc) codes for two-dimensional magnetic recording (tdmr)," in *International Conference on Computing, Networking and Communications*, Feb 2012, pp. 481 –485.

[39] G. D. Forney, "Trellis shaping," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 281–300, Mar. 1992.

[40] S. Cho and H. Lee, "Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. New York, NY, USA: ACM, 2009, pp. 347–357.

[41] S. Lin and D. J. Costello, Jr, *Error Control Coding*, 2nd ed. Pearson Prentice Hall, 2004.

[42] A. Calderbank, A. Duel-Hallen, P. Fishburn, and A. Rabinovich, "Interpolation by convolutional codes, overload distortion, and the erasure channel," *Trans. on Info. Theory*, vol. 45, no. 1, pp. 94 –105, Jan 1999.