Shaping Codes for Structured Data

Yi Liu* and **Paul H. Siegel***
*Electrical and Computer Engineering Dept., University of California, San Diego, La Jolla, CA 92093 U.S.A
{*yil333, psiegel*}*@ucsd.edu*

***Abstract*—In this work, we study data shaping codes for flash memory. We first review a recently proposed *direct* shaping code for SLC (one bit per cell) flash memory that reduces wear by minimizing the average fraction of programmed cells. Then we describe an adaptation of this algorithm that provides data shaping for MLC (two bits per cell) flash memory. It makes use of a page-dependent cost model and is designed to be compatible with the standard procedure of row-by-row, page-based, wordline programming. We also give simulation results demonstrating the performance of these data shaping codes when applied to English language text. Finally, we use a random walk model to analyze the potential error propagation properties of direct shaping codes when used in a noisy memory.**

I. Introduction

NAND flash memory has become a widely used data storage technology. It uses rectangular arrays, or *blocks* of floating-gate transistors (commonly referred to as *cells*) to store information. The flash memory cells gradually wear out with repeated writing and erasing, referred to as *program/erase (P/E) cycling*, but the damage caused by P/E cycling is dependent on the programmed cell level. For example, in SLC flash memory, each cell has two different states, erased and programmed, represented by 1 and 0, respectively. Storing 1 in a cell causes less damage, or *wear*, than storing 0. More generally, in multilevel flash memories, the cell wear is an increasing function of the programmed cell level.

Endurance coding, an encoding technique that reduces wear by converting unconstrained data to encoded sequences with a prescribed distribution of cell levels, has been proposed in [1] as a solution to this problem. In that work, for a given cell-level "wear-cost" model and a specified target code rate, the optimal distribution of levels that minimizes the average wear-cost was determined. Theoretically achievable gains for SLC and 4-level MLC devices were also computed. *Adaptive endurance coding* (AEC), which uses a concatenation of lossless data compression with endurance coding, was proposed for structured source data. The performance of enumerative endurance codes that reduce the average fraction of 0 symbols was evaluated empirically on SLC devices, as well as on MLC devices (applied to both pages). Results of a device-level simulation of AEC on SLC flash were also presented.

Data shaping for structured data can be viewed as a coding technique that combines lossless compression and endurance coding into a single encoding operation. The intent is to efficiently transform the structured data sequence directly into a sequence that induces less cell wear.

In this paper, we consider the problem of designing data shaping codes for MLC flash memory. In Section II we review a recently proposed shaping code that was designed for SLC flash memory. We illustrate its effectiveness by given an application to the English-language novel *Gone with the Wind*. In Section III we describe the structure and programming of MLC flash memory, and then propose a content-dependent cost model that reflects the cell wear associated with programming each level. Based on this model, we extend the data shaping code introduced in Section II to MLC flash memory. In Section IV, we study the problem of decoder error propagation by means of a random-walk recurrence analysis. We show that if a sufficiently large number of codewords have been read correctly, error propagation can be avoided.

II. Shaping Codes for SLC Flash

A. Encoder and Decoder

In the context of SLC flash memory, shaping codes for structured data – referred to as *direct* shaping codes – were first introduced by Sharon et al. [2]. Their construction makes use of a rate-1, adaptive encoding dictionary $\mathbf{D}$ that is used to map successive words of length m in the input sequence $\mathbf{w}$ into codewords of length m. The dictionary $\mathbf{D}$ comprises two lists, a dynamically ordered list $\mathcal{X}$ and a fixed output codeword list $\mathcal{Y}$. The output list $\mathcal{Y}$ consists of codewords $\mathbf{y_k} \in \{0,1\}^m$, $k = 1, 2, \ldots, 2^m$, ordered by non-decreasing *cost*, where the cost of a codeword $\mathbf{y_k}$ in this context means the number of 0 symbols in $\mathbf{y_k}$.

At any given time during the encoding process, the list $\mathcal{X}$ consists of pairs $\{(\mathbf{x}_k, n_k)\}$, $k = 1, 2 \ldots, 2^m$, where each pair represents a distinct length-m input word $\mathbf{x}_k \in \{0,1\}^m$ and the *frequency count*, or number of times, n_k, that it has appeared up to that point in the input data sequence. The list of pairs is dynamically ordered such that the n_k values are in non-increasing order, i.e, $n_1 \geqslant n_2 \geqslant \ldots \geqslant n_{2^m}$. At the start of the encoding process, the n_k are all initialized to the value 0, and the words $\mathbf{x}_k$ are ordered lexicographically.

Encoding proceeds as follows. When a data word $\mathbf{x}$ of length m is encountered in the input data sequence, its corresponding pair $(\mathbf{x}_k, n_k) = (\mathbf{x}, n)$ is found in the ordered list $\mathcal{X}$. The encoder then maps $\mathbf{x}$ to the length-m output word $\mathbf{y}$ that occupies the same position in output list $\mathcal{Y}$. The frequency count n of the word $\mathbf{x}$ is increased by 1 and the list $\mathcal{X}$ is reordered accordingly, with the pair $(\mathbf{x}, n+1)$ moved above all pairs with counts less than or equal to $n+1$.

Example 1. Consider a direct shaping code with parsing length $m = 2$. In the encoding dictionary $\mathbf{D}$, the ordered output list $\mathcal{Y}$ is $\{11, 10, 01, 00\}$. Consider the length-14 data sequence $\mathbf{w} = 10.11.00.10.11.10.00$. The first six length-2 input words contain three words $\mathbf{x}_1 = 10$, two words $\mathbf{x}_2 = 11$ and one word $\mathbf{x}_3 = 00$. The state of the dictionary $\mathbf{D}$ after encoding these 6 words is shown in Table I(a).

The final input word is $\mathbf{x} = 00$. According to Table I(a), it is mapped to output codeword $\mathbf{y_3} = 01$. We then add 1 to the count n_3 in the table and update the ordering of the entries in the input list $\mathcal{X}$. The updated dictionary $\mathbf{D}$ is shown in Table I(b).

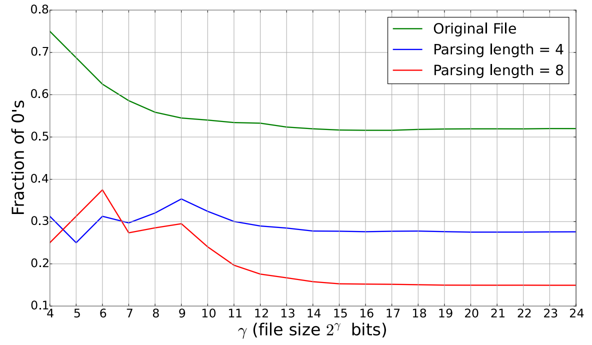

Fig. 1: Direct shaping codes applied to *Gone with the Wind*.

TABLE I: Length-2 dictionary when encoding $\mathbf{x} = 00$.

(a) Before encoding

x	n	y
10	3	11
11	2	10
00	1	01
01	0	00

(b) After encoding

x	n	y
10	3	11
00	2	10
11	2	01
01	0	00

The decoder dynamically reconstructs the dictionary and inverts the encoder mapping. When a codeword $\mathbf{y}$ is encountered in the encoded sequence, it is mapped to the binary length-m data word $\mathbf{x}$ that occupies the same position in the input list $\mathcal{X}$. Then the frequency count n of the word $\mathbf{x}$ is increased by 1 and the ordering of the input list $\mathcal{X}$ is updated accordingly. This rate-1 shaping code incurs no rate penalty and both encoding and decoding can be implemented with low complexity.

B. *Simulation Results*

We now present simulation results quantifying the endurance gain which can be achieved by the use of SLC direct shaping codes. The structured data we used was the English-language novel *Gone with the Wind*, represented in ASCII Code. The data file size was about 2^{24} bits. We evaluated the shaping code using encoder parsing length m equal to 4 bits and 8 bits. Fig. 1 shows the fraction of 0 symbols in the first 2^γ bits in the original data file and in the corresponding encoded files, for $\gamma = 4, \ldots, 24$. The fraction of 0's in the entire original data file was approximately 0.52. With parsing length equal to 4 bits, the fraction of 0's dropped to about 0.28, and with parsing length of 8 bits, the fraction was further reduced to about 0.15.

III. Data Shaping Codes for MLC Flash

A. *Cost Model for MLC Flash*

Every cell in an MLC flash memory can be charged to 4 different values of the threshold voltage, V_{th}. Thus, each cell can represent 2 bits of information. The levels are denoted by $0, 1, 2, 3,$ respectively, from lowest to highest, and the corresponding binary representations are given by the Gray code $11, 10, 00, 01$. In the binary representation of a level, the left-most bit is called the *lower bit* and the right-most bit is called the *upper bit*. To program the MLC flash cell, we assume the controller uses a two-step process. It first charges the cell to an intermediate voltage level that reflects the value of the lower bit. Then, taking into account the value of the upper bit, it completes charging of the cell to reach the appropriate final threshold voltage level. This process is shown schematically in Fig. 2.

The rows of cells in a flash memory block are called *wordlines*, and the wordlines are programmed sequentially, in a row-by-row manner. The lower bits of cells in a wordline constitute the *lower page*, while the upper bits form the *upper page*. When programming a block, information is programmed separately to lower pages and upper pages. During data retrieval, pages are also retrieved independently, with lower bit values recovered by reference to read threshold V_B, and upper bit values by reference to read thresholds V_A and V_C, also shown in Fig. 2.

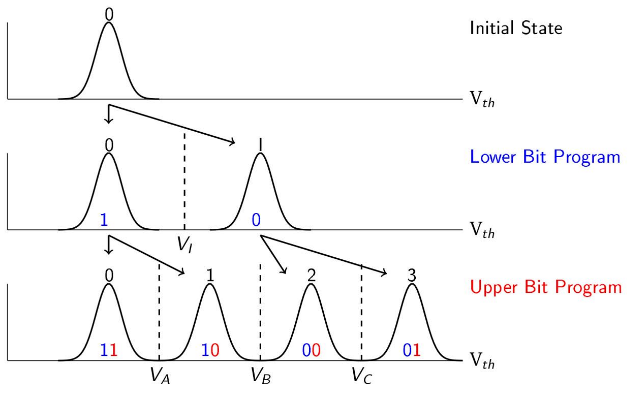

Fig. 2: Schematic of MLC flash cell programming.

To characterize and quantify the damage caused by different programmed levels in an MLC flash memory, we performed an experiment on several blocks in which we repeatedly programmed all the cells in the block to a specified, constant level, erasing the block (i.e., reducing voltage to level 0) after each programming cycle. The device we used was a 26 nm MLC flash chip with the default read threshold positions. After every 100 of these P/E cycles, we programmed the block with pseudo-random data, read it back, and recorded the cell error rate. The error rates, averaged over the blocks used in the experiment, are shown in Fig. 3. We see that cell damage caused by the levels $0, 1, 2, 3$ increases monotonically. A *cost model*, in the form of a vector of cell-level costs $[c_0, c_1, c_2, c_3]$, is used to quantify the relative amount of device wear associated with each of the cell levels. In practice, these costs will satisfy $c_0 \leqslant c_1 \leqslant c_2 \leqslant c_3$, reflecting the increased damage induced by higher programmed levels.

Given a length-m cell-level codeword $\mathbf{z} = [z_1, \ldots, z_m]$, we denote the cost associated with the symbol z_i by $c(z_i)$, and the total cost $c(\mathbf{z})$ associated with programming $\mathbf{z}$ is assumed to be the sum of the individual symbol costs; i.e., $c(\mathbf{z}) = \sum_{i=1}^m c(z_i)$.

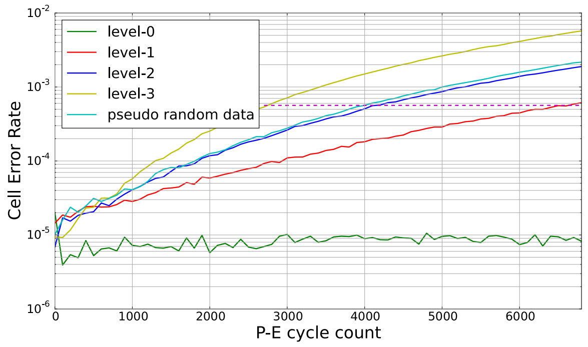

Fig. 3: Cell error rate of MLC Flash for different programming levels, while repeatedly programming a constant level.

A method for determining physically meaningful values of c_i was proposed by Li et al. in [3]. It is based upon measuring the number of cycles it takes for the error rate associated with a certain programmed level to reach a prespecified *maximum tolerable cell error rate*. If the design lifetime of the flash memory is T_0 P/E cycles, the maximum tolerable cell error rate CER_{max} is defined as the cell error rate after the memory is programmed with random data for T_0 P/E cycles. We view the damage caused by programming random data to the memory as being proportional to $1/T_0$.

We define $\Phi^i(T)$ to be the cell error rate observed after T cycles of programming the blocks to the level i, for $i = 0, 1, 2, 3$. Let T^i_{max} denote the P/E cycle number at which $\Phi^i(T)$ equals CER_{max}. We view the damage caused by programming to level i as proportional to $1/T^i_{max}$, so we define the cost associated with each level i, $i = 0, 1, 2, 3$, to be

$$c_i = \frac{T_0}{T^i_{max}}.$$

Example 2. We calculate the cost model corresponding to the error rate results in Fig. 3, assuming a design lifetime of $T_0 = 4000$ cycles. The error rate associated with level 0 remains essentially constant, so we set $c_0 = 0$. For the other levels, we find that $T^1_{max} = 6800$, $T^2_{max} = 4300$ and $T^3_{max} = 2800$, so the complete cost model is computed to be $[0, 0.59, 1.07, 1.43]$

B. *Shaping Codes for MLC Flash*

Application of the SLC shaping code independently to lower and upper pages will not be effective in reducing the average cost. This can be seen by referring to the schematic of the MLC flash cell programming process in Fig. 2. As shown in the schematic, the cost associated with an upper bit 1 or 0 depends on the value of the corresponding lower bit in the cell. The proposed MLC shaping encoder achieves improved wear reduction by using lower-page dependent dictionaries when encoding the upper pages, as we now describe.

First, fix a parsing length m. Encoding and programming of wordlines is done in a row-by-row, sequential manner. Suppose that we want to encode L data words to both lower and upper pages. For the lower pages, the encoder simply uses the direct shaping code with parsing length n. When programming an upper page, however, the encoder first reads the corresponding, previously programmed lower page.

TABLE II: Ordered list of upper page words for lower page codeword 1110.

index	Output List	index	Output List
0	1110	8	1000
1	1111	9	0100
2	1100	10	0010
3	1010	11	1001
4	0110	12	0101
5	1101	13	0011
6	1011	14	0000
7	0111	15	0001

Now, suppose the lower page that has been programmed and presumably correctly recovered consists of the sequence of length-m codewords $\mathbf{v}^{(\mathbf{k})}$, $k = 0, \ldots, L$. Consider the sequence of length-m data words $\mathbf{w}^{(\mathbf{k})}$, $k = 0, \ldots, L$ that need to be encoded for the upper page. To encode the kth data word $\mathbf{w}^{(\mathbf{k})}$, we encode using a direct shaping code that is based upon an adaptively-built dictionary that depends on the corresponding lower page codeword $\mathbf{v}^{(\mathbf{k})}$. The only difference in the operation of each lower-page dependent shaping encoder is that the ordering of the length-m encoder output words in terms of increasing cost depends specifically on the lower page codeword $\mathbf{v}$ and the cost model $[c_0, c_1, c_2, c_3]$.

To illustrate the design of the encoder, we will use the cost model $[c_0, c_1, c_2, c_3] = [0, 1, 1, 2]$. This simple cost model is motivated as follows. Consider the standard lower-upper page binary representation of the cell levels: 0=11, 1=10, 2=00, 3=01. We assign a cost of 0 to lower bit value 1, and a cost of 1 to lower bit value 0, in accordance with the MLC cell programming schematic. When the lower bit value is 1, we assign a cost of 0 to upper bit value 1, and a cost of 1 to upper bit value 0. On the other hand, when the lower bit value is 0, we assign a cost of 0 to upper bit value 0, and a cost of 1 to upper bit value 1. Again, these cost assignments are consistent with the MLC cell programming schematic. The cost associated with a cell level is then defined as the sum of the corresponding lower bit and upper bit costs.

Choose parsing length $n = 4$ and assume the lower page codeword is $\mathbf{v} = \mathbf{1110}$. For the first three bits, where the corresponding lower bit is a 1, programming the upper bit to 1 is better than to 0. For the last bit, where the corresponding lower bit is a 0, programming the upper bit to 0 is better than to 1. This implies that the lowest cost output word in the dictionary is 1110, corresponding to cell level word 0002, which has total cost 1. Similar reasoning leads to the ordered list of output words shown in Table II. The corresponding list of cell level words is given in Table III.

It is easy to verify that the costs of the corresponding cell level words are non-decreasing: cost 2 for words 1 to 4, cost 3 for words 5 to 10, cost 4 for words 11 to 14, and cost 5 for word 15. In general, the encoder uses 2^m direct shaping encoders operating in a sequence determined by the sequence of lower page codewords $\mathbf{v}^{(\mathbf{k})}$, $k = 0, \ldots, L$.

TABLE III: Corresponding list of cell level words for lower page codeword 1110.

index	Output List	index	Output List
0	0002	8	0112
1	0003	9	1012
2	0012	10	1102
3	0102	11	0113
4	1002	12	1013
5	0013	13	1103
6	0103	14	1112
7	1003	15	1113

C. *Encoding Algorithm for MLC Shaping Codes*

The upper page encoder uses an encoding table in which the order of the output codewords depends on the lower page codeword. The encoding table provides a mapping from an index $I \in \{1, \dots, 2^m\}$ to the corresponding upper page output word $\mathbf{y}$ in the ordered list. This mapping can be implemented using enumerative coding, introduced by Cover [4]. To explain the algorithm, we first introduce some notation. We denote the length-m lower page codeword by $\mathbf{v}$ and the upper page word by $\mathbf{y}$. They determine the cell level output word $\mathbf{z} = [z_1, \dots, z_m]$, where z_i is the cell level associated with the lower and upper bit pair $v_i y_i$. We denote the corresponding cost by $u_i = c(z_i) \in [c_0, c_1, c_2, c_3]$, $i = 1, 2, \dots, m$, and the total cost by $U = c(\mathbf{z}) = \sum_{i=1}^{m} u_i$.

Let $n(\mathbf{v}, U, y_1, y_2, \dots, y_k)$ be the number of upper page words $\mathbf{y}$ that, together with lower page codeword $\mathbf{v}$, yield total cost U, and whose first k coordinates are equal to $[y_1, y_2, \dots, y_k]$. To determine $n(\mathbf{v}, U, y_1, y_2, \cdots, y_k)$, we first calculate the polynomial $g_{\mathbf{v},k}(x) = (x^{c_0} + x^{c_1})^{n_1}(x^{c_2} + x^{c_3})^{n_0}$, where n_b, $b \in \{0, 1\}$ is the number of bits equal to b in the last $m-k$ bits of the lower page codeword, $[v_{k+1}, \dots, v_m]$. The degrees of terms in $g_{\mathbf{v},k}(x)$ represent the possible total costs of cell level vectors $[z'_{k+1}, \dots, z'_m]$ with associated lower bit vector $[v_{k+1}, \dots, v_m]$, and the corresponding coefficients represent the number of such vectors. Then $n(\mathbf{v}, U, y_1, y_2, \cdots, y_k)$ is equal to the coefficient of the term in $g_{\mathbf{v},k}(x)$ whose degree is equal to $U - \sum_{i=1}^{k} u_i$. If there is no such term in the polynomial, we set $n(\mathbf{v}, U, y_1, y_2, \cdots, y_k) = 0$. In particular, the polynomial $g_{\mathbf{v},0}(x)$ corresponding to $[v_1, \dots, v_m]$ tells us the costs W_j, $j = 1, \dots, J$ of all possible length-m cell level vectors with associated lower bit vector $\mathbf{v}$, along with the number of upper page words that together with $\mathbf{v}$ produce these costs. From these we calculate $n(\mathbf{v}, U)$, the number of upper page words that produce total cost less than or equal to U, given lower page codeword $\mathbf{v}$. Example 3 illustrates how to calculate $n(\mathbf{v}, U, y_1, y_2, \cdots, y_k)$.

Example 3. We want to calculate $n(\mathbf{v}, U, 1, 1)$ where the lower page codeword is $\mathbf{v} = \mathbf{1110}$ and the total cost is $U = 2$. The combined cost of the first two cells is 0, so the remaining cost is 2. We calculate the polynomial $(x^{c_0} + x^{c_1})^{n_1}(x^{c_2} + x^{c_3})^{n_0} = (1 + x)(x + x^2) = x + 2x^2 + x^3$. The coefficient of x^2 is 2, meaning there are two possible codewords with total cost $U = 2$ and first two bits $[1, 1]$.

With enumerative coding, instead of storing the ordered upper page output word list, we only need to store the values $n(\mathbf{v}, U, y_1, y_2, \cdots, y_k)$ and $n(\mathbf{v}, U)$. Algorithm 1 describes the encoding process.

Algorithm 1 Encoding upper page codewords

```
Input: Codeword length $m$, lower page codeword $\mathbf{v}$, index $I$
Output: Output codeword $\mathbf{y} = (y_1, y_2, \dots, y_m)$
1: Find $j$ such that $I > n(\mathbf{v}, W_{j-1})$ and $I \leqslant n(\mathbf{v}, W_j)$, set
   $I = I - n(\mathbf{v}, W_{j-1})$
2: If $I > n(\mathbf{v}, W_j, 0)$ set $y_1 = 1$ and set $I = I - n(\mathbf{v}, W_j, 0)$,
   otherwise set $y_1 = 0$
3: For $i$th bit, if $I > n(\mathbf{v}, W_j, y_1, \cdots, y_i - 1, 0)$ set $y_i = 1$
   and set $I = I - n(\mathbf{v}, W_j, y_1, \cdots, y_i - 1, 0)$, otherwise set
   $y_i = 0$
```

D. *Simulation Results*

We assessed the performance of the MLC shaping encoder using the same ASCII data file for the English-language novel *Gone with the Wind*. The data file was divided into two consecutive subfiles of size 2^{23} bits; the first was used for lower page encoding and the second for upper page encoding.

Fig. 4 shows the respective fractions of the encoded cell levels $0, 1, 2, 3$ appearing in the cell level sequences produced when no shaping code was applied to the first 2^{γ} bits of each subfile for $\gamma = 4, \dots, 23$. Fig. 5 shows the corresponding fractions when the proposed MLC shaping encoder is applied using the cost model $[c_0, c_1, c_2, c_3] = [0, 1, 1, 2]$ discussed above. The average cost is also plotted in both figures.

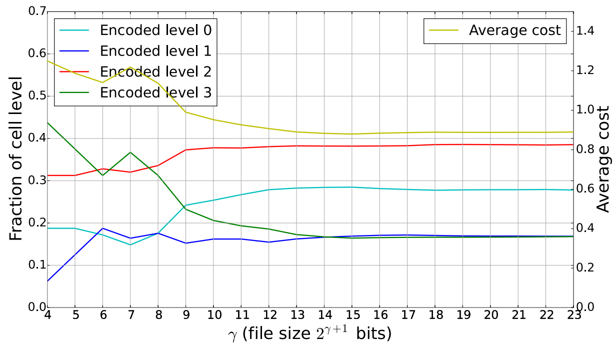

Fig. 4: Fractions of MLC cell levels for segments of *Gone with the Wind* without a shaping code.

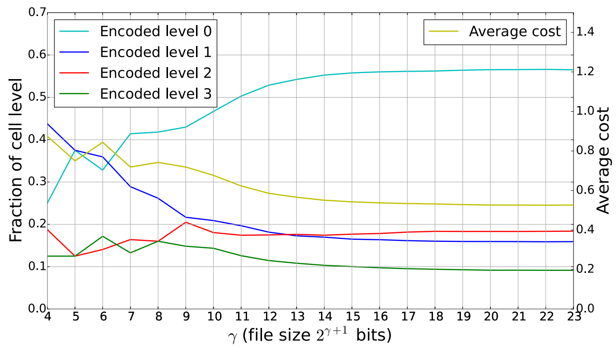

Fig. 5: Fractions of MLC cell levels for segments of *Gone with the Wind* with MLC shaping code.

Using these results, we see that for file size 2^{24}, the average cost when no coding is applied is 0.89. When SLC shaping is applied independently to lower and upper pages, the average cost is 0.62 (not shown). In contrast, after MLC shaping, the average cost is reduced to 0.53. When the empirical cost model of Example 2 is used in the encoding, the average costs for no coding, independent SLC shaping, and MLC shaping are 0.75, 0.46, and 0.42, respectively.

IV. Error Propagation Analysis of Direct Shaping Codes

The direct shaping decoder reproduces the dynamic construction of the encoding table. Errors in reading the flash memory can lead to incorrect word frequency counts that, in turn, can cause decoding errors if word counts are not sufficiently separated. In this section we introduce a framework for analyzing potential error propagation properties of direct shaping codes, based upon recurrence properties of random walks.

We first establish some notation. Assume t data words have been encoded. Let $\mathbf{N}(t) = \{n_1(t), n_2(t), n_3(t), \ldots, n_{2^m}(t)\}$ be the word counts and let $N_i(t) = n_i(t) - n_{i+1}(t)$ denote the *distance* between the counts for the ith and $(i+1)$st words. We say $N(t)$ is *stable* if $n_1(t) > n_2(t) > \ldots > n_{2^m}(t)$. We sometimes use the word stable to describe the dictionary that has stable $\mathbf{N}(t)$.

A. Recurrence probability in a two-word dictionary

We first consider the case $m = 1$, where the input dictionary contains only two words, denoted $\mathbf{w}_1$ and $\mathbf{w}_2$ for convenience. Recall that output word 1 has lower cost than output word 0. Let $P = \{P_1, P_2\}$ be the true probability distribution of input words $\mathbf{w}_1$ and $\mathbf{w}_2$, and assume $P_1 \geqslant P_2$. Suppose that, at time t_0, the input word counts satisfy $n_1(t_0) \neq n_2(t_0)$. Since there are only 2 words, the dictionary after time t can be represented by the word that has the higher count, denoted by $s(t) \in \{\mathbf{w}_1, \mathbf{w}_2\}$, and the distance we denote simply by $N(t) = n_1(t) - n_2(t)$. Let $s_e(t), N_e(t)$ represent the dictionary evolution during the encoding process, and let $s_d(t), N_d(t)$ represent the dictionary as reconstructed during decoding.

The distance behaves like a one-dimensional random walk, as depicted in Fig. 6. In the figure, the blue line represents the encoding process, starting at time denoted as $t = 0$, with $s_e(0) = \mathbf{w}_1$ and $N_e(0) = 5$.

Assuming no read errors through time $t = 5$, the decoder correctly reconstructs the encoding dictionary, retracing the blue line. At time $t = 5$, we have $s_d(5) = s_e(5) = \mathbf{w}_1$ and $N^e(5) = N^d(5) = 4$. At $t = 6$, the input was $\mathbf{w}_2$, implying $s_e(6) = \mathbf{w}_1$ and $N_e(6) = 3$, producing output codeword 0. This point is marked by A on the blue line. Now, suppose a read error occurs at decoding time $t = 6$. The codeword is incorrectly read as 1, which is decoded as input $\mathbf{w}_1$, resulting in $s_d(6) = \mathbf{w}_1$ and $N_d(6) = 5$. This deviation is marked by A' on the red dashed line, which represents the decoding process. Assume no further read errors occur. The decoder trace remains separated from the encoder trace, but the following three codewords are nevertheless decoded correctly. At time

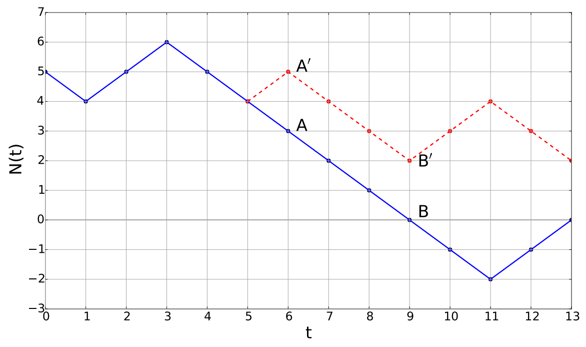

Fig. 6: Random walk behavior of a 2-symbol dictionary.

$t = 9$, since $N_e(9) = 0$, the encoder changes to $s_e(9) = \mathbf{w}_2$, while $N_d(9) = 2$ implies that the decoder continues with $s_d(9) = \mathbf{w}_1$. These two points are indicated by B and B' on the encoding and decoding traces, respectively. The next input word was $\mathbf{w}_2$ and the encoder output was 1, but, even though it is read correctly, the decoded word is $\mathbf{w}_1$. In fact, after this point, all codewords will be decoded incorrectly. This example shows that if the encoder reaches a point where $N_e(t) = 0$, there is the potential for error propagation.

For a general dictionary, for input words $\mathbf{w}_i$ and $\mathbf{w}_j$, with respective counts $n_i(t)$ and $n_j(t)$, suppose $n_i(t_0) \neq n_j(t_0)$ at time t_0. We say a *recurrence* occurs if, at some future time $t > t_0$, $n_i(t) = n_j(t)$. We want to evaluate the probability of such a recurrence. To do this, we use the following theorem, which generalizes a result stated in [5].

Theorem 1. *Consider two input words $\mathbf{w}_i, \mathbf{w}_j$, with probabilities P_i, P_j, respectively, and let $N(t) = n_i(t) - n_j(t)$. If $P_i \geqslant P_j$ and $N(t_0) \neq 0$ at some time t_0, the probability Q that a recurrence involving the two words $\mathbf{w}_i, \mathbf{w}_j$ will occur in the future is*

$$Q = \begin{cases} \left(\frac{P_j}{P_i}\right)^{N(t_0)} & \text{if } N(t_0) > 0 \\ 1 & \text{if } N(t_0) \leqslant 0. \end{cases}$$

■

From Theorem 1, we see that if we ensure that $N(t_0)$ is large enough for some t_0 (i.e., the dictionary is stable), the probability that $N(t) = 0$ for some $t > t_0$ will be small, and we can reduce the chance of error propagation. We would therefore like to determine the number of input words required for a dictionary to achieve stability, as well as the probability that the dictionary will subsequently become unstable. We study this in the next section.

B. Recurrence probability in a general dictionary

If $\mathbf{N}(t)$ is not stable, Theorem 1 states that there must be a recurrence in the future. So, we first assume that $\mathbf{N}(t_0)$ is stable for some t_0. Using Theorem 1, we then derive an upper bound on the recurrence probability for the entire dictionary. We denote by $\{\overline{i, i+1}\}$ the event that $n_i(t) \neq n_{i+1}(t), \forall t > t_0$ and by $\{i, i+1\}$ its complement. Notice that $\{i, i+1\}$ and $\{i+1, i+2\}$ are not independent, but $\{i, i+1\}$ and $\{i+2, i+3\}$ are independent. Note that the probability of a recurrence involving any two *adjacent* words in the dictionary is $P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{i, i+1\}\right\}$.

Lemma 2. *Let* $\mathbf{D}$ *be a direct shaping dictionary with parsing length* m. *Assume it is stable at some* t, *i.e.,* $\mathbf{N}(t)$ *is stable. Then, the probability of a recurrence involving two words in the dictionary, denoted* P_W, *is equal to the probability of a recurrence involving two* adjacent *words in the dictionary; i.e.,*

$$P_W = P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{i,i+1\}\right\}.$$

Proof: We prove that the two events are the same. Any recurrence condition can be viewed as a permutation of the set $\{1,2,\ldots,2^m\}$. So the set of all possible recurrence conditions is the symmetric group S_{2^m}. Using cycle notation [6] to represent the elements in S_{2^m}, any recurrence between two adjacent words is a transposition $(i,i+1)$. Since S_{2^m} can be generated by the elements $\langle(1,2),(2,3),\ldots,(k,k+1),\ldots,(2^m-1,2^m)\rangle$, the two events must be the same; namely, both are $S_{2^m}-\{\mathbb{1}\}$, where $\mathbb{1}$ is the identity element of S_{2^m}. ∎

We now derive an upper bound on the recurrence probability P_W. Let $P=\{P_1,P_2,\ldots,P_{2^m}\}$ be the true probability distribution of words, and assume $P_1\geqslant P_2\geqslant\ldots\geqslant P_{2^m}$.

Lemma 3. *The probability of a recurrence* P_W *satisfies*

$$P_W \leqslant 2-\prod_{i=1}^{2^{m-1}}(1-\frac{P_{2i}}{P_{2i-1}}^{N_{2i-1}(t)})-\prod_{i=1}^{2^{m-1}-1}(1-\frac{P_{2i+1}}{P_{2i}}^{N_{2i}(t)})$$

Proof: From the union bound, we have

$$\begin{aligned}
P_W &= P\left\{\bigcup_{i=1}^{2^m-1}\{i,i+1\}\right\}\\
&= P\left\{(\bigcup_{i=1}^{2^{m-1}}\{2i-1,2i\})\bigcup(\bigcup_{i=1}^{2^{m-1}-1}\{2i,2i+1\})\right\} \qquad (1)\\
&\leqslant P\left\{\bigcup_{i=1}^{2^{m-1}}\{2i-1,2i\}\right\}+P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{2i,2i+1\}\right\}.
\end{aligned}$$

Using the fact that $\{i,i+1\}$ and $\{i+2,i+3\}$ are independent, as are their complements, we get

$$\begin{aligned}
P_W &\leqslant P\left\{\bigcup_{i=1}^{2^{m-1}}\{2i-1,2i\}\right\}+P\left\{\bigcup_{i=1}^{2^{m-1}-1}\{2i,2i+1\}\right\}\\
&= 2-P\left\{(\bigcup_{i=1}^{2^{m-1}}\{2i-1,2i\})^C\right\}-P\left\{(\bigcup_{i=1}^{2^{m-1}-1}\{2i,2i+1\})^C\right\}\\
&= 2-P\left\{(\bigcap_{i=1}^{2^{m-1}}\overline{\{2i-1,2i\}})\right\}-P\left\{(\bigcap_{i=1}^{2^{m-1}-1}\overline{\{2i,2i+1\}})\right\} \qquad (2)\\
&= 2-\prod_{i=1}^{2^{m-1}}(1-\frac{P_{2i}}{P_{2i-1}}^{N_{2i-1}(t)})-\prod_{i=1}^{2^{m-1}-1}(1-\frac{P_{2i+1}}{P_{2i}}^{N_{2i}(t)}).
\end{aligned}$$

∎

We denote by $A(\mathbf{N}(t))$ the right side of equation (2). Notice that $A(\mathbf{N}(t))$ will decrease if any $N_i(t)$ increases, $i=1,2,\ldots,2^m-1$. Let $P(\mathbf{N}(t))$ denote the probability that after t steps the word counts are given by $\mathbf{N}(t)$. Clearly

$$P(\mathbf{N}(t))=\binom{t}{n_1(t),\ldots,n_{2^m}(t)}P_1^{n_1(t)}\ldots P_{2^m}^{n_{2^m}(t)}.$$

Combining Lemma 3 and the law of total probability, we have the following theorem.

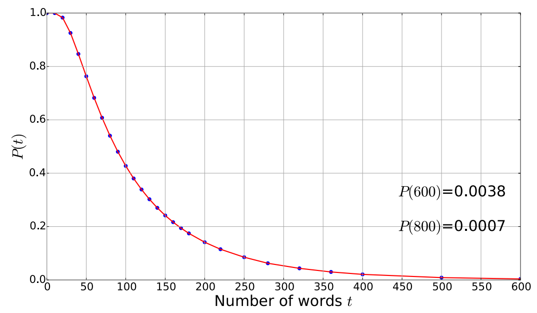

Fig. 7: Upper bound on $P(t)$ when $P=\{0.4,0.3,0.2,0.1\}$.

Theorem 4. *After* t *data words are encoded, the probability* $P(t)$ *that the dictionary will be unstable satisfies*

$$P(t)\leqslant\sum_{\substack{\mathbf{N}(t)\\ \text{stable}}}A(\mathbf{N}(t))P(\mathbf{N}(t))+\sum_{\substack{\mathbf{N}(t)\text{ not}\\ \text{stable}}}P(\mathbf{N}(t)).$$

Fig 7 shows the upper bound on $P(t)$ when we set $m=2$ and $P=\{0.4,0.3,0.2,0.1\}$. We see that $P(t)$ decreases rapidly as t increases. This indicates that if we make sure the first t data words are decoded correctly for large enough t, for example by combining shaping coding with error correction coding, we can significantly reduce the likelihood of future error propagation.

V. Conclusion

In this paper, we studied shaping codes to reduce programming wear when writing structured data to flash memory. We first reviewed so-called direct shaping codes for SLC flash memory. Using a page-oriented, programming cost model, we then extended this technique to MLC flash memory. The performance of these shaping codes was empirically evaluated on English-language text. Finally, we examined the error propagation behavior of direct shaping codes.

Acknowledgment

This work was supported in part by National Science Foundation (NSF) Grant CCF-1619053, the Center for Memory and Recording Research at UC San Diego, and Toshiba Corporation.

References

[1] A. Jagmohan, M. Franceschini, L. A. Lastras-Montao and J. Karidis, ”Adaptive endurance coding for NAND Flash,” *2010 IEEE Globecom Workshops*, Miami, FL, 2010, pp. 1841-1845.

[2] E. Sharon, et al.,”Data Shaping for Improving Endurance and Reliability in Sub-20nm NAND,” presented at *Flash Memory Summit*, Santa Clara, CA, Aug. 4–7, 2014.

[3] J. Li, K. Zhao, X. Zhang, J. Ma, M. Zhao, and T. Zhang. ”How much can data compressibility help to improve NAND flash memory lifetime?,” in *Proc. .13th USENIX Conf. File and Storage Technologies* (FAST’15), USENIX Assoc., Berkeley, CA, Feb. 16–19, 2015, pp. 227–240.

[4] T. Cover, ”Enumerative source encoding,” *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[5] P. G. Doyle and J. L. Snell, ”Random walks and electric networks,” 2000, arXiv:math/0001057

[6] D. S. Dummit, and R. M. Foote., *Abstract Algebra*. Englewood Cliffs: Prentice Hall, 1991.