Designing Floating Codes for Expected Performance

Hilary Finucane **Zhenming Liu*** **Michael Mitzenmacher***
School of Engineering and Applied Sciences
Harvard University

***Abstract*— Floating codes are codes designed to store multiple values in a Write Asymmetric Memory, with applications to flash memory. In this model, a memory consists of a block of n cells, with each cell in one of q states $\{0,1,\ldots,q-1\}$. The cells are used to represent k variable values from an ℓ-ary alphabet. Cells can move from lower values to higher values easily, but moving any cell from a higher value to a lower value requires first resetting the entire block to an all 0 state. Reset operations are to be avoided; generally a block can only experience a large but finite number of resets before wearing out entirely. A code here corresponds to mapping from cell states to variable values, and a transition function that gives how to rewrite cell states when a variable is changed.**

Previous work has focused on the developing codes that maximize the worst-case number of variable changes, or equivalently cell rewrites, that can be experienced before resetting. In this paper, we introduce the problem of maximizing the *expected* number of variable changes before resetting, given an underlying Markov chain that models variable changes. We demonstrate that codes designed for expected performance can differ substantially from optimal worst-case codes, and suggest constructions for some simple cases.

I. INTRODUCTION

A long-standing albeit not widely studied subfield of coding theory involves data storage in a setting where the stored information can change state in only limited ways. The seminal and perhaps canonical example is the write-once memory introduced by Rivest and Shamir [16]. Motivated primarily by the potential of digital optical disks, the authors consider the setting of write-once bit positions, which they dub *wits*. (The name, apparently, has not lasted the test of time.) Each wit initially contains a 0 that may subsequently be rewritten as a 1, but such a write is irreversible; a 1 cannot later be changed back to a 0. Paper punch cards provide a useful example of wits; once a position is punched, it cannot be reset.

If permanent storage is required, a medium of wits is quite useful, but if rewriting of information is required, this irreversibility is problematic. Rivest and Shamir therefore consider the question of how many wits are required to allow t rewrites of a k-bit value. By carefully choosing how wits can be used to represent values, they are able to design schemes that do significantly better than the naïve scheme using kt wits.

A variety of related models have been considered over the years; see, for example, [1], [3], [4], [5], [6], [7], [9], [10], [11], [12], [13], [14], [17], [18]. Questions regarding coding schemes of this type have resurfaced in recent years with the introduction of flash memories, which work under similar principles [12]. A flash memory utilizes floating-gate cells, which can be modeled as having q states $\{0,1,2,\ldots,q-1\}$. Roughly speaking, the states correspond to the number of electrons being held by the cell. Adding electrons to a cell is easy, but removing electrons from a cell is difficult. In our model, that means it is easy to move a cell from a lower-numbered state to a higher-numbered one, but not the other way around. Indeed, cells are generally organized into blocks, and in order to lower a state value within a cell, one must reset an entire block back to the all 0 state. Resetting blocks is considered very expensive, first because the rewriting time when resetting a block is large, but perhaps more importantly because the lifetime of a flash memory generally only allows a large but essentially fixed number of reset operations before the memory is no longer usable [2].

Previous work, including recent work on codes for flash memories, has focused on the problem of maximizing the number of rewrite operations before a reset operation is required *in the worst case*. Codes for this setting were dubbed floating codes [12]. Worst-case analysis is certainly a natural approach, particularly in settings where no resets are possible, and there is a need for fixed guarantees on rewrite performance in unknown environments. We suggest, however, that in

* Zhenming Liu and Michael Mitzenmacher are supported in part by NSF grant CCF-0634923. Contact author: michaelm@eecs.harvard.edu.

the setting of flash memories, where the product will be mass-produced, the product lifetime may allow a large number of reset operations, and there is the potential to study and model user behavior, *statistical* performance guarantees are more appropriate. The idea of average-case performance of codes of codes in this setting is not new; it is explicitly mentioned by Rivest and Shamir as an open question [16]. (Also, see different notions of average-case analysis in [1], [14].) As far as we know, however, our work actually initiates the study of the *expected* performance of floating codes.

Our primary contribution is a general model of the underlying problem of average-case performance of floating codes. Then, following the approach of [12], we consider particular possible implementations for specific parameter settings. At this point our work appears to raise more questions than it answers, leaving a variety of challenging open problems to consider.

II. A GENERAL MODEL FOR AVERAGE-CASE PERFORMANCE OF FLOATING CODES

We begin by reviewing the model for and definitions of floating codes given in [12]. The memory stores k variable values from an ℓ-ary alphabet, given by the variable vector $(v_1, v_2, \ldots, v_k)$ with $v_i \in \{0, 1, \ldots, \ell-1\}$. The memory consists of a block of n cells, represented by a cell state vector $(c_1, c_2, \ldots, c_n)$ with $c_i \in \{0, 1, \ldots, q-1\}$. A cell state vector $(c_1, c_2, \ldots, c_n)$ is said to be *above* a cell state vector $(d_1, d_2, \ldots, d_n)$ if $c_i \geq d_i$ for all i. Abusing notation, we well write $x \geq y$ if x and y are cell state vectors such that x is above y, and $x \not\geq y$ if x is not above y. (We avoid vector notation where the meaning is clear.) For a cell state vector x to change another state y with $y \not\geq x$, the memory must first be reset. If $y \geq x$, no reset is needed.

A floating code is defined by two functions. The decoding function $D : \{0, 1, \ldots, q-1\}^n \to \{0, 1, \ldots, \ell-1\}^k$ maps cell state vectors to variable vectors, and is used to decode the memory, transforming its current contents to the current variable values. The rewriting function $R : \{0, 1, \ldots, q-1\}^n \times \{1, \ldots, k\} \times \{0, 1, \ldots, \ell-1\} \to \{0, 1, \ldots, q-1\}^n$ gives information on how to transition when a single variable value changes; given a current cell state vector, a variable value to be changed, and the value that variable is changed to, the function R gives a new corresponding cell state vector. The restriction on R is that the current cell state vector must always decode via D to the current variable vector. Here when a rewrite causes a transition from x to y with $y \not\geq x$, the cost is 1 due to a reset; otherwise, the cost is 0.

The goal of [12] is to find decoding and rewriting functions that maximize (starting from the all-zero vector) the number of rewrites before a reset *in the worst case*. We here consider a different goal. We assume that there is a Markov chain with state space $\{0, 1, \ldots, \ell-1\}^k$ describing the behavior of the variable vector. For the most part we will follow [12] and assume that in only one variable changes at each time step, although more general Markov chains and rewriting functions can be considered. Given a decoding function D and rewriting function R, the Markov chain on the variable vector induces a corresponding Markov chain on the cell state vector. Let the equilibrium distribution of this chain be given by π_x, and let p_{xy} be the probability of transitioning to state vector y when in a state vector x. Then using standard theory, the average long-term cost per variable change is given by

$$A = \sum_{y \not\geq x} \pi_x p_{xy}. \tag{1}$$

Also, by standard renewal theory $1/A$ can be considered the long-term average time between reset operations. Our goal is to find functions D and R that minimize this cost A.

We emphasize that the above model can obviously be generalized in many directions, in manners similar to other proposed generalizations of simple write-only memories. We may have arbitrary alphabets $\mathscr{V}$ for the variables and $\mathscr{S}$ for the states, with an arbitrary stochastic process on $\mathscr{V}^k$ inducing corresponding processes on $\mathscr{S}^n$, given the rewriting transition function. The rewriting function could allow multiple variables to change at once, taking the form $R : \{0, 1, \ldots, q-1\}^n \times \{0, 1, \ldots, \ell-1\}^k \to \{0, 1, \ldots, q-1\}^n$, mapping the current cell state and a new value vector to a new cell state. There may be rules that a priori limit the transitions possible under the function R. There may be costs associated with all possible transitions (and/or all possible state vectors), and more general functions of these costs could be optimized. There may be history associated with either the variable state or the cell state, which could be incorporated into the decoding and rewriting functions. Many other generalizations can be imagined.

Indeed, we see the full generality of this coding problem as having potential applications beyond coding theory. Notice that, given the decoding function D, we can view the rewriting function R as a policy for a Markov decision process on the cell state vector, where the

possible actions at a state correspond to the collection of transitions to be made for each possible state change. Hence, underlying this problem is the question of how to design an underlying Markov decision process in the setting where we have an environment (the variable process) and we wish to minimize some function on the induced Markov decision process (the average cost per transition of the cell state process), where we have the ability to design the Markov decision process according to certain rules (in this case, the rules are that decoding is always successful). We are not aware of this specific problem in previous literature, and although we have no complexity results, we conjecture that many variations of this more general problem are at least NP-hard. (For a related NP-hardness result, see [5].)

III. THE CASE OF $k = 2$, $\ell = 2$

A. The case of $k = 2$, $\ell = 2$, $n = 2$

We begin with the case where $k = 2$, $\ell = 2$, and $n = 2$. That is, our value vector consists of two bits, and our states consist of two values in $\{0, \ldots, q-1\}$. Optimal codes under worst case analysis for these codes were described in [12]; they guarantee $(q-1) + \lfloor \frac{q-1}{2} \rfloor \approx \frac{3}{2}(q-1)$ transitions before a reset. Here we consider asymptotically optimal codes (as q grows large) for the average case, under the simple but natural model where the first of the two value bits is the next to flip with fixed probability p and the second is the next to flip with probability $1-p$ throughout the process. By asymptotically optimal, we mean that when $k = \ell = 2$ and n is fixed, the expected number of moves before a reset grows like $n(q-1) - o(q)$. (Indeed, we show this is the number of moves with high probability, with respect to the parameter q.) While it is perhaps not surprising that one could find an asymptotically optimal code for a given p, we show that in fact there exist codes that are simultaneously asymptotically optimal for every p, $0 < p < 1$.

Our code systems are based on *Gray codes* [8]. While we will give codes for all values of n, we begin with the important case of $n = 2$. We represent our code pictorially, for the case of $q = 4$, as follows (Figure 1).

The upper left hand corner represents the decoded value for the cell state $(0,0)$; it is written as 00, representing that both value bits are 0. More generally, the value in the ith row and jth column (counting from 0) represents the value vector under D for the cell state vector (i, j). Hence, for example, the decoded value for cell state $(2,1)$ is 10 (first bit 1, second bit 0). The rewriting function is implicit; from each cell, one moves to the closest available cell (generally one space *down* or to the *right*) that decodes to the proper new value, and a reset occurs only if no move is possible in the down and rightward direction. Note that on a reset, one starts the next cycle in one of four positions: $(0,0), (0,1), (1,0)$ or $(2,0)$.

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	00

Fig. 1. The code, or the 2DGC, for $k = 2$, $\ell = 2$, $n = 2$, $q = 4$. The upper left corner represents the decoded value vector for cell $(0,0)$; the lower right corner represents the decoded value vector for cell $(3,3)$. Transitions are greedy, to the closest available cell decoding to the appropriate value.

For larger values of q, we simply cycle through the Gray code values repeatedly. For example, the code when $q = 8$ is given in (Figure 2).

00	01	11	10	00	01	11	10
10	00	01	11	10	00	01	11
11	10	00	01	11	10	00	01
01	11	10	00	01	11	10	00
00	01	11	10	00	01	11	10
10	00	01	11	10	00	01	11
11	10	00	01	11	10	00	01
01	11	10	00	01	11	10	00

Fig. 2. Code for $k = 2$, $\ell = 2$, $n = 2$, $q = 8$.

We call these 2-dimensional Gray Codes, or a 2DGC for short. As we shall see, the 2DGC is not optimal, but it is asymptotically optimal, simultaneously for every value of p, in the following sense.

Theorem 1: For any p, the number of transitions before a reset for the 2DGC is $2(q-1) - o(q)$ with high probability (in q).

Proof: Let us say that a cell (x,y) is even if $x+y$ is even and odd if $x+y$ is odd. At any point away from the right boundary, from an even cell one move downs (respectively right) when the first bit (respectively second bit) flips, and vice versa for the odd cells. Consider the first $2q - 2q^{2/3}$ steps, which are necessarily split as evenly as possible between the even and odd cells; for convenience we say there are $q - q^{2/3}$ moves each at even and odd cells, as rounding and differences by 1 are absorbed in the $o(q)$ term. For the moves in even cells, the expected number of down moves is $p(q - q^{2/3})$, and similarly, for odd cells, the

expected number of down moves is $(1-p)(q-q^{2/3})$. As each move is an independent trial (as long as neither boundary is hit), it follows from standard Chernoff bounds [15][Theorem 4.4] that the number of down moves (and similarly right moves) is at most $q-3$ with high probability (larger than $1-1/\mathrm{poly}(q)$), and hence regardless of the starting position after the last reset with high probability the number of transitions before a reset is $2q-o(q)$. ■

It should be noted that our assumption regarding the underlying model is particularly important. If the two bits alternate flipping, then the number of transitions before a reset will be much less than $2q$, as a boundary will be reached in only q steps.

B. The case of $k=2$, $\ell=2$, $n>2$

To obtain similar results when $n>2$, we first note that for even values of n, we can simply successively glue together the $n=2$ result multiple times. That is, one first moves for $2(q-1)-o(q)$ transitions in the first two dimensions, then for $2(q-1)-o(q)$ transitions in the next two dimensions, and so on, to obtain a total of $n(q-1)-o(nq)$ transitions with high probability. (With care, one could obviously tighten the lower order term; we do not pursue this here.) It similarly suffices to demonstrate a code for the case $n=3$ that allows $3(q-1)+o(q)$ moves with high probability to obtain a similar result for odd n. To compare with worst-case results, we note that [12] shows optimal codes that ensure $(n-1)(q-1)+\lfloor\frac{q-1}{2}\rfloor$ transitions before a reset when $k=\ell=2$.

We again utilize a construction based on Gray codes, although some care must be taken to handle the third dimension properly. We again represent the code pictorially, with the value in the ith row and jth column from the kth square representing the value vector under D for the cell state vector (i,j,k). An example for $q=4$ is given in Figure 3, and for $q>4$ the code is again obtained by cycling through the Gray code values repeatedly in each two-dimensional square. Note that the cell state vectors for a given i and j are the same for all even values of k, and the same for all odd values of k. As an example, the decoded value for cell state $(2,1,1)$ is 11. This gives us a 3-dimensional Gray Codes (3DGC). For the 3DGC, there is some ambiguity, as a single transition of a value can lead to multiple cells of distance 1 from the current cell in the 3DGC. We resolve this ambiguity by having our rewriting function, from each cell, move to the closest available cell that decodes to the proper new value, with priority toward moving *up* in the third dimension. That is, from cell state (x,y,z), when possible our move increases the z value by 1. (Note *up* and *down* are not opposites here.)

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	00

10	00	01	11
11	10	00	01
01	11	10	00
00	01	11	10

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	00

10	00	01	11
11	10	00	01
01	11	10	00
00	01	11	10

Fig. 3. The code, or the 3DGC, for $k=2$, $\ell=2$, $n=3$, $q=4$. The upper left corner in the first square represents the decoded value vector for cell $(0,0,0)$; the lower right corner in the last square represents the decoded value vector for cell $(3,3,3)$. Transitions are greedy, breaking ties by increasing the z coordinate.

Theorem 2: For any p, the number of transitions before a reset for the 3DGC is $3(q-1)-o(q)$ with high probability (in q).

Proof: For a cell (x,y,z), we say that it is z-even if the z coordinate is even and that it is xy-even if the sum $x+y$ is even, and similarly for z-odd and xy-odd. With the given code, the moves can be classified according to four cell types (away from the boundaries):

- xy-even and z-even: move up with probability p and right with probability $1-p$;
- xy-even and z-odd: move up with probability p and down with probability $1-p$;
- xy-odd and z-even: move up with probability $1-p$ and right with probability p;
- xy-odd and z-odd: move up with probability $1-p$ and down with probability p.

We first note that, away from the top boundary $z=q-1$, the probability that we fail to move up in at least one of the next two moves it at most $p(1-p)\le 1/4$. It follows easily that, as long as we don't hit one of

the boundaries in the x or y dimension, the number of up moves after any reset over the next $3(q-1)-3q^{2/3}$ transitions will be $q-1$ with high probability for sufficiently large q.

We now want to show that the number of down and right moves until we reach $z=q-1$ are essentially split equally. This ensures we do not hit another boundary before $z=q-1$, and since when we reach $z=q-1$ we will be in the same setting as the 2DGC, it also implies that $2q-o(q)$ down and right moves are performed with high probability.

There is a minor complication in that the number of moves spent on a level $z=a$ before moving up to $z=a+1$ depends on whether we start at that level on an *xy*-even or *xy*-odd cell, which in turn depends on the value of a and the starting point after the last reset. However, the probability of starting at an *xy*-even cell after moving up quickly converges to it equilibrium value (which is p) after a small number of levels, and so we may ignore this complication, as the difference is absorbed into the $o(q)$ term. Given that we start at an *xy*-even cell, and are away from any of the boundaries, the probability that we stay at the same level z for k moves (always moving right when starting at a z-even cell, and down for a z-odd cell) is the same for every value of z, and is further dominated by a geometric distribution. The same holds for *xy*-odd cells. Standard applications of Chernoff bounds therefore again give with high probability the deviation between down and right moves over the first $3(q-1)-3q^{2/3}$ moves is $o(q)$ with high probability, and the result follows. ■

We emphasize that for any specific value of q, Theorem 2 does not state that the 3DGC is the optimal code. Indeed, Gray codes can be layered in different ways to obtain possible codes, and the parameters in any given circumstance might determine which performs best.

C. Calculations and comparisons

We now present some results based on calculating the performance of our codes in Table I. We compare three different codes, in the setting where $k=\ell=n=2$. First, we consider the 2-dimensional Gray Code (2DGC) previously described. We also consider a modified version, 2DGC+, described below. Finally, we consider the worst-case optimal code (2DWC), from [12]. The resulting average long-term costs, corresponding to equation (1), are obtained by running the appropriate Markov chain for $100,000,000$ steps. (While we could have instead determined the values by explicitly computing the equilibrium distribution for each chain, we found it useful to develop a simulator in the process of our work.) We vary p across a range of values, and present the corresponding costs (to four decimal places). We also similarly consider the 3-dimensional Gray Code (3DGC) and the worst-case optimal code for three dimensions (3DWC).

A simple but useful improvement on the 2DGC, especially when q is small, is to change the lower-right corner to 11 from 00, as in Figure 4. This change improves the average cost because if the process is at 00 on the lower-right corner, the next transition will cause a reset to the upper leftmost 01 or 10 position. However, the process would transition to the same position when starting from the 00 in the upper left corner. Hence, when the process reaches the cell 00 in the lower right hand corner, it saves nothing over resetting to the upper left corner, since the next move will lead to the same position. The cost is therefore reduced by instead using the cell to hold value pair 11, which can usefully prevent a reset for one further move on some occasions.

As can be seen from Table I, the 2DGC almost always performs better than the worst-case code, the exceptions being some cases where $q=4$. This is perhaps not surprising, in that the worst-case code was designed with a different consideration in mind. Our main point, however, it that it is interesting that a single code outperforms this code over the entire range of p. The 2DGC+ variation performs even slightly better than the 2DGC, although the gap quickly vanishes with q, as one would expect. The additive gap between the worst-case code and both the 2DGC and 2DGC+ also declines with q, but the multiplicative gap actually appears to increase. That is, assuming that the lifetime is dominated by the time between reset operations and that our model is suitable, the percentage increase in lifetime by using 2DGC or 2DGC+ as the underlying code is increasing with q.

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	11

Fig. 4. The 2DGC+. Changing the lower right corner gives a slight improvement.

Similar results are obtained with the 3DGC, also presented in Table I. Again, we see significant improvements over the worst case code across the entire range of p. Hence, while our results for these codes are asymptotic, and they may in fact not be optimal,

TABLE I

AVERAGE LONG-TERM COST (RESETS/MOVES) OF VARIOUS FLOATING CODES AS p VARIES.

Scheme	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
2DWC ($q=4$)	0.2088	0.2104	0.2134	0.2175	0.2222	0.2273	0.2324	0.2379	0.2438
2DGC ($q=4$)	0.2119	0.2146	0.2165	0.2176	0.2180	0.2175	0.2164	0.2146	0.2120
2DGC+ ($q=4$)	0.1763	0.1831	0.1874	0.1897	0.1905	0.1898	0.1874	0.1831	0.1763
2DWC ($q=8$)	0.0919	0.0926	0.0935	0.0944	0.0952	0.0962	0.0971	0.0980	0.1000
2DGC ($q=8$)	0.0797	0.0811	0.0820	0.0825	0.0827	0.0826	0.0820	0.0811	0.0797
2DGC+ ($q=8$)	0.0753	0.0771	0.0780	0.0785	0.0787	0.0786	0.0780	0.0771	0.0753
2DWC ($q=12$)	0.0592	0.0595	0.0599	0.0602	0.0606	0.0610	0.0613	0.0617	0.0625
2DGC ($q=12$)	0.0491	0.0499	0.0504	0.0506	0.0507	0.0506	0.0504	0.0499	0.0491
2DGC+ ($q=12$)	0.0476	0.0484	0.0489	0.0492	0.0492	0.0491	0.0489	0.0484	0.0476
3DWC ($q=4$)	0.1460	0.1466	0.1473	0.1480	0.1481	0.1479	0.1474	0.1466	0.1460
3DGC ($q=4$)	0.1287	0.1310	0.1326	0.1334	0.1333	0.1322	0.1300	0.1273	0.1243
3DWC ($q=8$)	0.0596	0.0601	0.0607	0.0611	0.0615	0.0619	0.0622	0.0625	0.0629
3DGC ($q=8$)	0.0514	0.0521	0.0526	0.0528	0.0528	0.0525	0.0521	0.0514	0.0505
3DWC ($q=12$)	0.0374	0.0378	0.0382	0.0385	0.0388	0.0391	0.0394	0.0397	0.0400
3DGC ($q=12$)	0.0321	0.0324	0.0327	0.0328	0.0328	0.0327	0.0325	0.0322	0.0317

the principle behind their design yields demonstrably better performance over worst-case codes given our assumptions.

Finally, while this feature may not matter dramatically in practice, we believe the simplicity of 2DGC+ and 3DGC are clear advantages.

IV. THE CASE OF $k=3$, $\ell=2$, $n=2$

000	100	101	001	011	111	110	010
010	000	100	101	001	011	111	110
110	010	000	100	101	001	011	111
111	110	010	000	100	101	001	011
011	111	110	010	000	100	101	001
001	011	111	110	010	000	100	101
101	001	011	111	110	010	000	100
100	101	001	011	111	110	010	000

Fig. 5. The basic building block for the 2DGC-3, where $k=3$ and $l=2$

While more general results for expected performance of floating codes rate to be more challenging, the use of Gray codes may be useful beyond the cases we have considered thus far. We here show how to utilize Gray codes to obtain a code that seems to perform well for the case of $k=3$, $\ell=2$, $n=2$. Our model for the three bits of data is that at each time step the first is the next to change with probability p_1, the second with probability p_2, and the third with probability p_3. We again consider the asymptotic behavior as q grows large, where our decoding function for $q=8$ is given by Figure 5 and for larger q is obtained by cycling in each dimension. Also, the transition function (described in more detail below) is the standard move to the nearest suitable cell. Notice that now, by necessity, we can not arrange to move only to an adjacent cell on a transition; with our Gray code design, some transitions require moving three cells down or to the right. For convenience, we refer to this code as 2DGC-3. By classifying these transitions, we obtain the following result:

Theorem 3: The number of transitions before a reset for the 2DGC-3 is $2(q-1)/(2-p_1)-o(q)$ with high probability (in q).

Proof: Upon examination, away from the boundaries, there are four types of cell states that for convenience we label as W, X, Y, and Z.

- W: move right 1 when first bit flips, down 1 when second bit flips, right 3 when third bit flips;
- X: move down 1 when first bit flips, down 3 when second bit flips, right 1 when third bit flips;
- Y: move right 1 when first bit flips, right 3 when second bit flips, down 1 when third bit flips;
- Z: move down 1 when first bit flips, right 1 when second bit flips, down 3 when third bit flips.

The 2DGC-3 code consists of repeated blocks of the form given in Figure 6.

W	X	Y	Z
Z	W	X	Y
Y	Z	W	X
X	Y	Z	W.

Fig. 6. An alternative view of the the basic building block for the 2DGC-3.

Consider W, X, Y, and Z as states of a Markov chain.

Then the Markov process on how value bits change induces the following Markov chain on these four states:

- W: moves to X with probability p_1, moves to Z with probability $(1-p_1)$;
- X: moves to W with probability p_1, moves to Y with probability $(1-p_1)$;
- Y: moves to Z with probability p_1, moves to X with probability $(1-p_1)$;
- Z: moves to Y with probability p_1, moves to W with probability $(1-p_1)$.

A straightforward analysis gives that, for any constant c, over the first $c(q-1)-q^{2/3}$ transitions after a reset, regardless of the initial starting point, with high probability $c(q-1)/2+o(q^{2/3})$ of the steps are spent in each of the states W,X,Y and Z as long as no boundary is reached. Following the same reasoning as in Theorem 1, the expected number of total spaces moved down or to the right in the first $c(q-1)-q^{2/3}$ transitions is $c(2-p_1)(q-1)/2-o(q)$ with high probability (in q). Choosing $c=2/(2-p_1)$, we obtain that we avoid the boundary and a reset for $2(q-1)/(2-p_1)-o(q)$ transitions with high probability. ■

Similar analyses can be performed for larger values of k; however, we do not have a proof that such codes are asymptotically optimal for $k>2$, as we do not at this point have an upper bound.

As with the standard 2DGC, one can possibly improve the performance slightly in practice by changing values along the boundary. For example, changing the lower right corner from 000 to 110 will give slightly better performance.

An interesting consideration brought on by this result is the idea that we could want multiple codes; here, we could have three similar but distinct codes, with the ith code taking $2(q-1)/(2-p_i)-o(q)$ transitions with high probability. We could then dynamically choose the code based on recent behavior, so that if the relative frequency of each bit flipping changes, our code could, after a reset, conceivably change with it.

V. CONCLUSIONS

We have argued for the study of average-case performance of floating codes, suggesting a general model and refining it in order to obtain some specific initial results. We have found simple, asymptotically optimal codes for storing two bit values under the model that the first bit is the next to flip with probability p. We have also found that Gray codes provide a potentially useful building block in the design of such codes, combining simplicity with strong performance. Our initial work suggests that designing floating codes for expected performance is a potentially feasible approach that could improve practical performance.

We conclude with a large number of open questions.

- Are there versions of the code optimization problem we have described that are NP-hard or harder?
- Can one find efficient approaches to find optimal floating codes for expected performance? Or approaches that are at least efficient for small parameters?
- Our Gray code constructions are vulnerable to patterned sequences of bit changes. Can we modify our codes to avoid this problem with little additional cost?
- Can we find expressions for lower bounds on the cost as a function of k,ℓ, n, and the Markov chain on the value variables? Can we find specific lower bounds for the Markov chain where the ith bit is the next to flip with probability p_i?
- Under what circumstances is there wasted space, in the sense that there are cell state vectors that are never reached in the optimal solution? Are there other possible utilizations for such cell states?
- What gains are possible considering small families of codes, instead of single codes, where one of the codes from the family can be chosen each time a reset occurs?
- Can we find constructions for larger values of the parameters k,ℓ, and n?
- What results can we obtained for expected performance in similar settings using error-correction [11] or buffer coding [3].

REFERENCES

[1] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Information and Computation*, pp. 80-97, 1989.

[2] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, 91(4):489-502, 2003.

[3] V. Bohossian, A. Jiang, and J. Bruck, "Buffer coding for asymmetric multi-level memory," in *Proc. IEEE International Symposium on Information Theory*, pp. 1186-1190, 2007.

[4] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for multi-level flash memories: correcting asymmetric limited-magnitude errros," in *Proc. IEEE International Symposium on Information Theory*, pp. 1176-1180, 2007.

[5] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *IEEE Transactions on Information Theory*, vol. IT-30, pp. 470-480, 1984.

[6] F. Fu and A. J. Han 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. 1, pp. 308-313, 1999.

[7] F. Fu and R. W. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Transactions on Information Theory*, vol. 46, no. 7, pp. 2299-2314, 2000.

[8] F. Gray. "Pulse code communications," U.S. Patent 2632058, March 1953.

[9] C. Heegard, "On the capacity of permanent memory," *IEEE Transactions on Information Theory*, vol. IT-31, pp. 34-42, 1985.

[10] C. Heegard and A. El Gamal, "On the capacity of computer memory with defects," *IEEE Transactions on Information Theory*, vol. IT-29, pp. 731-739, 1983.

[11] A. Jiang, "On the generalization of error-correcting WOM codes," in *Proc. IEEE International Symposium on Information Theory*, pp. 1391-1395, 2007.

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

[13] A. Jiang and J. Bruck, "Joint coding for flash memory storage," Technical Report (CaltechParadise, ETR087), 2008. To appear in *Proc. IEEE International Symposium on Information Theory*, 2008.

[14] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," Technical Report (CaltechParadise, ETR086), 2008. To appear in *Proc. IEEE International Symposium on Information Theory*, 2008.

[15] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, Cambridge, UK, 2005.

[16] R. L. Rivest and A. Shamir, "How to reuse a 'write-once" memory," *Information and Control*, vol. 55, pp. 227-231, 1984.

[17] G. Simonyi, "On write-unidirectional memory codes," *IEEE Transactions on Information Theory*, vol. 35, no.3, pp. 663-669, 1989.

[18] J. K. Wolf, A.D. Wyner, J. Ziv, and J. Korner, "Coding for a write-once memory,", *AT& T Bell Laboratories Technical Journal*, vol. 63, no. 6, pp. 1089-1112, 1984.