Optimized Cell Programming for Flash Memories

Anxiao (Andrew) Jiang and Hao Li
Computer Science and Engineering Department
Texas A&M University
College Station, TX 77843, USA
Email: {ajiang,hao}@cse.tamu.edu

***Abstract*—Flash memory cells use the charge they store to represent data. The amount of charge injected into a cell is called the cell's level. Programming a cell is the process of increasing a cell's level to the target value via charge injection, and the storage capacity of flash memories is limited by the precision of cell programming. To optimize the precision of the final cell level, a cell is programmed adaptively with multiple rounds of charge injection. Due to the high cost of block erasure, when cells are programmed, their levels are only allowed to increase. Such a storage medium can be modelled by a Write Asymmetric Memory model. It is interesting to study how well such storage media can be programmed.**

In this paper, we focus on the programming strategy that optimizes the expected precision. The performance criteria considered here include two metrics that are suitable for the multi-level cell technology and the rank modulation technology, respectively. Assuming that the charge-injection noise has a uniform random distribution, we present an effective algorithm for finding the optimal programming strategy. The optimal strategy can be used to program cells efficiently.

I. Introduction

Flash memories use floating-gate cells to store information. Charge (e.g., electrons) can be injected into a cell using the hot-electron injection mechanism or the Fowler-Nordheim tunneling mechanism, and the injected charge becomes trapped [3]. This is called *programming* or *writing* a cell. The amount of charge in a cell determines its threshold voltage, which is called its *level*. The more charge in a cell, the higher its level. The cell level can be lowered by removing the stored charge using the Fowler-Nordheim tunneling mechanism. However, in flash memories, cells are organized as blocks, each containing about 10^5 cells. To remove charge from any cell, the whole block needs to be *erased* (that is, all cells in the block have their charge removed), and then reprogrammed. Block erasures are very costly because they significantly reduce the longevity and efficiency of flash memories. Therefore, when a cell is being programmed, the cell level is only allowed to increase (because lowering the level would require the costly erasure operation) [3]. The charge injection process is noisy, so usually multiple rounds of charge injection are used to shift the cell level monotonically and cautiously toward the target level [1]. In current flash memories, two or more discrete cell levels are used to represent data. (Cells with two discrete levels are called single-level cells, or SLC. Cells with more than two levels are called multi-level cells, or MLC.) In the rank-modulation technology proposed recently [10], [11], [12], [14], instead of using discrete cell levels (of fixed values), the relative order of cell levels is used to represent data.

Since the cell levels monotonically increase during programming, flash memories are a type of Write Asymmetric Memory [5]. In [2], [4], [5], [6], [8], [9], [10], [13], [15], coding schemes are studied on how to modify data or correct errors by only increasing cell levels. It is clearly also interesting to study how to program cells accurately, as the precision of cell programming determines the storage capacity of flash memories. In [7], an optimal programming strategy is explored to achieve the zero-error capacity. The strategy considers the worst-case performance of cell programming.

In this paper, we focus on the cell programming strategy that optimizes the expected performance. The performance criteria considered here include two metrics that are suitable for the multi-level cell technology and the rank modulation technology, respectively. Knowing how well cells can be programmed on average is useful for studying the storage capacity of cell ensembles. We present an effective algorithm for finding the optimal programming strategy, which can in turn be used to program cells efficiently.

II. The Cell Programming Problem

Without loss of generality (w.l.o.g.), we assume that initially, the cell level is 0. A cell can be programmed using at most t rounds of charge injection. The objective is to make the final cell level be close to a target value $\theta \in [0, \mathcal{L}]$, where $\mathcal{L}$ is an upper bound determined by the physics of flash memories. There is a cost $C(x)$ associated with the final cell level x, and the function $C(x)$ monotonically increases with $|x - \theta|$. Two forms of $C(x)$ will be introduced later.

We assume that in each round of charge injection, the flash memory can choose the aimed increment of the cell level to be $i\Delta$ for some $i \in \{0, 1, 2, 3, \cdots\}$. Here Δ models the minimum resolution of the programming circuit. Let $\epsilon \in (0, 1)$ and $\delta > 0$ be two parameters. To model the noisy charge-injection process, we assume that if the aimed increment of the cell level is $i\Delta$, the actual increment of the cell level is randomly distributed in the range $[i\Delta(1-\epsilon), i\Delta(1+\delta))$.[1] For simplicity, in this paper, we assume the distribution is a uniform random distribution. More practical noise models can be studied in the future.

[1]The inclusion and exclusion of the boundary values are chosen for mathematical convenience, and are easy to deal with in practice.

978-1-4244-4561-5/09/$25.00 ©2009 IEEE

In this paper, we consider two families of cost functions.

Definition 1. (COST FUNCTION $C(x)$ FOR MLC AND RANK MODULATION) *In the multi-level cell (MLC) technology, the final cell level should be close to one of a set of discrete levels. It is appropriate to define $C(x)$ as*

$$C(x) = |x-\theta|^p$$

for some positive integer p.

In the rank modulation technology [10], [11], [12], the objective is usually to shift the cell level above *a certain value θ. It is appropriate to define $C(x)$ as*

$$C(x) = \begin{cases} \infty & , \text{ if } x < \theta \\ (x-\theta)^p & , \text{ if } x \geq \theta \end{cases}$$

for some positive integer p.

Let $i_1\Delta, i_2\Delta, \cdots, i_t\Delta$ denote the aimed increment of the cell level in the t rounds of charge injection, and let $x_1, x_2, \cdots, x_t$ denote the the actual cell level after each round. After the j-th round, the flash memory can measure x_j and adaptively choose the aimed level increment $i_{j+1}\Delta$ for the next round. The objective of the cell programming problem is to find the adaptive strategy of selecting $i_1, i_2, \cdots, i_t$ such that the expected cost of the final cell level, $E(C(x_t))$, is minimized. (Here $E(x)$ is the expectation of the random variable x.)

III. ADAPTIVE CELL PROGRAMMING

Given that the cell level is x_j after $j < t$ rounds of charge injection, how to choose the aimed level increment $i_{j+1}\Delta$ of the next round? We define two functions, $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$, for the computation.

Definition 2. (FUNCTIONS $\mathcal{A}(x;i)$ AND $\alpha(x;i;j)$)

$\mathcal{A}(x;i)$ is the minimum achievable expected cost of the final cell level given that (1) the current cell level is $\theta + x$, and (2) we can program the cell with i more rounds of charge injection.

$\alpha(x;i;j)$ is the minimum achievable expected cost of the final cell level given that (1) the current cell level is $\theta + x$, (2) we can program the cell with i more rounds of charge injection, and (3) in the first round of the i rounds of charge injection, we choose the aimed level increment to be $j\Delta$.

It is simple to see that $\mathcal{A}(x;i) = \min_{j=0,1,2\cdots} \alpha(x;i;j)$. For the cell programming problem, since the initial cell level is $0 = \theta + (-\theta)$ and t rounds of charge injection can be used, the objective is to find a strategy that makes the final cell level's expected cost be $\mathcal{A}(-\theta;t)$. During the programming process, given that the cell level is x_j after $j < t$ rounds of charge injection, the flash memory should adaptively choose $i_{j+1}\Delta$ as the aimed level increment of the $(j+1)$-th round such that i_{j+1} minimizes the value of $\alpha(x_j - \theta; t-j; i_{j+1})$.

The cost function we consider is for MLC or rank modulation, which is shown in Definition 1. Let us compute some initial values of $\mathcal{A}(x;i)$ – particularly, $\mathcal{A}(x;1)$ – for them.

A. When the cost function is for MLC

The cost function for MLC is $C(x) = |x-\theta|^p$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p = 2$. The other values of p can be dealt with similarly.

When $p = 2$, we have

$$\begin{array}{rll} & \mathcal{A}(x;1) & \\ = & \min_{j=0,1,2\cdots} & \alpha(x;1;j) \\ = & \min\{ & \alpha(x;1;0), \\ & \min_{j=1,2,3\cdots} & \int_{\theta+x+j\Delta(1-\epsilon)}^{\theta+x+j\Delta(1+\delta)} \frac{1}{j\Delta(\epsilon+\delta)} \cdot |y-\theta|^2 dy\} \\ = & \min\{ & x^2, \\ & \min_{j=1,2,3\cdots} & \frac{1}{j\Delta(\epsilon+\delta)} \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} y^2 dy\} \\ = & \min_{j=0,1,2\cdots} & x^2 + j\Delta(2+\delta-\epsilon)x + \frac{1}{3}j^2\Delta^2(3+ \\ & & 3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2) \end{array}$$

To see which value of j minimizes the above equation, define $f(j) = x^2 + j\Delta(2+\delta-\epsilon)x + \frac{1}{3}j^2\Delta^2(3+3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2)$. Since $0 < \epsilon < 1$ and $\delta > 0$, $3 + 3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2 > 0$, so $f(j)$ is convex. By setting $\frac{df(j)}{dj} = 0$, we find that $f(j)$ is minimized when $j = \frac{-3x(2+\delta-\epsilon)}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}$ (assuming that j does not have to be an integer). We can see that the above value for j is positive if and only if x is negative. Since j actually needs to be a non-negative integer, we find that to minimize $f(j)$, j should take the following value j^*:

$$j^* = \begin{cases} \lceil \frac{-3(2+\delta-\epsilon)x}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)} - 0.5 \rceil & , \text{ if } x < 0 \\ 0 & , \text{ if } x \geq 0 \end{cases}$$

Let $\gamma = \frac{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}{3(2+\delta-\epsilon)}$. Then when $x < 0$, $j^* = \lceil \frac{-x}{\gamma} - 0.5 \rceil$. So for $i = 1, 2, \cdots, \lceil \frac{\theta}{\gamma} - 1.5 \rceil$, when $x \in [-(i+0.5)\gamma, -(i-0.5)\gamma)$, we have $j^* = i$ and

$$\begin{array}{rl} \mathcal{A}(x;1) = & x^2 + i\Delta(2+\delta-\epsilon)x \\ & + \frac{1}{3}i^2\Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2). \end{array}$$

Similarly, when $x \in [-0.5\gamma, 0)$, we have $j^* = 0$ and $\mathcal{A}(x;1) = x^2$. When $x \in [-\theta, -(\lceil \frac{\theta}{\gamma} - 0.5 \rceil - 0.5)\gamma)$, we have $j^* = \lceil \frac{\theta}{\gamma} - 0.5 \rceil$ and $\mathcal{A}(x;1) = x^2 + \lceil \frac{\theta}{\gamma} - 0.5 \rceil \Delta(2+\delta-\epsilon)x + \frac{1}{3}\lceil \frac{\theta}{\gamma} - 0.5 \rceil^2 \Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)$. When $x \geq 0$, we have $j^* = 0$ and $\mathcal{A}(x;1) = x^2$. Therefore, we can partition the domain of x, $[-\theta, \infty)$, into $\lceil \frac{\theta}{\gamma} + 0.5 \rceil$ regions, while in each region $\mathcal{A}(x;1)$ is a degree-2 polynomial. So $\mathcal{A}(x;1)$ is piecewise polynomial.

It is not hard to see that when $p \neq 2$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity we skip the details.

B. When the cost function is for rank modulation

The cost function for rank modulation is $C(x) = \infty$ if $x < \theta$ and $C(x) = (x-\theta)^p$ if $x \geq \theta$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p = 1$. The other values of p can be dealt with similarly.

We have $\mathcal{A}(x;1) = \min_{j=0,1,2\cdots} \alpha(x;1;j)$. The value of j that minimizes $\alpha(x;1;j)$ is the minimum integer that satisfies

the constraint $\theta + x + j\Delta(1-\epsilon) \geq \theta$, which is $j = \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil$. So if $x < 0$, we have

$$\begin{aligned}\mathcal{A}(x;1) &= \int_{\theta+x+\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(1-\epsilon)}^{\theta+x+\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(1+\delta)} \frac{1}{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(\epsilon+\delta)} \cdot (y-\theta) dy \\ &= x + \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(1 + \frac{\delta-\epsilon}{2})\end{aligned}$$

If $x \geq 0$, clearly $\mathcal{A}(x;1) = x$.

So for $i = 1, 2, \cdots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil - 1$, when $x \in [-i\Delta(1-\epsilon), -(i-1)\Delta(1-\epsilon))$, $\mathcal{A}(x;1) = x + i\Delta(1 + \frac{\delta-\epsilon}{2})$. When $x \in [-\theta, -(\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil - 1)\Delta(1-\epsilon))$, $\mathcal{A}(x;1) = x + \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil \Delta(1 + \frac{\delta-\epsilon}{2})$. When $x \geq 0$, $\mathcal{A}(x;1) = x$. So we can partition the domain of x, $[-\theta, \infty)$, into $\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil + 1$ regions, while in each region, $\mathcal{A}(x;1)$ is a linear function. So $\mathcal{A}(x;1)$ is piecewise polynomial.

It is not hard to see that when $p \neq 1$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity we skip the details.

IV. Computing $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$

When $i \geq 2$, we have

$$\mathcal{A}(x;i) = \min_{j=0,1,2\cdots} \alpha(x;i;j)$$

and

$$\alpha(x;i;j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\delta+\epsilon)} dy$$

for $j \geq 1$. (We have $\alpha(x;i;0) = \mathcal{A}(x;i-1)$.)

It will be interesting to find an effective approach to compute the general functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ using the above recursion. In this paper, we present an efficient algorithm using the property that they are both piecewise polynomials. (Note that this property of being piecewise polynomial has been proved for $\mathcal{A}(x;1)$. It will be shown that it holds for $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ with $i \geq 2$, too.)

Let us define some notations. Given integers i, j, let $p_{i,j}$ and

$$b_{i,j}(-1)\ ,\ b_{i,j}(0)\ ,\ b_{i,j}(1)\ ,\ \cdots\ ,\ b_{i,j}(p_{i,j})$$

be the numbers with the following properties: (1) $b_{i,j}(-1) > b_{i,j}(0) > b_{i,j}(1) > b_{i,j}(2) > \cdots > b_{i,j}(p_{i,j})$; (2) $b_{i,j}(-1) = \infty$, $b_{i,j}(0) = 0$, $b_{i,j}(p_{i,j}) = -\theta$; (3) for $k = 0, 1, \cdots, p_{i,j}$, the function $\alpha(x;i;j)$ is a polynomial of x when $x \in [b_{i,j}(k), b_{i,j}(k-1))$.

Given an integer $i \geq 1$, let q_i and

$$B_i(-1)\ ,\ B_i(0)\ ,\ B_i(1)\ ,\ \cdots\ ,\ B_i(q_i)$$

be the numbers with the following properties: (1) $B_i(-1) > B_i(0) > B_i(1) > \cdots > B_i(q_i)$; (2) $B_i(-1) = \infty$, $B_i(0) = 0$, $B_i(q_i) = -\theta$; (3) for $k = 0, 1, \cdots, q_i$, the function $\mathcal{A}(x;i)$ is a polynomial of x when $x \in [B_i(k), B_i(k-1))$.

A. Computing $\alpha(x;i;j)$ with $i \geq 2$

We first show how to compute $\alpha(x;i;j)$ with $i \geq 2$.

Given a real number $x \in [-\theta, \infty)$ and an integer $i \geq 1$, we call the unique integer $j \in \{0, 1, \cdots, q_i\}$ such that

$$x \in [B_i(j)\ ,\ B_i(j-1))$$

the "(i)-*index of* x", and denote it by

$$index(i;x).$$

Note that $index(i;x)$ decreases as x increases. Let us use $\mathcal{I}(i;x;j)$ to denote the set of (i)-indices of the real numbers in the interval

$$[x + j\Delta(1-\epsilon)\ ,\ x + j\Delta(1+\delta)).$$

We get

$$\begin{aligned}\mathcal{I}(i;x;j) = \{\ & index(i; x + j\Delta(1-\epsilon)); \\ & index(i; x + j\Delta(1-\epsilon)) - 1; \\ & index(i; x + j\Delta(1-\epsilon)) - 2; \\ & \cdots \\ & \lim_{\nu \to 0^+} index(i; x + j\Delta(1+\delta) - \nu)\ \ \}\end{aligned}$$

The last element in the above set is a limit, because the interval $[x + j\Delta(1-\epsilon)\ ,\ x + j\Delta(1+\delta))$ does not contain the boundary value $x + j\Delta(1+\delta)$.

Let us define the set $S_{i,j}$ (for $i \geq 2$) as

$$\begin{aligned}S_{i,j} = \{\ & s \in (-\theta, 0) | \text{ either } s = B_{i-1}(k) - j\Delta(1-\epsilon) \\ & \text{for some } 0 \leq k \leq q_{i-1} - 1, \text{ or} \\ & s = B_{i-1}(k) - j\Delta(1+\delta) \\ & \text{for some } 0 \leq k \leq q_{i-1} - 1\}.\end{aligned}$$

Then we have the following lemma.

Lemma 3. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by*

$$s_1, s_2, \cdots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \cdots, |S_{i,j}| + 1$, for any two numbers x_1, x_2 in the interval (s_k, s_{k-1}),

$$\mathcal{I}(i-1; x_1; j) = \mathcal{I}(i-1; x_2; j).$$

Proof: W.l.o.g., assume that $x_1 < x_2$. We just need to prove that (1)

$$index(i-1; x_1 + j\Delta(1-\epsilon)) = index(i-1; x_2 + j\Delta(1-\epsilon))$$

and (2)

$$\begin{aligned}&\lim_{\nu \to 0^+} index(i-1; x_1 + j\Delta(1+\delta) - \nu) \\ =\ &\lim_{\nu \to 0^+} index(i-1; x_2 + j\Delta(1+\delta) - \nu).\end{aligned}$$

Let us prove condition (1) by contradiction. Assume that $index(i-1; x_1 + j\Delta(1-\epsilon)) \neq index(i-1; x_2 + j\Delta(1-\epsilon))$. Then there must be some $B_{i-1}(k')$ such that

$$x_1 + j\Delta(1-\epsilon) < B_{i-1}(k') \leq x_2 + j\Delta(1-\epsilon).$$

So

$$x_1 < B_{i-1}(k') - j\Delta(1-\epsilon) \leq x_2.$$

Since

$$B_{i-1}(k') - j\Delta(1-\epsilon) \in S_{i,j} = \{s_1, s_2, \cdots, s_{|S_{i,j}|}\},$$

x_1 and x_2 cannot be in the same interval (s_k, s_{k-1}). That is a contradiction. So $index(i-1; x_1 + j\Delta(1-\epsilon)) = index(i-1; x_2 + j\Delta(1-\epsilon))$.

Condition (2) can be proved similarly. For simplicity, we skip the details. ∎

Theorem 4. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by*

$$s_1, s_2, \cdots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \cdots, |S_{i,j}| + 1$, the function $\alpha(x; i; j)$ is a polynomial of x for $x \in (s_k, s_{k-1})$. Furthermore, it can be computed as follows. Let

$$u = \lim_{\nu \to 0^+} index(i-1; s_k + j\Delta(1-\epsilon) + \nu),$$

and let

$$v = \lim_{\nu \to 0^+} index(i-1; s_{k-1} + j\Delta(1+\delta) - \nu).$$

Then,

$$\begin{aligned}\alpha(x; i; j) = \quad & \int_{x+j\Delta(1-\epsilon)}^{B_{i-1}(u-1)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy + \\ & \sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy + \\ & \int_{B_{i-1}(v)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy\end{aligned}$$

Proof: We know that

$$\alpha(x; i; j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)} dy.$$

Since $x \in (s_k, s_{k-1})$, by Lemma 3, $index(i-1; x + j\Delta(1-\epsilon)) = \lim_{\nu\to 0^+} index(i-1; s_k + j\Delta(1-\epsilon) + \nu) = u$ and $\lim_{\nu\to 0^+} index(i-1; x + j\Delta(1+\delta) - \nu) = \lim_{\nu\to 0^+} index(i-1; s_{k-1} + j\Delta(1+\delta) - \nu) = v$. So in the above integration, we can partition the domain for y into smaller intervals, in each of which the function $\mathcal{A}(y; i-1)$ is a polynomial of y. So the way to compute $\alpha(x; i; j)$ in this theorem is correct.

$\mathcal{A}(y; i-1)$ is a polynomial of y for $y \in [B_{i-1}(u), B_{i-1}(u-1)) \supseteq [x + j\Delta(1-\epsilon), B_{i-1}(u-1))$ and for $y \in [B_{i-1}(v), B_{i-1}(v-1)) \supseteq [B_{i-1}(v), x + j\Delta(1+\delta))$. Also note that the value of $\sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy$ is independent of $x \in (s_k, s_{k-1})$. Since polynomials are closed under integration and summation, we get that $\alpha(x; i; j)$ is a polynomial of x for $x \in (s_k, s_{k-1})$. ∎

The above theorem shows that $\alpha(x; i; j)$ is an integration of $\mathcal{A}(x; i-1)$. It is easy to see that if $\mathcal{A}(x; i-1)$ is a piecewise polynomial of degree d, then $\alpha(x; i; j)$ is a piecewise polynomial of degree at most $d+1$. As we will see, $\mathcal{A}(x; i)$ is also a piecewise polynomial of degree at most $d+1$.

Corollary 5. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by $s_1, s_2, \cdots, s_{|S_{i,j}|}$ such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_{-1} = \infty$, $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 0, 1, 2, \cdots, |S_{i,j}| + 1$, the function $\alpha(x; i; j)$ is a polynomial of x for $x \in [s_k, s_{k-1})$.*

Proof: Since the integration of a finite function is a continuous function, we get $\alpha(s_k; i; j) = \lim_{\nu\to 0^+} \alpha(s_k + \nu; i; j)$. With Theorem 4, it is not hard to see that the conclusion holds. ∎

With the algorithm in Theorem 4, we can partition the domain of x, $[-\theta, \infty)$, into the intervals

$$[-\theta, s_{|S_{i,j}|}), [s_{|S_{i,j}|}, s_{|S_{i,j}|-1}), \cdots, [s_1, 0), [0, \infty)$$

and compute the polynomial $\alpha(x; i; j)$ for each interval. To simplify the future computation, if the polynomials in adjacent intervals happen to be the same, we merge them into one interval.

B. Computing $\mathcal{A}(x; i)$ with $i \geq 2$

In Section III, we have shown how to compute $\mathcal{A}(x; 1)$. We now show how to compute $\mathcal{A}(x; i)$ for $i \geq 2$.

It is easy to see that when $j \geq \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil$, $\alpha(x; i; j) \geq \alpha(x; i; \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil)$ (because setting the aimed level increment too high only increases the expected cost of the final cell level). So when $i \geq 2$, we have

$$\mathcal{A}(x; i) = \min_{j=0}^{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil} \alpha(x; i; j) \tag{1}$$

We first use the algorithm in Theorem 4 to compute the functions

$$\alpha(x; i; 0)\ ,\ \alpha(x; i; 1), \cdots, \alpha(x; i; \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil).$$

(Note that when $x \in [-\theta, 0)$, $\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.) Let $S_{i,j}$ be as defined before. And denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by

$$s_1^{i,j}, s_2^{i,j}, \cdots, s_{|S_{i,j}|}^{i,j}$$

such that $0 > s_1^{i,j} > s_2^{i,j} > \cdots > s_{|S_{i,j}|}^{i,j} > -\theta$. We know that $\alpha(x; i; j)$ is a polynomial of x for x in each of the following intervals

$$[-\theta, s_{|S_{i,j}|}^{i,j})\ ,\ [s_{|S_{i,j}|}^{i,j}\ ,\ s_{|S_{i,j}|-1}^{i,j})\ ,\ \cdots\ ,\ [s_1^{i,j}, 0)\ ,\ [0, \infty)$$

Given the integer $i \geq 2$, let us define the set P as

$$P = \bigcup_{j=0}^{\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil} \{s_1^{i,j}, s_2^{i,j}, \cdots, s_{|S_{i,j}|}^{i,j}\}$$

Let us alternatively denote the elements in P by

$$p_1, p_2, \cdots, p_{|P|}$$

such that

$$p_1 > p_2 > \cdots > p_{|P|}.$$

Also let $p_{-1} = \infty$, $p_0 = 0$ and $p_{|P|+1} = -\theta$. We naturally have the following conclusion.

Lemma 6. *For $k = 0, 1, 2, \cdots, |P|+1$, the function $\alpha(x; i; j)$ is a polynomial of x for $x \in [p_k, p_{k-1})$. (Here $i \geq 2$ and $0 \leq j \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.)*

With the above observation, we can easily compute the function $\mathcal{A}(x; i)$ for x in each interval $[p_k, p_{k-1})$, where $k = 0, 1, \cdots, |P|+1$. That is because by Equation 1, $\mathcal{A}(x; i)$ is the minimum of at most $\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil + 1$ known polynomials. The method of computation should be clear, so we skip its details. The only thing to note is that if these polynomials intersect, the interval $[p_k, p_{k-1})$ may need to be partitioned into more smaller intervals, such that in each smaller interval, $\mathcal{A}(x; i)$ is still a polynomial of x.

As before, after the above computation, if the polynomials for $\mathcal{A}(x; i)$ in adjacent intervals happen to be the same, we merge them into one interval for a more succinct representation.

V. Optimal Cell Programming Strategy

In this section, we describe the cell-programming strategy that minimizes the expected cost of the final cell level. Recall that at most t rounds of charge injection can be used for programming a cell. We use the algorithm described before to compute the functions $\mathcal{A}(x; i)$ for $i = 1, 2, \cdots, t$, and compute the functions $\alpha(x; i; j)$ for $i = 1, 2, \cdots, t$ and $j = 0, 1, 2, \ldots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$. These functions are then stored in the storage system, to be looked up during the actual cell-programming process.[2]

For $i = 1, 2, \cdots, t$, let x_i denote the actual cell level after the i-th round of charge injection. Let $x_0 = 0$ denote the initial cell level. The objective of cell programming is to minimize the expectation of $C(x_t)$. The optimal cell-programming strategy is as follows:

For $i = 0, 1, \cdots, t-1$, set the aimed level increment in the $(i+1)$-th round of charge injection to be $j^*\Delta$ such that

$$\alpha(x_i - \theta; t-i; j^*) = \mathcal{A}(x_i - \theta; t-i).$$

It should be noted that once the functions $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$ are stored, it is very efficient to look them up for the actual programming of cells. Let us now analyze the time complexity of computing these functions. For simplicity, we use the cost function $C(x) = (x-\theta)^2$ for the multi-level cell technology as an example, but the results can be easily extended for both general cost functions in Definition 1.

When $C(x) = (x-\theta)^2$, the function $\mathcal{A}(x; 1)$ is a degree-2 polynomial of x in $O(\frac{\theta}{\Delta\delta})$ intervals. By induction (for simplicity we only present the conclusion and skip the detailed analysis), for $i = 2, 3, \cdots, t$ and $j = 0, 1, \cdots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$, the function $\alpha(x; i; j)$ is a degree-$(i+1)$ polynomial of x in $O(\frac{1-\epsilon}{\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-2)} i!)$ intervals; for $i = 2, 3, \cdots, t$, the function $\mathcal{A}(x; i)$ is a degree-$(i+1)$ polynomial in $O(\frac{\theta}{\Delta\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-1)}(i+1)!)$ intervals. So the overall time complexity of computing all the functions is $O(\frac{1-\epsilon}{\delta}(\frac{2\mathcal{L}^3}{\Delta^3(1-\epsilon)^3})^t (t+1)!)$. So when the number of rounds of charge injection t is a constant, Δ is not arbitrarily small and ϵ is not arbitrarily close 1, the complexity is upper bounded by a polynomial of the parameters. We note that the above complexity is derived based on a very pessimistic analysis. The actual complexity is usually (significantly) lower.

[2]Since $\theta \leq \mathcal{L}$, in the above computation, we let $\theta = \mathcal{L}$. Functions $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$ computed this way can be used for any $\theta \leq \mathcal{L}$.

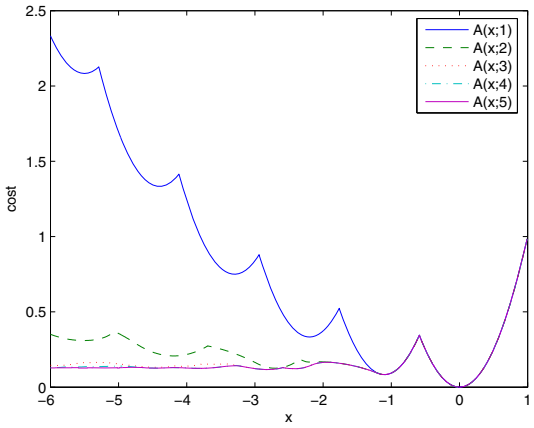

Fig. 1. The functions $\mathcal{A}(x; 1)$, $\mathcal{A}(x; 2)$, $\mathcal{A}(x; 3)$, $\mathcal{A}(x; 4)$ and $\mathcal{A}(x; 5)$. Here the cost function is for MLC, and $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

VI. Numerical Computation

We demonstrate the numerical computation of the functions $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$. We consider two cases for the cost function: for MLC, and for rank modulation (see Definition 1).

A. Multi-level Cells

For MLC, we set the cost function as

$$C(x) = (x-\theta)^2$$

and set the parameters as $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

The function $\mathcal{A}(x; i)$ is shown in Fig. 1, for $x \in [-6, 1)$ and $i = 1, 2, \cdots, 5$. We can see that $\mathcal{A}(x; i)$ is piecewise polynomial, and it monotonically decreases when i increases (because more rounds of charge injection leads to more accurate programming). We can also see that $\mathcal{A}(x; i)$ converges quickly as i increases.

$\mathcal{A}(x;3)$	
[-6,-5.56)	$23.9 + 11.2x + 1.76x^2 + 0.0917x^3$
[-5.56,-5.49)	$16.4 + 8.37x + 1.43x^2 + 0.0815x^3$
[-5.49,-5.39)	$24.9 + 12.7x + 2.16x^2 + 0.122x^3$
[-5.39,-4.76)	$14.3 + 8.76x + 1.8x^2 + 0.122x^3$
[-4.76,-4.18)	$7.66 + 4.6x + 0.924x^2 + 0.0611x^3$
[-4.18,-4.16)	$15.5 + 10.6x + 2.42x^2 + 0.183x^3$
[-4.16,-3.79)	$8.84 + 5.8x + 1.27x^2 + 0.0917x^3$
[-3.79,-3.77)	$0.948 + 1.63x + 0.722x^2 + 0.0917x^3$
[-3.77,-3.56)	$1.55 + 0.771x + 0.107x^2$
[-3.56,-3.42)	$-6.75 - 6.21x - 1.85x^2 - 0.183x^3$
[-3.42,-3.39)	$-1.22 - 0.743x - 0.1x^2 - 1.49e-08x^3$
[-3.39,-2.59)	$5.91 + 5.57x + 1.76x^2 + 0.183x^3$
[-2.59,-2.19)	$7.1 + 7.92x + 2.97x^2 + 0.367x^3$
[-2.19,-1.82)	$1.84 + 3.1x + 1.87x^2 + 0.367x^3$
[-1.82,-1.19)	$-0.259 - 0.413x - 0.1x^2$
[-1.19,-0.588)	$1.29 + 2.2x + x^2$
[-0.588,1)	x^2

Fig. 2. The function $\mathcal{A}(x; 3)$ for MLC.

$\alpha(x;3;3)$	
[-6,-5.99)	$-9.86-4.78x-0.761x^2-0.0407x^3$
[-5.99,-5.49)	$16.4+8.37x+1.43x^2+0.0815x^3$
[-5.49,-5.39)	$24.9+12.7x+2.16x^2+0.122x^3$
[-5.39,-4.76)	$14.3+8.76x+1.8x^2+0.122x^3$
[-4.76,-4.13)	$7.66+4.6x+0.924x^2+0.0611x^3$
[-4.13,-3.99)	$5.06+2.29x+0.267x^2-9.93e-09x^3$
[-3.99,-2.99)	$12.8+8.12x+1.73x^2+0.122x^3$
[-2.99,-2.39)	$9.55+4.85x+0.633x^2$
[-2.39,1)	$11.6+6.6x+x^2$

Fig. 3. The function $\alpha(x;3,3)$ for MLC.

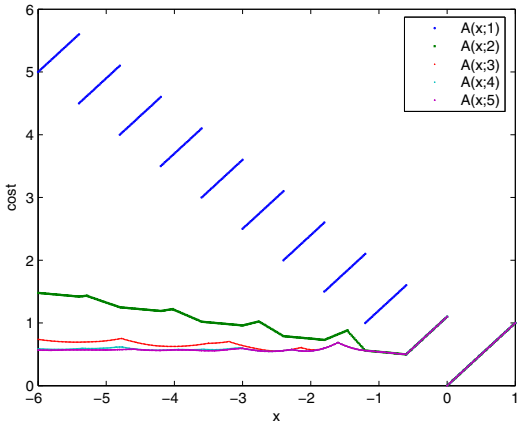

Fig. 4. The functions $\mathcal{A}(x;1)$, $\mathcal{A}(x;2)$, $\mathcal{A}(x;3)$, $\mathcal{A}(x;4)$ and $\mathcal{A}(x;5)$. Here the cost function is for rank modulation, and $\Delta=1,\epsilon=0.4,\delta=0.6$.

As an example, we show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Fig. 2 and Fig. 3, respectively, for $x\in[-6,1)$. The left column of the table shows the domain for x, and the right column shows the polynomial ($\mathcal{A}(x;3)$ or $\alpha(x;3;3)$) in this domain.

B. Rank Modulation

For rank modulation, we set the cost function as

$$C(x)=\begin{cases}\infty & \text{, if } x<\theta\\ x-\theta & \text{, if } x\geq\theta\end{cases}$$

and set the parameters as $\Delta=1,\epsilon=0.4,\delta=0.6$.

$\mathcal{A}(x;3)$	
[-6,-5.4)	$4.76+1.5x+0.138x^2$
[-5.4,-5.16)	$3.42+1x+0.0917x^2$
[-5.16,-4.8)	$6.47+2.18x+0.206x^2$
[-4.8,-4.77)	$4.89+1.52x+0.138x^2$
[-4.77,-4.56)	$2.04+0.853x+0.122x^2$
[-4.56,-4.2)	$5.22+2.25x+0.275x^2$
[-4.2,-3.6)	$3.6+1.48x+0.183x^2$
[-3.6,-3.5)	$2.41+0.817x+0.0917x^2$
[-3.5,-3.2)	$4.08+1.94x+0.275x^2$
[-3.2,-3)	$2.32+1.38x+0.275x^2$
[-3,-2.66)	$1.08+0.56x+0.138x^2$
[-2.66,-2.4)	$4.02+2.76x+0.55x^2$
[-2.4,-2.14)	$2.43+1.44x+0.275x^2$
[-2.14,-2.06)	$1.25+0.89x+0.275x^2$
[-2.06,-1.8)	$4.77+4.3x+1.1x^2$
[-1.8,-1.6)	$2.99+2.32x+0.55x^2$
[-1.6,-1.2)	$1.23+1.22x+0.55x^2$
[-1.2,-1.2)	$-0.88-1.2x$
[-1.2,-0.6)	$0.44-0.1x$
[-0.6,0)	$1.1+x$
[0,1)	x

Fig. 5. The function $\mathcal{A}(x;3)$ for rank modulation.

$\alpha(x;3;3)$	
[-6,-5.82)	$-1.99-0.76x-0.0458x^2$
[-5.82,-5.4)	$1.64+0.487x+0.0611x^2$
[-5.4,-4.8)	$5.21+1.81x+0.183x^2$
[-4.8,-4.56)	$2.04+0.853x+0.122x^2$
[-4.56,-4.2)	$5.22+2.25x+0.275x^2$
[-4.2,-3.6)	$3.6+1.48x+0.183x^2$
[-3.6,-3.26)	$2.41+0.817x+0.0917x^2$
[-3.26,-3)	$5.35+2.61x+0.367x^2$
[-3,-2.4)	$3.7+1.51x+0.183x^2$
[-2.4,-1.8)	$2.64+0.633x$
[-1.8,1)	$3.3+x$

Fig. 6. The function $\alpha(x;3,3)$ for rank modulation.

The function $\mathcal{A}(x;i)$ is shown in Fig. 4, for $x\in[-6,1)$ and $i=1,2,\cdots,5$. Again, we see that $\mathcal{A}(x;i)$ is piecewise polynomial, it monotonically decreases with i, and it converges quickly with i. For illustration, we also show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Fig. 5 and Fig. 6, respectively, for $x\in[-6,1)$.

VII. Conclusions

To learn and achieve the storage capacity of flash memories, it is necessary to understand how to program cells accurately. Based on the iterative and monotonic cell-programming method, a cell-programming strategy is presented in this paper that optimizes the expected performance.

Acknowledgment

This work was supported in part by the NSF CAREER Award CCF-0747415 and the NSF grant ECCS-0802107.

References

[1] 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 *Proc. IEEE International Symposium on Circuits and Systems*, 2005, pp. 2148-2151.

[2] V. Bohossian, A. Jiang and J. Bruck, "Buffer codes for asymmetric multi-level memory," in *Proc. IEEE ISIT*, 2007, pp. 1186-1190.

[3] P. Cappelletti, C. Golla, P. Olivo and E. Zanoni (*Ed.*), *Flash memories*, Kluwer Academic Publishers, 1st Edition, 1999.

[4] H. Finucane, Z. Liu and M. Mitzenmacher, "Designing floating codes for expected performance," in *Proc. 46th Annual Allerton Conference*, 2008.

[5] A. Jiang, V. Bohossian and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE Int. Symp. on Information Theory (ISIT)*, 2007, pp. 1166-1170.

[6] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE ISIT*, 2008, pp. 1741-1745.

[7] A. Jiang and J. Bruck, "On the capacity of flash memories," in *Proc. Int. Symp. Inform. Theory and Its Appl. (ISITA)*, 2008, pp. 94-99.

[8] A. Jiang, M. Langberg, M. Schwartz and J. Bruck, "Universal rewriting in constrained memories," to appear in *Proc. IEEE ISIT*, 2009.

[9] A. Jiang, H. Li and Y. Wang, "Error scrubbing codes for flash memories," in *Proc. Canadian Workshop on Inform. Theory (CWIT)*, 2009, pp. 32-35.

[10] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE ISIT*, 2008, pp. 1731-1735.

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

[12] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2008, pp. 1736-1740.

[13] R. L. Rivest and A. Shamir, "How to reuse a 'write-once' memory," *Information and Control*, vol. 55, pp. 1-19, 1982.

[14] Z. Wang, A. Jiang and J. Bruck, "On the capacity of bounded rank modulation for flash memories," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Seoul, Korea, June-July 2009.

[15] E. Yaakobi, A. Vardy, P. H. Siegel and J. K. Wolf, "Multidimensional flash codes," in *Proc. 46th Annual Allerton Conference*, 2008.