On The Parallel Programming of Flash Memory Cells

Eitan Yaakobi*, Anxiao (Andrew) Jiang†, Paul H. Siegel*, Alexander Vardy*, Jack K. Wolf*

*Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093, U.S.A.
{*eyaakobi, psiegel, avardy, jwolf*}*@ucsd.edu*

†Computer Science and Engineering
Texas A&M University
College Station, TX 77843, U.S.A.
ajiang@cse.tamu.edu

***Abstract*—Parallel programming is an important tool used in flash memories to achieve high write speed. In parallel programming, a common programm voltage is applied to many cells for simultaneous charge injection. This property significantly simplifies the complexity of the memory hardware, and is a constraint that limits the storage capacity of flash memories. Another important property is that cells have different hardness for charge injection. It makes the charge injected into cells differ even when the same program voltage is applied to them.**

In this paper, we study the parallel programming of flash memory cells, focusing on the above two properties. We present algorithms for parallel programming when there is information on the cells' hardness for charge injection, but there is no feedback information on cell levels during programming. We then proceed to the programming model with feedback information on cell levels, and study how well the information on the cells' hardness for charge injection can be obtained. The results can be useful for understanding the storage capacity of flash memories with parallel programming.

I. Introduction

Parallel programming is an important tool used in flash memories to achieve high write speed. Studying how cells are programmed is crucial for understanding the storage capacity of flash memories. The basic storage units in flash memories are floating-gate cells. The cells use the charge (e.g., electrons) stored in them to represent data, and the amount of charge stored in a cell is called the cell's *level*. A cell level can be increased or decreased by injecting charge into it or by removing charge from it, respectively, using the hot-electron injection mechanism or the Fowler-Nordheim tunnelling mechanism [1]. Flash memories have an important *asymmetric* feature when programming cells: while charge can be progressively injected into a cell, to remove charge from any cell, a large block of cells (about 10^6 cells) must be first erased together (i.e., all charge in them be fully removed) before reprogrammed. This is called *block erasure*, a very costly operation that substantially reduces the speed and longevity of flash memories [1]. To avoid block erasures, flash memory cells are usually programmed cautiously with multiple rounds of charge injection, so that every cell's level gradually approaches its target level [6], [8].

Parallel programming in flash memories has two important properties: *shared program voltages*, and *variation in charge-injection properties*. A flash memory injects charge into a cell by applying a program voltage to the cell. In parallel programming, a common program voltage is applied to many cells for simultaneous charge injection [1]. Compared to applying a separate program voltage to each cell, this property significantly simplifies the complexity of the memory hardware. It is also a constraint for the storage capacity of flash memories. Another important property is that cells have different hardness for charge injection [6], [8]. Some cells are *easy* to program, while others are *hard* to program. When the same program voltage is applied to cells, the easier-to-program cells will have more charge injected into them than the harder-to-program cells. This hardness is an intrinsic property of each cell. To accurately control the final cell levels, different program voltages should be applied to cells based on their hardness for programming. A widely used method in industry is the *Incremental Step Pulse Programming* (ISPP) scheme [6], [8], which gradually increases the program voltage to first program the easier cells and then the harder cells. To ensure that the (easier) cells are not overprogrammed, the subsequent program voltages are not applied to the cells whose levels are already sufficiently close to their target levels.

In previous works, the optimal programming of single flash memory cells has been studied. In [3], an optimal programming algorithm was presented for storing as much data as possible in a single cell, which achieves the zero-error storage capacity under its programming noise model. In [4], an algorithm was shown for optimizing the expected cell programming precision, when the programming noise follows a random distribution. Note that for programming a single cell, the two important properties for parallel programming – *shared program voltages* and *variation in programming hardness* – are not considered. So for parallel programming, new techniques need to be developed.

In this paper, we study the parallel programming of flash memory cells, focusing on the two new properties introduced above. We present polynomial-time algorithms that optimize the precision of the final cell levels, when there is information on the cells' programming hardness but there is no feedback information on cell levels during programming. We then proceed to the programming model with feedback information on cell levels, and study a related problem: how accurately can the cells' programming hardness be estimated through the feedback information on cell levels. The results can be useful for understanding the storage capacity of flash memories with parallel programming.

II. Terms and Concepts

Let $c_1, c_2, \ldots, c_n$ be n flash memory cells, whose initial *levels* (i.e., charge levels) are all 0. When they are programmed, a cell's level can only increase. We model the hardness of charge injection for the cells by the positive parameters $\alpha_1, \alpha_2, \ldots, \alpha_n$. Specifically, for $i \in [n] \triangleq \{1, 2, \ldots, n\}$, if a

978-1-4244-8264-1/10/$26.00 ©2010 IEEE

program voltage V is used to inject charge into the cell c_i, the level of c_i will increase by

$$\alpha_i V + \epsilon,$$

where ϵ is called the *programming noise* and is a random number with a probabilistic distribution. (The distribution of ϵ may depend on α_i and V.) In this paper, we will mostly consider the case $\epsilon = 0$ (which means the programming noise is sufficiently small), and focus on studying how α_i (i.e., the variance in charge-injection hardness) impacts parallel programming.

For $i \in [n]$, let $\theta_i \geqslant 0$ denote the target level of the cell c_i. The programming process consists of t rounds of charge injection, and the goal is to make the final level of c_i be very close to θ_i. To guarantee high write speed, t is usually a small constant (e.g., $t = 6$ or 8 for MLC flash). Let $V_1, V_2, \ldots, V_t$ denote the program voltages of the t rounds of programming, respectively. In each round, the memory can decide whether to apply the program voltage to a cell or not. For $i \in [n]$ and $j \in [t]$, let $b_{i,j}$ be a 0/1 integer such that in the j-th round of programming, if the program voltage V_j is applied to the cell c_i, then $b_{i,j} = 1$; otherwise, $b_{i,j} = 0$. For $i \in [n]$ and $j \in [t]$, let $\ell_{i,j}$ denote the level of c_i after the j-th round of programming. Then we have

$$\ell_{i,j} = \alpha_i \cdot (V_1, V_2, \cdots, V_j) \cdot (b_{i,1}, b_{i,2}, \cdots, b_{i,j})^T.$$

We measure the performance of programming by a cost function $\mathcal{C}(\Theta, L)$, where $\Theta \triangleq (\theta_1, \theta_2, \ldots, \theta_n)$ are the target levels and $L \triangleq (\ell_{1,t}, \ell_{2,t}, \ldots, \ell_{n,t})$ are the final cell levels. There are various ways to define $\mathcal{C}(\Theta, L)$. We will adopt the ℓ_p metric and denote it by $\mathcal{C}_p(\Theta, L)$:

$$\mathcal{C}_p(\Theta, L) \triangleq \left(\sum_{i=1}^{n} |\theta_i - \ell_{i,t}|^p \right)^{1/p}.$$

Given the integer p, our objective is to minimize $\mathcal{C}_p(\Theta, L)$.

In the parallel programming problem, Θ and t are given parameters. We can choose $V_1, \ldots, V_t$ and $b_{1,1}, \ldots, b_{n,t}$ for the best performance. It makes a difference whether we know the values of $\alpha_1, \ldots, \alpha_n$, and whether we can learn the cell levels after each round of programming (which is feedback information). If the feedback information on cell levels is available, the programming algorithm can be adaptive. Note that it is very time consuming to measure the exact cell levels. In practice, it is much faster to compare the cell levels with some preset threshold levels (using comparators), and use the information on cell levels to make decisions on programming [6], [8]. We show examples in the following.

Example 1. *Let $n = 8$, $\Theta = (1.0, 1.0, 2.0, 2.0, 1.0, 2.0, 2.0, 2.0)$, $t = 2$, and $(\alpha_1, \ldots, \alpha_8) = (0.5, 0.5, 0.8, 0.75, 0.5, 0.42, 0.85, 0.46)$ be known parameters. Assume there is no feedback information on the cell levels after each round of programming.*

Suppose we choose $V_1 = 2.0$ and $V_2 = 2.5$, and choose

$$\begin{pmatrix} b_{1,1} & b_{2,1} & \cdots & b_{8,1} \\ b_{1,2} & b_{2,2} & \cdots & b_{8,2} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Then we get $L = (1.0, 1.0, 2.0, 1.875, 1.0, 1.89, 2.125, 2.07)$, and $\mathcal{C}_p(\Theta, L) = (2 \times 0.125^p + 0.11^p + 0.07^p)^{1/p}$. If $p = 2$, then the programming performance is $\mathcal{C}_2(\Theta, L) = 0.220$.

Example 2. *Let $n \geqslant 1$, $\Theta = (1.0, 1.0, \ldots, 1.0)$, and $t = 2$. Let $\alpha_{\min}$ and $\alpha_{\max}$ be two known parameters such that $0 < \alpha_{\min} < \alpha_{\max}$. Suppose that the values of $\alpha_1, \cdots, \alpha_n$ are unknown; however, it is known that $\alpha_i \in [\alpha_{\min}, \alpha_{\max}]$ for $i \in [n]$. Suppose that we have feedback information on cell levels in the following way: after the first round of programming, we can compare all cells with a common preset threshold level τ to see if their levels are above or below τ. Based on this information, the memory can adaptively adjust the second round of programming. Suppose that we want to minimize $\mathcal{C}_\infty(\Theta, L)$.*

It can be verified that the following programming algorithm is optimal. Choose $V_1 = \frac{2}{\sqrt{\alpha_{\max}}(\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}})}$, $V_2 = \frac{2(\sqrt{\alpha_{\max}}-\sqrt{\alpha_{\min}})}{\sqrt{\alpha_{\max}\alpha_{\min}}(\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}})}$, and $\tau = \frac{2\sqrt{\alpha_{\min}}}{\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}}}$. After the first round of programming, all the cell levels are in the range $[\alpha_{\min}V_1, \alpha_{\max}V_1] = [\frac{2\alpha_{\min}}{\sqrt{\alpha_{\max}}(\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}})}, \frac{2\sqrt{\alpha_{\max}}}{\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}}}]$. (Note that $\tau < 1$, $\alpha_{\max}V_1 > 1$, and $1 - \tau = \alpha_{\max}V_1 - 1$.)

Let $\mathcal{S}$ denote the set of cells whose levels are less than τ after the first round of programming. (Note that for any cell $c_i \in \mathcal{S}$, we know now that $\alpha_i \in [\alpha_{\min}, \frac{\tau}{V_1})$.) Only the cells in S are programmed in the second round. After the second round, the levels of the cells in $\mathcal{S}$ are in the range $[\alpha_{\min}(V_1 + V_2), \frac{\tau}{V_1} \cdot (V_1 + V_2)) = [\tau, \alpha_{\max}V_1)$. So are the cells in $\{c_1, \ldots, c_n\} \setminus \mathcal{S}$. So we get $\mathcal{C}_\infty(\Theta, L) = 1 - \tau = \alpha_{\max}V_1 - 1 = \frac{\sqrt{\alpha_{\max}}-\sqrt{\alpha_{\min}}}{\sqrt{\alpha_{\max}}+\sqrt{\alpha_{\min}}}$.

III. Optimal Programming without Feedback

In this section, we present an optimal programming algorithm when the cells' hardness for programming – $\alpha_1, \alpha_2, \ldots, \alpha_n$ – is known. In this case, there is no need to obtain the feedback information on cell levels during programming, because they change deterministically. (In practice, $\alpha_1, \ldots, \alpha_n$ may be estimated values through some cell profiling process.) Let n, t, $\theta_1, \ldots, \theta_n$ and p also be known parameters. The programming algorithm needs to find $V_1, \ldots, V_t$ and $b_{1,1}, \ldots, b_{n,t}$ that minimize $\mathcal{C}_p(\Theta, L)$.

The parallel programming problem here is similar to the Subspace/Subset Selection Problem (SSP) [2] and the Sparse Approximate Solution Problem (SAS) [7], because we can see $\vec{p_i} \triangleq (b_{1,i}, b_{2,i}, \ldots, b_{n,i})^T$ (for $i = 1, \ldots, t$) as vectors, see $V_1, \ldots, V_t$ as real coefficients, and the objective is to make the linear combination $(\vec{p_1}, \ldots, \vec{p_t}) \cdot (V_1, \ldots, V_t)^T$ be close to a given vector $(\theta_1, \ldots, \theta_n)^T$. Both SSP and SAS problems are NP hard. However, we show here that the parallel programming problem has a polynomial-time solution (for many commonly used ℓ_p metrics, including $p = 1, 2, \infty$), using the condition that here $t = O(1)$ is a constant.

Without loss of generality (WLOG), we assume that $\frac{\theta_1}{\alpha_1} \leqslant \frac{\theta_2}{\alpha_2} \leqslant \cdots \leqslant \frac{\theta_n}{\alpha_n}$. (It takes time complexity $O(n \lg n)$ to sort the n ratios and label the n cells this way.)

Lemma 3. *There exists an optimal solution $V_1, \ldots, V_t, b_{1,1}, \ldots, b_{n,t}$ such that for any $1 \leqslant i < j \leqslant n$,*

$$(V_1, \ldots, V_t) \cdot (b_{i,1}, \ldots, b_{i,t})^T \leqslant (V_1, \ldots, V_t) \cdot (b_{j,1}, \ldots, b_{j,t})^T$$

Proof: For $k = 1, 2, \ldots, n$, let us define $y_k \triangleq (V_1, \ldots, V_t) \cdot (b_{k,1}, \ldots, b_{k,t})^T$. Note that for each $k \in [n]$, $\mathcal{C}_p(\Theta, L)$ is monotonically increasing in $|\theta_k - \ell_{k,t}| = |\theta_k - \alpha_k y_k|$. Consider any optimal solution where $y_i > y_j$. Since $|\theta_i - \ell_{i,t}|$ is minimized, we have $\frac{y_i + y_j}{2} \leqslant \frac{\theta_i}{\alpha_i}$. Since

$|\theta_j - \ell_{j,t}|$ is minimized, we have $\frac{\theta_j}{\alpha_j} \leqslant \frac{y_i+y_j}{2}$. Since $\frac{\theta_i}{\alpha_i} \leqslant \frac{\theta_j}{\alpha_j}$, we get $\frac{\theta_i}{\alpha_i} = \frac{y_i+y_j}{2} = \frac{\theta_j}{\alpha_j}$. So $|\theta_i - \alpha_i y_i| = |\theta_i - \alpha_i y_j|$. So we can make $(b_{i,1}, \ldots, b_{i,t})$ take the value of $(b_{j,1}, \ldots, b_{j,t})$ and still get an optimal solution. This way we can convert any optimal solution to an optimal solution that has the property shown in the lemma. ■

Let $B_1, B_2, \ldots, B_{2^t}$ denote the 2^t distinct binary vectors of length t, respectively. And let $\mathcal{S}_B \triangleq \{B_1, B_2, \ldots, B_{2^t}\}$. (Let $B_1, \ldots, B_{2^t}$ be column vectors, e.g., $(0, 0, \ldots, 0)^T$. Clearly, for $i = 1, \ldots, n$, $(b_{i,1}, b_{i,2}, \ldots, b_{i,t})^T \in \mathcal{S}_B$.) There are $2^t!$ permutations for the elements in $\mathcal{S}_B$, which we denote by $\pi_1, \pi_2, \ldots, \pi_{2^t!}$. For $i \in [2^t!]$, we denote the permutation $\pi_i(\mathcal{S}_B)$ by $\left(B_1^{\pi_i}, B_2^{\pi_i}, \ldots, B_{2^t}^{\pi_i}\right)$.

Define $\vec{V} \triangleq (V_1, V_2, \ldots, V_t)$. Given the program voltages $V_1, \ldots, V_t$, let $\mathcal{I}(\vec{V})$ be the integer in $[2^t!]$ such that

$$\vec{V} \cdot B_1^{\pi_{\mathcal{I}(\vec{V})}} \leqslant \vec{V} \cdot B_2^{\pi_{\mathcal{I}(\vec{V})}} \leqslant \cdots \leqslant \vec{V} \cdot B_{2^t}^{\pi_{\mathcal{I}(\vec{V})}}.$$

(If there is more than one such integer, $\mathcal{I}(\vec{V})$ can be any one of them.) The following result follows from Lemma 3.

Lemma 4. *There exists an optimal solution $V_1, \ldots, V_t, b_{1,1}, \ldots, b_{n,t}$ such that we can partition the set $\{1, 2, \ldots, n\}$ into 2^t subsets*

$$\{1, 2, \ldots, k_1\}, \{k_1+1, k_1+2, \ldots, k_2\}, \ldots, \{k_{i-1}+1, k_{i-1}+2, \ldots, k_i\}, \ldots, \{k_{2^t-1}+1, k_{2^t-1}+2, \ldots, n\}$$

with this property: $\forall\ i \in [2^t]$ and $j \in \{k_{i-1}+1, k_{i-1}+2, \ldots, k_i\}$, we have $(b_{j,1}, b_{j,2}, \ldots, b_{j,t})^T = B_i^{\mathcal{I}(\vec{V})}$. (Here we let $0 = k_0 \leqslant k_1 \leqslant k_2 \leqslant \cdots \leqslant k_{2^t} = n$.)

There are $\binom{n+2^t-1}{2^t-1}$ ways to choose the integers $k_1, k_2, \ldots, k_{2^t-1}$. (Note that if $k_{i-1} = k_i$ for some i, then the subset $\{k_{i-1}+1, k_{i-1}+2, \ldots, k_i\}$ is actually empty.) For $i = 1, 2, \ldots, \binom{n+2^t-1}{2^t-1}$, let $(k_1(i), k_2(i), \ldots, k_{2^t-1}(i))$ be the integers chosen for $(k_1, k_2, \ldots, k_{2^t-1})$ in the i-th way.

We present the parallel programming algorithm. The idea is to fix each permutation π_i and the partitioning integers $k_1, \ldots, k_{2^t-1}$, and search for the optimal solution.

Algorithm 5 PROGRAMMING WITHOUT FEEDBACK

Let $f_{opt} \leftarrow \infty$.
Let $\vec{V}_{opt} \leftarrow (-1, -1, \ldots, -1)$. ($\vec{V}_{opt}$ has length t.)
Let $b_{i,j}^{opt} \leftarrow -1$ for $i \in [n]$ and $j \in [t]$.
For $i = 1, 2, \ldots, 2^t!$
{ For $j = 1, 2, \ldots, \binom{n+2^t-1}{2^t-1}$
{ Find $V_1, V_2, \ldots, V_t$ that minimize the function $f \triangleq$

$$\sum_{d=1}^{2^t} \sum_{q=k_{d-1}(j)+1, k_{d-1}(j)+2, \ldots, k_d(j)} |\theta_q - \alpha_q \cdot (V_1, V_2, \ldots, V_t) \cdot B_d^{\pi_i}|^p$$

subject to the constraints that $V_1, V_2, \ldots, V_t \geqslant 0$.
Let $V_1^*, \ldots, V_t^*$ denote the optimal solution to the above optimization problem, and let f^* be the corresponding minimum value of the objective function f.
If $f^* < f_{opt}$, do:
{ $f_{opt} \leftarrow f^*$.
$\vec{V}_{opt} \leftarrow (V_1^*, \ldots, V_t^*)$.
For $d = 1, 2, \ldots, 2^t$
{ For $q = k_{d-1}(j)+1, k_{d-1}(j)+2, \ldots, k_d(j)$
{ $\left(b_{q,1}^{opt}, b_{q,2}^{opt}, \ldots, b_{q,t}^{opt}\right)^T \leftarrow B_d^{\pi_i}$.
} } } } }
Output the solution $\vec{V}_{opt}, b_{1,1}^{opt}, \ldots, b_{n,t}^{opt}$. (The corresponding minimized value of the cost function $\mathcal{C}_p(\Theta, L)$ is $f_{opt}^{1/p}$.

Theorem 6. *Algorithm 5 outputs an optimal solution to the parallel programming problem.*

Proof: In each iteration of Algorithm 5, $V_1^*, \ldots, V_t^*$ and the values $b_{1,1}, \ldots, b_{n,t}$ (which are specified by the vectors $B_d^{\pi_i}$ in the iteration) form a feasible solution to the parallel programming problem. On the other hand, an optimal solution to the parallel programming problem that has the properties in Lemma 3 and Lemma 4 must be found by Algorithm 5. ■

It can be seen that when $p = 1$, 2, and ∞ (which are the most commonly used metrics for $\mathcal{C}_p(\Theta, L)$), the time complexity of Algorithm 5 is polynomial in n. (Note that t is a small constant.) For example, consider $p = 2$. In each iteration of Algorithm 5, the optimization problem is a quadratic programming problem with $t = O(1)$ non-negativity constraints for variables, which can be solved with time complexity of $O(n)$ [5]. There are $2^t!\binom{n+2^t-1}{2^t-1} = O(n^{2^t-1})$ iterations. So Algorithm 5 has time complexity $O(n^{2^t})$.

Note that we can further optimize the complexity of Algorithm 5 by using the fact that not all the $2^t!$ permutations π_i need to be checked. For simplicity, we skip the details.

We conclude this section by studying a simplified yet useful case of parallel programming. In this case, all n cells have the same target cell level, that is, $\theta_1 = \cdots = \theta_n = \theta$. (For multi-level cells (MLC), it is a natural heuristic solution to program cells of the same target level together, which corresponds to this case.) We consider programming noise ϵ; that is, if the program voltage V is applied to a cell c_i, its charge level will increase by $\alpha_i V + \epsilon_i(V)$, where the programming noise $\epsilon_i(V)$ is a random number related to V. The values of $\alpha_1, \ldots, \alpha_n$ are unknown, but we see them as i.i.d. random variables with known expectation $\mu_1(\alpha)$ and second moment $\mu_2(\alpha)$. (That is, if we use $E(x)$ to denote the expectation of a random variable x, then for $i \in [n]$, $E(\alpha_i) = \mu_1(\alpha)$ and $E(\alpha_i^2) = \mu_2(\alpha)$. Note that the statistics $\mu_1(\alpha)$ and $\mu_2(\alpha)$ are easy to obtain by the memory by testing many cells with the values of the injected and trapped charges in the cells.) We also assume that the programming noise $\epsilon_1(V), \ldots, \epsilon_n(V)$ are i.i.d. random variables; and for $i \in [n]$, we assume $E(\epsilon_i(V)) = 0$ (namely, its expectation is zero) and $E\left((\epsilon_i(V))^2\right) = \sigma V^2$ (namely, its standard deviation is proportional to the program voltage V). We can use t rounds of programming. We consider the ℓ_2 metric for the cost function and define it as

$$\mathcal{C} \triangleq \sum_{i=1}^{n} (\theta - \ell_{i,t})^2.$$

And our objective is to minimize $E(\mathcal{C})$, that is, the expected cost. We call this case *uniform programming*.

Clearly, the optimal solution is to apply the same set of program voltages $V_1, \ldots, V_t$ to all the n cells. The following theorem presents the optimal solution and the optimal cost.

Theorem 7. *For the uniform programming problem, the optimal solution is to set*

$$V_1 = \cdots = V_t = \frac{\theta \mu_1(\alpha)}{t\mu_2(\alpha) + \sigma}$$

and $b_{i,j} = 1$ *for* $i \in [n]$ *and* $j \in [t]$. *The corresponding minimized value of the expected cost* $E(\mathcal{C})$ *is*

$$n\left(1 - \frac{\mu_1(\alpha)^2}{\mu_2(\alpha) + \frac{\sigma}{t}}\right)\theta^2.$$

Proof: It can be seen that in the optimal solution, $\ell_{1,t}, \ell_{2,t}, \ldots, \ell_{n,t}$ are i.i.d. random variables. So $E(\mathcal{C}) = nE\left((\theta - \ell_{1,t})^2\right)$. For $i \in [t]$, let $\epsilon_{1,i}$ denote the programming noise for cell c_1 in the i-th round of programming. Then

$$\begin{aligned} &E(\mathcal{C})/n = E\left(\left(\theta - \textstyle\sum_{i=1}^{t}(\alpha_1 V_i + \epsilon_{1,i})\right)^2\right) \\ &= \theta^2 - 2\theta E\left(\textstyle\sum_{i=1}^{t}(\alpha_1 V_i + \epsilon_{1,i})\right) + E\left(\left(\textstyle\sum_{i=1}^{t}(\alpha_1 V_i + \epsilon_{1,i})\right)^2\right) \\ &= \theta^2 - 2\theta\mu_1(\alpha)\textstyle\sum_{i=1}^{t} V_i + \mu_2(\alpha)\left(\textstyle\sum_{i=1}^{t} V_i\right)^2 + \sigma\textstyle\sum_{i=1}^{t} V_i^2 \end{aligned}$$

Note that $\sum_{i=1}^{t} V_i^2 \geqslant \frac{1}{t}\left(\sum_{i=1}^{t} V_i\right)^2$. So we get $E(\mathcal{C})/n \geqslant \left(\mu_2(\alpha) + \frac{\sigma}{t}\right)\left(\sum_{i=1}^{t} V_i\right)^2 - 2\theta\mu_1(\alpha)\sum_{i=1}^{t} V_i + \theta^2 \geqslant \left(1 - \frac{\mu_1(\alpha)^2}{\mu_2(\alpha)+\frac{\sigma}{t}}\right)\theta^2$. The above inequalities will become equalities if and only if $\sum_{i=1}^{t} V_i = \frac{\theta\mu_1(\alpha)}{\mu_2(\alpha)+\frac{\sigma}{t}}$ and $V_1 = V_2 = \cdots = V_t$. ∎

IV. Feedback Information on Cell Levels

In this section, we extend the study to parallel programming with feedback information on cell levels. That is, the memory can measure the cell levels after every round of charge injection. Note that in practice, it is very time consuming to measure the exact level of every cell. It is much faster to compare the cell levels (in parallel using comparators) with one or more preset threshold levels, to see if the cell levels are above or below the threshold level [1], [6], [8]. This is the scheme we consider here. By obtaining this feedback information on cell levels, the memory can learn more about the cells' hardness for charge injection, and adaptively choose the subsequent program voltages for optimal performance.

Let $\alpha_{\min}$ and $\alpha_{\max}$ be two known parameters, where $0 < \alpha_{\min} < \alpha_{\max}$. We assume that initially (i.e., before programming starts), the only knowledge on cells' hardness for charge injection is that for $i \in [n]$, $\alpha_i \in [\alpha_{\min}, \alpha_{\max}]$. After every round of programming, based on the feedback information on cell levels, the memory can estimate the values of the α_i's better, and adaptively optimize the following program voltages. Therefore, the programming algorithm is a combination of two iterative processes: *iteratively obtaining information on the cells' hardness for programming (i.e., the values of α_i's)*, and *adaptively optimizing the program voltages (i.e., $V_1, \ldots, V_t$) and the on/off of cells (i.e., $b_{1,1}, \ldots, b_{n,t}$).*

In this section, we focus on the first iterative process, and study this related problem: How much information can be learned about a cell's hardness for charge injection, α, through t rounds of programming? Specifically, let $[\alpha'_{\min}, \alpha'_{\max}] \subseteq [\alpha_{\min}, \alpha_{\max}]$ denote the range we can narrow down to such that after t rounds of programming, we can learn for sure that $\alpha \in [\alpha'_{\min}, \alpha'_{\max}]$. The smaller $\alpha'_{\max} - \alpha'_{\min}$ is, the better.

We assume that after every round of programming, we can compare a cell with one preset threshold level. We formally formulate the problem as follows. Let c be a cell whose initial level is 0. Let α denote the cell's hardness for charge injection, such that when a program voltage V is applied, the cell's level will increase by αV. Initially, the only knowledge about α is that $\alpha \in [\alpha_{\min}, \alpha_{\max}]$ for some known parameters $\alpha_{\min}, \alpha_{\max}$. We can use up to t rounds of programming (whose program voltages are denoted by $V_1, \ldots, V_t$), and choose in advance r threshold levels $\tau_1, \tau_2, \ldots, \tau_r$. For $i \in [t]$, let ℓ_i denote the cell's level after the i-th round of programming, and let $[\alpha_i^{\min}, \alpha_i^{\max}]$ denote the range such that after the i-th round of programming, we know for sure that $\alpha \in [\alpha_i^{\min}, \alpha_i^{\max}]$. (By convention, let $\ell_0 = 0$ and $[\alpha_0^{\min}, \alpha_0^{\max}] = [\alpha_{\min}, \alpha_{\max}]$. Clearly, when $0 \leqslant i < j \leqslant t$, $[\alpha_i^{\min}, \alpha_i^{\max}] \supseteq [\alpha_j^{\min}, \alpha_j^{\max}]$.) For $i \in [t]$, after the i-th round of programming, we can compare ℓ_i with one threshold level – say τ_j – to see if $\ell_i < \tau_j$ or $\ell_i \geqslant \tau_j$, compute $\alpha_i^{\min}$ and $\alpha_i^{\max}$, and adaptively choose the next program voltage V_{i+1}. Our objective is to choose $\tau_1, \ldots, \tau_r$ in advance, choose $V_1, \ldots, V_t$ online, such that $\alpha_t^{\max} - \alpha_t^{\min}$ is minimized (in the worst case).

Let $\Delta(t, r, \alpha_{\min}, \alpha_{\max})$ denote the smallest achievable value of $\alpha_t^{\max} - \alpha_t^{\min}$ over all possible solutions. Intuitively, $\Delta(t, r, \alpha_{\min}, \alpha_{\max}) \geqslant (\alpha_{\max} - \alpha_{\min})/2^t$. (Every comparison with a threshold level can reduce the interval size by at most half.) We now present a better bound. Given $i \geqslant j$, define $\left(\!\binom{i}{j}\!\right) \triangleq \sum_{k=0}^{j}\binom{i}{k}$. For $i < j$, define $\left(\!\binom{i}{j}\!\right) \triangleq 2^i$. Note that $\left(\!\binom{i}{j}\!\right) = \left(\!\binom{i-1}{j}\!\right) + \left(\!\binom{i-1}{j-1}\!\right)$.

Theorem 8. $\Delta(t, r, \alpha_{\min}, \alpha_{\max}) \geqslant (\alpha_{\max} - \alpha_{\min}) / \left(\!\binom{t}{r}\!\right)$.

Proof: We present the sketch of proof. As base cases, $\Delta(t, 0, \alpha_{\min}, \alpha_{\max}) = \Delta(0, r, \alpha_{\min}, \alpha_{\max}) = \alpha_{\max} - \alpha_{\min}$. Now consider $t, r \geqslant 1$, and we use induction on t, r. After one round of programming, we compare the cell level ℓ with some threshold level τ; then for some $\alpha' \in [\alpha_{\min}, \alpha_{\max}]$ we can determine if $\alpha \geqslant \alpha'$ (because $\ell \geqslant \tau$) or $\alpha < \alpha'$ (because $\ell < \tau$). There are $t-1$ more rounds; and if the cell level ℓ has exceeded τ, there are at most $r-1$ threshold levels left to compare with. (Note that the cell level can only increase.) So by induction, we get $\Delta(t, r, \alpha_{\min}, \alpha_{\max}) = \min_{\alpha'} \max\{\Delta(t-1, r-1, \alpha', \alpha_{\max}), \Delta(t-1, r, \alpha_{\min}, \alpha')\} \geqslant \min_{\alpha'} \max\{\frac{\alpha_{\max}-\alpha'}{\left(\!\binom{t-1}{r-1}\!\right)}, \frac{\alpha'-\alpha_{\min}}{\left(\!\binom{t-1}{r}\!\right)}\} \geqslant \min_{\alpha'} \frac{(\alpha_{\max}-\alpha')+(\alpha'-\alpha_{\min})}{\left(\!\binom{t-1}{r-1}\!\right)+\left(\!\binom{t-1}{r}\!\right)} = \frac{\alpha_{\max}-\alpha_{\min}}{\left(\!\binom{t}{r}\!\right)}$. ∎

We now present an optimal algorithm whose performance matches the above bound. It has $r \leqslant t$. (Having $r > t$ does not help.) In the algorithm, for $i = 1, \ldots, r$, let $\tau_i = \tau_1\left(\frac{\alpha_{\max}}{\alpha_{\min}}\right)^{i-1}$. (The value of $\tau_1 > 0$ can be arbitrary. By convention, let $\tau_0 \triangleq 0$.) Let $z_1, \ldots, z_{t+1}$ and *flag* be integer parameters, and set $z_1 = r$ and *flag* $= 1$. Then, for $i = 1, 2, \ldots, t$, the i-th round of programming (and its following computation) is performed as follows:

1) If $z_i = 0$, then $\alpha_i^{\min} \leftarrow \alpha_{i-1}^{\min}$, $\alpha_i^{\max} \leftarrow \alpha_{i-1}^{\max}$, $z_{i+1} \leftarrow 0$

and skip the next three steps. (In this case, we see V_i as $V_i = 0$; namely, the cell is not really programmed.)

2) $\alpha'_i \leftarrow \frac{\alpha_{i-1}^{\min}\left(\binom{t-i}{z_i-1}\right)+\alpha_{i-1}^{\max}\left(\binom{t-i}{z_i}\right)}{\left(\binom{t-i}{z_i-1}\right)+\left(\binom{t-i}{z_i}\right)}$.

3) If *flag* $= 1$, then $V_i \leftarrow \frac{\tau_{r-z_i+1}}{\alpha'_i} - \frac{\tau_{r-z_i}}{\alpha_{i-1}^{\min}}$;
If *flag* $= 0$, then $V_i \leftarrow \frac{\tau_{r-z_i+1}}{\alpha'_i} - \frac{\tau_{r-z_i+1}}{\alpha_{i-1}^{\max}}$.
Program the cell with program voltage V_i.

4) Compare cell level ℓ_i with threshold level τ_{r-z_i+1}.
If $\ell_i < \tau_{r-z_i+1}$, then $\alpha_i^{\min} \leftarrow \alpha_{i-1}^{\min}$, $\alpha_i^{\max} \leftarrow \alpha'_i$, $flag \leftarrow 0$, $z_{i+1} \leftarrow z_i$.
If $\ell_i \geqslant \tau_{r-z_i+1}$, then $\alpha_i^{\min} \leftarrow \alpha'_i$, $\alpha_i^{\max} \leftarrow \alpha_{i-1}^{\max}$, $z_{i+1} \leftarrow z_i - 1$.

The next lemma shows that the program voltages are all non-negative, which means the algorithm can be successfully implemented.

Lemma 9. *In the algorithm, for $i = 1, \ldots, t$, $V_i \geqslant 0$.*

Proof: $V_1 = \frac{\alpha_{\min}\left(\binom{t-1}{z_1-1}\right)+\alpha_{\max}\left(\binom{t-1}{z_1}\right)}{\left(\binom{t-1}{z_1-1}\right)+\left(\binom{t-1}{z_1}\right)} > 0$. Now consider $i \geqslant 2$ and $z_i > 0$. If $flag = 1$, then $V_i = \frac{\tau_{r-z_i+1}}{\alpha'_i} - \frac{\tau_{r-z_i}}{\alpha_{i-1}^{\min}} = \frac{\alpha_{\max}}{\alpha_{\min}} \cdot \frac{\tau_{r-z_i}}{\alpha'_i} - \frac{\tau_{r-z_i}}{\alpha_{i-1}^{\min}} = \tau_{r-z_i}\left(\frac{\alpha_{\max}}{\alpha_{\min}} \cdot \frac{1}{\alpha'_i} - \frac{1}{\alpha_{i-1}^{\min}}\right)$. Since $\alpha_{\min} \leqslant \alpha_{i-1}^{\min}$ and $\alpha'_i \leqslant \alpha_{\max}$, we get $V_i \geqslant 0$. If $flag = 0$, then $V_i = \frac{\tau_{r-z_i+1}}{\alpha'_i} - \frac{\tau_{r-z_i+1}}{\alpha_{i-1}^{\max}} = \tau_{r-z_i+1} \cdot \left(\frac{1}{\alpha'_i} - \frac{1}{\alpha_{i-1}^{\max}}\right) \geqslant 0$. ∎

The following lemma can be proved by induction. Due to space limitation, we skip the details of the proof.

Lemma 10. *In the algorithm, for $i = 1, \ldots, t$, we have $\sum_{j=1}^{i} V_j = (\tau_{r-z_i+1})/\alpha'_i$ when $z_i > 0$. (If $z_i = 0$, $V_i = 0$.)*

The next theorem shows the performance of the algorithm.

Theorem 11. *The algorithm outputs $\alpha_t^{\min}, \alpha_t^{\max}$ such that*

$$\alpha_t^{\max} - \alpha_t^{\min} = \frac{\alpha_{\max} - \alpha_{\min}}{\left(\binom{t}{r}\right)}.$$

Proof: We present the sketch of the proof. Let k be the smallest integer in $\{1, 2, \ldots, t-1\}$ such that $z_{k+1} = 0$ in the algorithm; if $z_i > 0$ for $i \in [t-1]$, then let $k = t$. The k-th round is the final round when the cell is really programmed, and we have $\alpha_t^{\min} = \alpha_k^{\min}$, $\alpha_t^{\max} = \alpha_k^{\max}$. In the following, we consider only the case $k = t$. (The case $k < t$ is a simple extension.) Note that $z_{t+1} = 0$. We prove for $i \in \{0, 1, \ldots, t\}$, $\alpha_t^{\max} - \alpha_t^{\min} = \left(\alpha_i^{\max} - \alpha_i^{\min}\right) / \left(\binom{t-i}{z_{i+1}}\right)$. This clearly holds for $i = t$. We now prove it using induction on i, for $i = t-1, t-2, \ldots, 0$. (The case $i = 0$ leads to this theorem.)

Consider the $(i+1)$-th round of programming. The parameter α'_{i+1} is chosen such that $\alpha_i^{\max} - \alpha'_{i+1} = \frac{\left(\binom{t-i-1}{z_{i+1}-1}\right)(\alpha_i^{\max}-\alpha_i^{\min})}{\left(\binom{t-i-1}{z_{i+1}-1}\right)+\left(\binom{t-i-1}{z_{i+1}}\right)}$ and $\alpha'_{i+1} - \alpha_i^{\min} = \frac{\left(\binom{t-i-1}{z_{i+1}}\right)(\alpha_i^{\max}-\alpha_i^{\min})}{\left(\binom{t-i-1}{z_{i+1}-1}\right)+\left(\binom{t-i-1}{z_{i+1}}\right)}$. This means $\frac{\alpha_i^{\max}-\alpha'_{i+1}}{\left(\binom{t-i-1}{z_{i+1}-1}\right)} = \frac{\alpha'_{i+1}-\alpha_i^{\min}}{\left(\binom{t-i-1}{z_{i+1}}\right)}$. Since either "$\alpha_{i+1}^{\max} = \alpha_i^{\max}$, $\alpha_{i+1}^{\min} = \alpha'_{i+1}$ and $z_{i+2} = z_{i+1} - 1$" or "$\alpha_{i+1}^{\max} = \alpha'_{i+1}$, $\alpha_{i+1}^{\min} = \alpha_i^{\min}$ and $z_{i+2} = z_{i+1}$", using induction we get $\alpha_t^{\max} - \alpha_t^{\min} = \left(\alpha_{i+1}^{\max} - \alpha_{i+1}^{\min}\right) / \left(\binom{t-i-1}{z_{i+2}}\right) = \frac{\alpha_i^{\max}-\alpha'_{i+1}}{\left(\binom{t-i-1}{z_{i+1}-1}\right)} = \frac{\alpha'_{i+1}-\alpha_i^{\min}}{\left(\binom{t-i-1}{z_{i+1}}\right)} = \frac{\left(\alpha_i^{\max}-\alpha'_{i+1}\right)+\left(\alpha'_{i+1}-\alpha_i^{\min}\right)}{\left(\binom{t-i-1}{z_{i+1}-1}\right)+\left(\binom{t-i-1}{z_{i+1}}\right)} = \frac{\alpha_i^{\max}-\alpha_i^{\min}}{\left(\binom{t-i}{z_{i+1}}\right)}$. ∎

From Theorem 8 and Theorem 11, we see that the bound in Theorem 8 is actually exact. We get the following result.

Corollary 12 *The above algorithm is optimal. And*

$$\Delta(t, r, \alpha_{\min}, \alpha_{\max}) = \frac{\alpha_{\max} - \alpha_{\min}}{\left(\binom{t}{r}\right)}.$$

V. Conclusion

Parallel programming is an important technique for flash memories. And flash memories have a unique iterative and monotonic cell-programming model. In this paper, we study parallel programming for flash memories, focusing on its two special properties: shared program voltages and varied programming hardness. As future work, we will further generalize the model and study the storage capacity of flash memories.

VI. Acknowledgement

This work was supported in part by the University of California Lab Fees Research Program, Award No. 09-LR-06-118620-SIEP and the Center for Magnetic Recording Research at the University of California, San Diego. The work of Anxiao (Andrew) Jiang was supported in part by the NSF CAREER Award CCF-0747415 and NSF grant ECCS-0802107.

References

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

[2] D. Haugland, "A bidirectional greedy heuristic for the subspace selection problem," *Lecture Notes in Computer Science*, vol. 4638, pp. 162-176, August 2007.

[3] A. Jiang and J. Bruck, "On the capacity of flash memories," in *Proc. Int. Symp. on Information Theory and Its Applications*, pp. 94-99, 2008.

[4] A. Jiang and H. Li, "Optimized cell programming for flash memories," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pp. 914-919, 2009.

[5] M. K. Kozlov, S. P. Tarasov, and L. G. HaEijan, "Polynomial solvability of convex quadratic programming," *Soviet Math. Doklady*, vol. 20, pp. 1108-1111, 1979.

[6] H. T. Lue, T. H. Hsu, S. Y. Wang, E. K. Lai, K. Y. Hsieh, R. Liu, and C. Y. Lu, "Study of incremental step pulse programming (ISPP) and STI edge effect of BE-SONOS NAND flash," *Proc. IEEE Int. Symp. on Reliability Physics*, vol. 30, no. 11, pp. 693-694, May 2008.

[7] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 30, no. 2, pp. 227-234, April 1995.

[8] K. D. Suh *et al.*, "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1149-1156, November 1995.