ISIT'2013 1569718503

Parallel Programming of Rank Modulation

Minghai Qin
Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093 – 0401, USA
mqin@ucsd.edu

Anxiao (Andrew) Jiang
Computer Science and Engineering
Texas A&M University
College Station, TX 77843, USA
ajiang@cse.tamu.edu

Paul H. Siegel
Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093 – 0401, USA
psiegel@ucsd.edu

***Abstract*—Rank modulation is a technique for representing stored information in an ordered set of flash memory cells by a permutation that reflects the ranking of their voltage levels. In this paper, we consider two figures of merit that can be used to compare parallel programming algorithms for rank modulation. These two criteria represent different trade-offs between the programming speed and the lifetime of flash memory cells. In the first scenario, we want to find the minimum number of programming rounds required to increase a specified cell-level vector ℓ_0 to a cell-level vector corresponding to a target rank permutation τ, with no restriction on the maximum allowable cell level. We derive lower and upper bounds on this number, denoted by $t_1^*(\tau, \ell_0)$. In the second scenario, we seek an efficient programming strategy to achieve a cell-level vector $\ell(\tau)$ consistent with the target permutation τ, such that the maximum cell level after programming is minimized. Equivalently, this strategy maximizes the number of information update cycles supported by the device before requiring a block erasure. We derive upper bounds on the minimum number of programming rounds required to achieve cell-level vector $\ell(\tau)$, denoted by $t_2^*(\tau, \ell_0)$, and propose a programming algorithm for which the resultant number of programming rounds is close to $t_2^*(\tau, \ell_0)$.**

I. Introduction

Flash memories have become the most widely-used non-volatile memories due to the fast read/write speed, low power consumption and high density. The basic memory unit, called a *cell*, is a floating-gate transistor of which the *level* is determined by the amount of charge (e.g., electrons) trapped in it. Traditionally, the data is represented by the amount of trapped charge quantized to q discrete levels and thus each cell can represent $\log_2 q$ bits of information. The cell level is increased or decreased by the hot-electron injection mechanism or Fowler-Nordheim tunneling mechanism [1]. A *page* consists of about 10^4 cells organized such that they are read and written simultaneously and a *block* consists of tens to hundreds of pages.

One of the most conspicuous properties of flash memory cells is the asymmetry in programming (increasing cell levels) and erasing (decreasing cell levels to 0). While increasing a cell level can be easily accomplished by applying a certain voltage to a cell to inject a desired amount of charge, decreasing a cell level is an enormously expensive operation in the sense that the whole block containing it has to be first erased (removing all charge of all cells within the block) before reprogramming to the target levels. This operation, called a *block erasure*, is time-consuming and, moreover, it degrades the performance of the flash memory cells. Typically, a block can tolerate 10^3 (when $q = 8$) to 10^5 (when $q = 2$) erasures before it begins to malfunction. Therefore, in order to avoid unnecessary block erasures caused by "overshooting" problems, flash memory programming is carefully accomplished by several rounds of charge injection that make cell levels monotonically approach their target values.

In order to increase the programming speed and to reduce the complexity of hardware realization, parallel programming is used, whereby a common voltage is applied to a group of cells to inject a specified charge simultaneously. For a discrete set of cell levels, parallel programming techniques that minimize a cost function of the target cell-level vector Θ and the programmed cell-level vector ℓ after t programming rounds were studied in [3], [6], [10].

One way in which the storage capacity of flash memory devices has been increased is through the use of a larger number of levels in each cell. However, as the quantization of the cell levels becomes finer, the problem of overshooting can be exacerbated, thereby compromising data integrity and limiting the achievable storage capacity. A novel idea called **rank modulation** [4] has been proposed to solve such overshooting problems. With rank modulation, the information is represented by the permutation induced by the levels of an ordered set of n cells. As a result, it does not require a discrete set of target cell levels, and recovery of the stored information is easily implemented by comparing the charges of the n cells. However, to date, programming algorithms for rank modulation, as described in [2], [4], [5], for example, have been based upon sequential programming of individual cells, resulting in fundamental limitations on the programming speed.

In this paper, we study two approaches to parallel programming for rank modulation where cell levels are integer-valued. In the first, our objective is to minimize the number of programming rounds needed to produce a cell-level vector representing the target permutation, under the assumption that there is no constraint on the magnitude of cell-level increments. A consequence of this assumption is that the cell levels may approach their physical upper limits quickly, implying that this programming technique may limit the number of information updates that can be made before a block erasure is required. In contrast, the second approach aims to minimize the number of programming rounds subject to a constraint on the difference between the maximum cell level before and after programming. This technique therefore allows the maximum

possible number of subsequent information updates before a block erasure is required. These two scenarios represent different trade-offs between programming speed and the lifetime of the flash memory.

Since the minimum number of programming rounds is a function of the target permutation τ and the initial cell-state vector $\boldsymbol{\ell}_0$, we denote them by $t_1^*(\tau, \boldsymbol{\ell}_0)$ and $t_2^*(\tau, \boldsymbol{\ell}_0)$, respectively, in the two programming scenarios. We will use the notation t_i^* and $t_i^*(\tau, \boldsymbol{\ell}_0)$ interchangeably, $i = 1, 2$, in contexts where there is no ambiguity. We first derive universal lower and upper bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$ as a function of τ. These bounds show that, when compared to the push-to-top programming scheme proposed in [4], the minimal number of programming rounds required in this scenario is approximately $\frac{1}{3}$ of that used in the push-to-top scheme. In the second scenario, the smallest set of cell-level increments needed to represent the target permutation can be determined as in [2]. We derive an upper bound on $t_2^*(\tau, \boldsymbol{\ell}_0)$ and propose an algorithm with complexity $O(n^2 \log n)$ to search for the optimal set of voltages in each round of programming. The resulting number of programming rounds is shown to be only 2.5% more than the minimum number obtained by a brute-force optimal search.

The rest of the paper is organized as follows. In Section II, we model the cell increments during parallel programming mathematically. In Section III, we aim to minimize the number of programming rounds for rank modulation when cells are allowed to increase to arbitrarily high levels. In Section IV, we aim to minimize the number of programming rounds for rank modulation when the cell increment is minimized.

II. Preliminaries

We denote n flash memory cells by $c_1, \ldots, c_n$. Cells are programmed by injecting charge to increase the cell levels. We denote by $[m : n]$ the set of integers $\{i \in \mathbb{Z} | m \leqslant i \leqslant n\}$ and $[1 : n]$ is denoted by $[n]$. The set of non-negative real numbers and non-negative integers are denoted by $\mathbb{R}_+$ and $\mathbb{Z}_+$, respectively. As assumed in [6], [10], the number of programming rounds (charge injection) is t and when a voltage $V_j, j \in [t]$, is applied to cell c_i, the cell level of c_i will be increased by $\alpha_i V_j$. We call α_i the ***hardness of charge injection*** of c_i, and we define the cell-hardness vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. Suppose the cell-state vector is $\boldsymbol{\ell}_0$ before programming and denote the set of voltages that can be applied in the t rounds of parallel programming by $\boldsymbol{V} = (V_1, \ldots, V_t)$. The final cell-state vector is then

$$\ell_{i,t} = \ell_{i,0} + \alpha_i \sum_{j=1}^{t} b_{i,j} V_j, \tag{1}$$

where the (i, j) entry in the matrix

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & b_{2,1} & \cdots & b_{n,1} \\ b_{1,2} & b_{2,2} & \cdots & b_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,t} & b_{2,t} & \cdots & b_{n,t} \end{bmatrix} \in \{0,1\}^{t \times n}$$

indicates whether or not voltage V_j is applied to cell c_i during the j-th round of programming, for $i \in [n], j \in [t]$.

Let $\boldsymbol{\ell}_t = (\ell_{1,t}, \ldots, \ell_{n,t})$ be the final cell-state vector after programming. We assume $\ell_{i,t} \neq \ell_{j,t}, \forall i \neq j$. Let Σ_n be the set of all permutations of $[n]$. Let $\tau = (\tau_1, \ldots, \tau_n) \in \Sigma_n$ be the rank permutation of $\boldsymbol{\ell}_t$, denoted by $\mathrm{rank}(\boldsymbol{\ell}_t) = \tau$, where τ_i is the index of the cell with the i-th lowest level. For example, if $\boldsymbol{\ell}_t = (1.5, 3.5, 0.5, 2)$, then $\mathrm{rank}(\boldsymbol{\ell}_t) = \tau = (3, 1, 4, 2)$.

The figure of merit for the programming algorithm is the number of required programming rounds, t. The cell programming problem is to find a set of positive real programming voltages $\boldsymbol{V} \in \mathbb{R}_+^t$ and a binary matrix $\mathbf{B} \in \{0,1\}^{t \times n}$ of cell-programming indicators such that $\mathrm{rank}(\boldsymbol{\ell}_t) = \tau$ and t is minimized, where $\boldsymbol{\ell}_t$ is given by Equation (1).

III. Rank modulation minimizing programming rounds

In this section, we assume $\alpha_i = 1, \forall i \in [n]$ and both $\ell_{i,j}$ and $V_j, i \in [n], j \in [t]$ are integers. We further assume there is no physical upper limit on the cell levels. Given the initial cell-state vector $\boldsymbol{\ell}_0$ with $\mathrm{rank}(\boldsymbol{\ell}_0) \in \Sigma_n$ and the target permutation $\tau \in \Sigma_n$, we would like to find a set of programming voltages $\boldsymbol{V} \in \mathbb{Z}_+^t$ and a binary matrix $\mathbf{B} \in \{0,1\}^{t \times n}$ of cell-programming indicators such that $\mathrm{rank}(\boldsymbol{\ell}_t) = \tau$ and t is minimized, where $\boldsymbol{\ell}_t$ can now be expressed as

$$\ell_{i,t} = \ell_{i,0} + \sum_{j=1}^{t} b_{i,j} V_j. \tag{2}$$

The following definitions will be useful in deriving bounds on the minimum possible value of t.

Definition 1. *A **decomposition** of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma_n$ decomposes σ into subsequences $((v_1), \ldots, (v_m))$, where $(v_i), i \in [m]$, is a subsequence of the form $(\sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_k})$, in which the relative orders within σ are preserved, i.e., for any two $u_j, u_k \in [n]$ in (v_i), σ_{u_j} is to the left of σ_{u_k} in v_i if and only if σ_{u_j} is to the left of σ_{u_k} in σ.*

*An **increasing block** $(\sigma_u, \sigma_{u+1}, \ldots, \sigma_v), 1 \leqslant u \leqslant v \leqslant n$, of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a contiguous subsequence of σ such that $\sigma_i < \sigma_{i+1}, \forall i \in [u : v-1]$. An increasing block decomposition of σ is a decomposition such that each subsequence is an increasing block.*

*An **increasing subsequence** $(\sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_k}), u_i \in [n], i \in [k]$, of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a subsequence (not necessarily contiguous) of σ such that $\sigma_{u_i} < \sigma_{u_{i+1}}, \forall i \in [k-1]$. An increasing subsequence decomposition of σ is a decomposition such that each subsequence is an increasing subsequence. A **decreasing subsequence** is defined similarly.*

Example 1. If $\sigma = (3, 1, 4, 5, 6, 2)$, then an increasing block decomposition is $((3), (1, 4, 5, 6), (2))$ and an increasing subsequence decomposition is $((3, 4, 5, 6), (1, 2))$.

Without loss of generality, we assume the initial rank is the identity permutation, i.e., $\sigma = (1, 2, \ldots, n)$. It is clear that the optimal t^* depends on τ and the initial cell-state vector $\boldsymbol{\ell}_0$, so we denote it by $t_1^*(\tau, \boldsymbol{\ell}_0)$. The next theorem gives a universal bound on $t_1^*(\tau, \boldsymbol{\ell}_0)$ as a function of τ.

Theorem 1. *Let m_1 and m_2 be the minimum number of subsequences in an increasing block decomposition and an increasing subsequence decomposition of τ, respectively. Then the optimal t_1^* satisfies*

$$\lceil \log m_2 \rceil \leqslant t_1^*(\tau, \boldsymbol{\ell}_0) \leqslant \lceil \log m_1 \rceil.$$

Proof: First we prove the lower bound. We will invoke the following lemma, whose proof is omitted due to space constraints.

Lemma 2. *The minimum number of subsequences in any increasing subsequence decomposition of a permutation σ is equal to the length of the longest decreasing subsequence in σ.*

According to Lemma 2, the length of the longest decreasing subsequence in τ is m_2. Let this decreasing subsequence be

$$(d_1, d_2, \ldots, d_{m_2}),$$

where

$$d_1 > d_2 > \cdots > d_{m_2}$$

and

$$\ell_{d_1,t} < \ell_{d_2,t} < \cdots < \ell_{d_{m_2},t}.$$

Since the initial permutation is assumed to be the identity permutation $\sigma = (1, \ldots, n)$, meaning $\ell_{d_1,0} > \ell_{d_2,0} > \cdots > \ell_{d_{m_2},0}$, the increments of the d_i-th cell, denoted by $(I_{d_1}, \ldots, I_{d_{m_2}})$, where $I_{d_i} = \ell_{d_i,t} - \ell_{d_i,0}, \forall i \in [m]$, have to satisfy

$$I_{d_1} < I_{d_2} < \cdots < I_{d_{m_2}},$$

i.e., at least m_2 distinct increments are needed. It follows that the number of programming rounds satisfies the lower bound $\lceil \log m_2 \rceil \leqslant t_1^*$.

We now proceed to the proof of the upper bound, which is achieved by a specific programming strategy.

Suppose

$$((\tau_1, \ldots, \tau_{k_1}), (\tau_{k_1+1}, \ldots, \tau_{k_2}), \ldots, (\tau_{k_{m_1-1}+1}, \ldots, \tau_{k_{m_1}}))$$

is an increasing block decomposition of τ of size m_1, where $k_{m_1} = n$. By assumption, the initial permutation is an identity permutation, so the cell levels satisfy

$$\ell_{1,0} < \ell_{2,0} < \cdots < \ell_{n,0} < N$$

for some integer N, which implies that $|\ell_{i,0} - \ell_{j,0}| < N, \forall i, j \in [n]$. Let $t = \lceil \log m_1 \rceil$ and $\boldsymbol{V} = (V_1, \ldots, V_t)$, where $V_j = 2^{j-1}N, \forall j \in [t]$. The set of achievable increments is, therefore, $\mathcal{I} = \{iN | i \in [0 : 2^t - 1]\}$. Now, for every index $i \in [n]$, we determine the the block index $j \in [m_1]$ such that i lies in the j-th block of the increasing block decomposition of τ, and we increase the level of the cell c_i by $(j-1)N$. Since $1 \leqslant j \leqslant m_1 \leqslant 2^t$, it follows that $(j-1)N \in \mathcal{I}$, meaning that the cell level increments that are produced can all be achieved by $\boldsymbol{V}$. We now show that the rank permutation of the cell-state vector $\boldsymbol{\ell}_t$ attained by this programming strategy is τ.

Consider a pair of cell indices u and v in τ such that u is to the left of v. If they are both in the same increasing block, then

$$\ell_{u,t} = \ell_{u,0} + I_u = \ell_{u,0} + I_v < \ell_{v,0} + I_v = \ell_{v,t},$$

where the first and last equalities follow from the definition of $I_u, u \in [n]$, and the second equality follows from the fact that the cell increments are the same within each block and the inequality follows from the relation $u < v$. If u is in the i-th block and v is in the j-th block, where $i < j$, then

$$\begin{aligned}\ell_{u,t} &= \ell_{u,0} + I_u = \ell_{u,0} + (i-1)N \\ &< N + (i-1)N = iN \leqslant (j-1)N = I_v < \ell_{v,t}.\end{aligned}$$

This proves $\ell_{u,t} < \ell_{v,t}$ if u is on the left of v in τ. Since u and v are chosen arbitrarily, the rank permutation of the final cell-state vector $\boldsymbol{\ell}_t$ is τ. Therefore, $t_1^* \leqslant \lceil \log m_1 \rceil$. ■

Lemma 2 establishes the connection between the increasing subsequence decomposition and the longest decreasing subsequence of a permutation $\tau \in \Sigma_n$. Efficient algorithms with time complexity $O(n \log n)$ that find the longest increasing/decreasing subsequence of $\tau \in \Sigma_n$ were studied in [7]–[9] and it was shown that partitioning a permutation into a minimum number of monotone subsequences is NP-hard [9]. The problem of decomposing a permutation into a minimum number of increasing subsequences is interesting by itself and next we would like to give an algorithm with time complexity $O(n \log n)$ that performs the minimum increasing subsequence decomposition of $\tau \in \Sigma_n$.

Algorithm 1: Finding Minimum Number of Increasing Subsequences

Input: A permutation $\tau = (\tau_1, \tau_2, \cdots, \tau_n)$ of the integer set $[n]$;
Output: m increasing subsequences $S_1, S_2, \ldots, S_m$.
$m \leftarrow 1$, $S_1 \leftarrow \{\tau_1\}$, $T \leftarrow \{\tau_1\}$;
for $i = 2$ **to** n **do**
  if τ_i is smaller than all the integers in T **then**
    $m \leftarrow m + 1, S_m \leftarrow \{\tau_i\}, T \leftarrow T \cup \{\tau_i\}$;
  else
    Find the largest integer p in T that is less than τ_i;
    Suppose p is contained in S_j;
    $S_j \leftarrow (S_j, \tau_i)$;
    Replace p by τ_i in T.

Note that, at each step, the set T represents the last entries in the increasing subsequences considered up to that point.

Theorem 3. *The output of Algorithm 1 satisfies*

1) *$m = m_2$;*
2) *the subsequences $S_1, S_2, \cdots, S_m$ are an increasing subsequence decomposition of τ.*

The time complexity of Algorithm 1 is $O(n \log n)$.

Proof: The proof is omitted due to space limitations. ■

Fig. 1 shows the lower and upper bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$ averaged over all $n!$ permutations of $\tau \in \Sigma_n$. Also shown, in the curve labeled "No parallel," is the number of rounds required for "push-to-top" programming [4]. The parallel programming scheme requires roughly $\frac{1}{3}$ of the number of programming rounds compared to the push-to-top scheme.

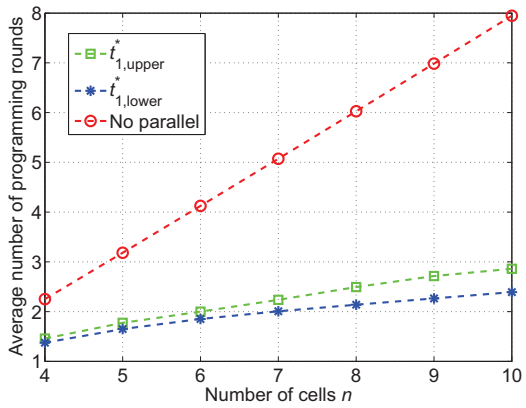

Fig. 1. Lower and upper bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$

For the case of small n, the optimal $t_1^*(\tau, \boldsymbol{\ell}_0)$ can be derived by examining each $\tau \in \Sigma_n$. For example, if $n = 3$, then

$$t_1^*(\tau, \boldsymbol{\ell}_0) = \begin{cases} 0, & \text{if } \tau = (1,2,3); \\ 2, & \text{if } \tau = (3,2,1); \\ 1, & \text{otherwise.} \end{cases}$$

Table I shows $t_1^*(\tau, \boldsymbol{\ell}_0)$ for $n = 4$. In the table, $t_1^*((3142), \boldsymbol{\ell}_0) = 1$ if $\ell_{4,0} - \ell_{3,0} \geqslant 2$ and $\ell_{2,0} - \ell_{1,0} \geqslant 2$; and $t_1^*((3142), \boldsymbol{\ell}_0) = 2$, otherwise.

TABLE I
$t_1^*(\tau, \boldsymbol{\ell}_0)$ AS A FUNCTION OF τ

τ	t^*	τ	t^*	τ	t^*	τ	t^*
1234	0	2134	1	3124	1	4123	1
1243	1	2143	1	3142	-	4132	2
1324	1	2314	1	3214	2	4213	2
1342	1	2341	1	3241	2	4231	2
1423	1	2413	1	3412	1	4312	2
1432	2	2431	2	3421	1	4321	2

Conjecture 4 *For each $\tau \in \Sigma_n$, there exists an $\boldsymbol{\ell}_0$, such that $t_1^*(\tau, \boldsymbol{\ell}_0)$ equals the lower bound $\lceil \log m_2 \rceil$; i.e., Theorem 1 provides a tight universal lower bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$.*

IV. Rank modulation maximizing the number of updates

As in the previous section, we assume $\alpha_i = 1, \forall i \in [n]$ and both $\ell_{i,j}$ and V_j are integers, for $i \in [n], j \in [t]$. Suppose that the rank of the initial cell-state vector $\boldsymbol{\ell}_0$ is the identity permutation $\sigma = (1, \ldots, n)$ and let the target permutation be $\tau \in \Sigma_n$. According to [2], in order to maximize the number of updates (or equivalently minimize the maximum cell level after programming), the final cell-state vector satisfies $\ell_{\tau_1,t} = \ell_{\tau_1,0}$, and $\ell_{\tau_i,t} = \max\{\ell_{\tau_{i-1},t} + 1, \ell_{\tau_i,0}\}, i \in [2:n]$.

Our goal is to minimize the number of programming rounds t such that the final cell-state vector is $\boldsymbol{\ell}_t$. Define the set of cell-level increments $\mathcal{I} = \{I_1, \ldots, I_m\}$ to be the set of distinct integers in $\boldsymbol{\ell}_t - \boldsymbol{\ell}_0$. Without loss of generality, we assume $0 < I_1 < I_2 < \cdots < I_m$. The cell programming problem can be formulated as follows: given $\mathcal{I}$, minimize t such that there exists $\boldsymbol{V}^T = (V_1, \ldots, V_t) \in \mathbb{Z}_+^t$ and for each $I_i \in \mathcal{I}, i \in [m]$, there exists a $\boldsymbol{b}_i \in \{0,1\}^t$ such that $\boldsymbol{V}^T \cdot \boldsymbol{b}_i = I_i$. The pair $(\boldsymbol{V}, \mathbf{B})$ is called an **optimal pair** if $(\boldsymbol{V}, \mathbf{B})$ achieves the minimum number of rounds t. A vector $\boldsymbol{V}$ is called **optimal** if there exists a $\mathbf{B}$ such that $(\boldsymbol{V}, \mathbf{B})$ is an optimal pair.

For $\boldsymbol{V} \in \mathbb{Z}_+^t$, we call the set of integers $\{z \in \mathbb{Z} | z = \boldsymbol{V}^T \cdot \boldsymbol{b}$ for some $\boldsymbol{b} \in \{0,1\}^t\}$ the **span** of $\boldsymbol{V}$, denoted by $S(\boldsymbol{V})$. If $\mathcal{I} \subset S(\boldsymbol{V})$, we say $\boldsymbol{V}$ **covers** $\mathcal{I}$. Therefore, the cell programming problem translates to finding a vector $\boldsymbol{V}$ of minimum length that covers a given integer set.

Example 2. Suppose $\mathcal{I} = \{2,5,7,8,10\}$. Let $\boldsymbol{V} = (2,3,5)^T$, and the $\boldsymbol{b}_i$'s are chosen such that

$$(2,3,5) \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} = (2,5,7,8,10).$$

Therefore, there exists a $\boldsymbol{V}$ of length 3 that covers $\mathcal{I}$.

This problem is related to a well-known NP-complete problem, Subset Sum Problem. Finding the optimal $\boldsymbol{V}$ for each given $\mathcal{I}$ is therefore a hard task, so we will settle for finding a solution which, though not necessarily optimal, compares favorably to the simple push-to-top [4] solution.

First we provide a general upper bounds on $t_2^*(\tau, \boldsymbol{\ell}_0)$ in the following theorem.

Theorem 5. *The minimum possible number of programming rounds satisfies*

$$t_2^* \leqslant \min\{\lceil \log(I_m + 1) \rceil, 1 + \lceil \log(I_m - I_1 + 1) \rceil, m\}.$$

Proof: If $\boldsymbol{V}$ is chosen as $(2^0, 2^1, \ldots, 2^{\lceil \log(I_m+1) \rceil - 1})$, then it is guaranteed that $\boldsymbol{V}$ covers $[2^{\lceil \log(I_m+1) \rceil} - 1] \supset [I_m] \supset \mathcal{I}$. Therefore, $t_2^* \leqslant \lceil \log(I_m + 1) \rceil$.

If $\boldsymbol{V}$ is chosen as $(I_1, 2^0, 2^1, \ldots, 2^{\lceil \log(I_m - I_1 + 1) \rceil - 1})$, then it is guaranteed that $\boldsymbol{V}$ covers $[I_1 : I_1 + 2^{\lceil \log(I_m - I_1 + 1) \rceil} - 1] \supset [I_1 : I_m]$. Therefore, $t_2^* \leqslant 1 + \lceil \log(I_m - I_1 + 1) \rceil$.

If $\boldsymbol{V}$ is chosen as $(I_1, I_2, \ldots, I_m)$, then it is guaranteed that $\boldsymbol{V}$ covers $\mathcal{I}$. Therefore, $t_2^* \leqslant |\mathcal{I}| = m$. ■

Here are some further results and conjectures related to this parallel programming problem.

Proposition 6. *There exists an optimal $\boldsymbol{V}$ such that all the elements in $\boldsymbol{V}$ are distinct.*

Proof: The proof is omitted due to space limitations. ■

Conjecture 7 *If the integrality constraint on $\boldsymbol{V}$ is loosened, requiring only that $\boldsymbol{V} \in \mathbb{R}_+^t$, then there still exists an optimal $\boldsymbol{V}$ such that $\boldsymbol{V} \in \mathbb{Z}_+^t$; i.e., t_2 is achieved using integer-valued voltages $\boldsymbol{V}$.*

Next we present an algorithm that searches for a short, but not necessarily minimum-length, vector $\boldsymbol{V}$ that covers the given integer set $\mathcal{I}$. Theorem 5 suggests that I_m, the largest elements in $\mathcal{I}$, and $|\mathcal{I}|$, the cardinality of $\mathcal{I}$ have a strong influence on the size and composition of an optimal $\boldsymbol{V}$. Note that

once V_1 is fixed, we can reduce all elements greater than V_1 by V_1, and reformulate the problem into that of finding a vector $\tilde{\boldsymbol{V}}$ that covers $\mathcal{I}_{\text{new}} = \{i \in \mathcal{I} | i < V_1\} \cup \{i | (i + V_1) \in \mathcal{I}\}$. The following algorithm is based on an iterative, greedy search for V_1 that, in each iteration, chooses V_1 to minimize $|\mathcal{I}_{\text{new}}|$ and, should more than one such V_1 exist, chooses one that mnimizes the largest element in $\mathcal{I}_{\text{new}}$.

Algorithm 2: Finding $\boldsymbol{V}$ that covers $\mathcal{I}$

Input: an integer set $\mathcal{I} = \{I_1, \ldots, I_m\}$
Output: an integer vector $\boldsymbol{V}$ that covers $\mathcal{I}$
$\boldsymbol{V} \leftarrow \emptyset$;
while $\mathcal{I} \not\subset S(\boldsymbol{V})$ **do**
  $\mathcal{I} \leftarrow$ ascending_sort($\mathcal{I}$);
  $min\_size \leftarrow |\mathcal{I}|$;
  $max\_elem \leftarrow I_1$;
  $V_1 \leftarrow 0$;
  for $v = 1$ **to** I_m **do**
    $\mathcal{I}_{\text{new}} \leftarrow$ reduced_set($\mathcal{I}, v$);
    if $|\mathcal{I}_{new}| < min\_size$ **then**
      $V_1 \leftarrow v$;
      $min\_size \leftarrow |\mathcal{I}_{\text{new}}|$;
      $max\_elem \leftarrow \max\{\mathcal{I}_{\text{new}}\}$;
    else
      if $|\mathcal{I}_{new}| = min\_size$ **then**
        $I_{\text{new}}^{\max} \leftarrow = \max\{\mathcal{I}_{\text{new}}\}$;
        if $I_{new}^{max} < max\_elem$ **then**
          $V_1 \leftarrow v$;
          $max\_elem \leftarrow \max\{\mathcal{I}_{\text{new}}\}$;
  $\boldsymbol{V} \leftarrow (\boldsymbol{V}, V_1)$; // append V_1 to $\boldsymbol{V}$

In Algorithm 2, ascending_sort($\mathcal{I}$) sorts $\mathcal{I}$ in ascending order and $\max\{\mathcal{I}_{\text{new}}\}$ returns the maximum element of a set $\mathcal{I}_{\text{new}}$. For a given $v \in [1 : I_m]$, reduced_set($\mathcal{I}, v$), calculated using Algorithm 3, generates $\mathcal{I}_{\text{new}}$ for the next iteration and minimizes $|\mathcal{I}_{\text{new}}|$ and $I_{\text{new}}^{\max}$.

Algorithm 3: $\mathcal{I}_{\text{new}}$=reduced_set($\mathcal{I}, v$)

Input: a sorted integer set $\mathcal{I} = \{I_1, \ldots, I_m\}$ in ascending order and an integer $v \in [1 : I_m]$
Output: an integer set $\mathcal{I}_{\text{new}}$
$flag \leftarrow$ all-zero vector of length m;
for $i = m$ **downto** 1 **do**
  if $I_i > v$ ***and*** $flag[i] == 0$ **then**
    $I_i \leftarrow I_i - v$;
    $flag[k] \leftarrow 1$ for all k such that $I_k = I_i$;
$\mathcal{I}_{\text{new}} \leftarrow$ the set of distinct elements of $\mathcal{I}$;

Proposition 8. *Suppose $I_m < N$ for some $N \in \mathbb{Z}$. Then the complexity of Algorithm 3 is $O(N)$ and that of Algorithm 2 is $O(N^2 \log N)$.*

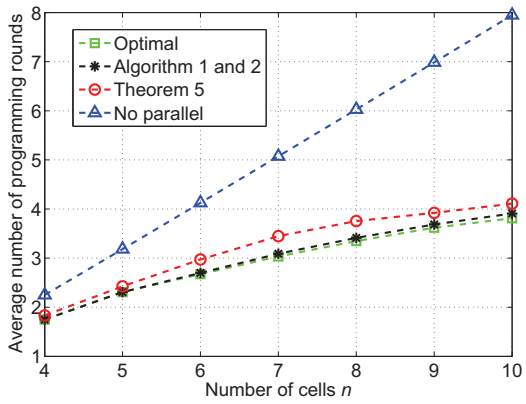

Fig. 2. Number of programming rounds needed to achieve ℓ_t

Fig. 2 shows results obtained using Algorithm 2 (and Algorithm 3). The horizontal axis represents the number of cells n and the vertical axis represents the number of programming rounds t averaged over all $n!$ target rank permutations $\tau \in \Sigma_n$. Also shown in the figure is the upper bound of Theorem 5, the number of rounds using the push-to-top strategy [2], [4], and the optimal number obtained using a brute-force, exhaustive search that minimizes t for each $\tau \in \Sigma$. We note that the number of programming rounds obtained using Algorithm 2 (and 3) is only 2.5% above the optimal result when $n = 10$.

V. Acknowledgement

This research was supported in part by the NSF under Grant CCF-1116739, NSF CAREER Award CCF-0747415, NSF Grant CCF-1217944, and the Center for Magnetic Recording Research at the University of California, San Diego.

The authors would like to thank Aman Bhatia and Bing Fan for helpful discussions.

References

[1] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash Memories*. Kluwer Academic Publishers, 1st Edition, 1999.
[2] E. Gad, A. Jiang, and J. Bruck, "Compressed encoding for rank modulation," in *Proc. IEEE Int. Symp. Inform. Theory*, St. Petersburg, Russia, July–August 2011, pp. 884–888.
[3] A. Jiang and H. Li, "Optimized cell programming for flash memories," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, BC, Canada, August 2009, pp. 914–919.
[4] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inform. Theory*, vol. 55, no. 6, pp. 2659 – 2673, June 2009.
[5] A. Jiang and Y. Wang, "Rank modulation with multiplicity," in *Proc. Globecom 2010*, Miami, FL, USA, December 2010, pp. 1866–1870.
[6] M. Qin, E. Yaakobi, and P. H. Siegel, "Optimized cell programming for flash memories with quantizers," in *Proc. IEEE Int. Symp. Inform. Theory*, Cambridge, MA, USA, July 2012, pp. 1000–1004.
[7] C. Schensted, "Longest increasing and decreasing subsequences," *Canad. J. Math. 13*, pp. 179–191, 1961.
[8] R. Stanley, "Increasing and decreasing subsequences and their variants," in *Proc. Internat. Cong. (Madrid 2006)*. American Mathematical Society, 2007, pp. 545–579.
[9] G. Stefano, S. Krause, M. Lübbeck, and U. Zimmermann, "On minimum k-modal partitions of permutations," *J. Discrete Alg. 6*, pp. 381–392, 2008.
[10] E. Yaakobi, A. Jiang, P. H. Siegel, A. Vardy, and J. K. Wolf, "On the parallel programming of flash memory cells," in *Proc. IEEE Inform. Theory Workshop*, Dublin, Ireland, August–September 2010.