Performance Coding: Codes for Fast Write and Read in Multi-Level NVMs

Evyatar Hemo and **Yuval Cassuto** Department of Electrical Engineering, Technion – Israel Institute of Technology
evyatar@tx.technion.ac.il, ycassuto@ee.technion.ac.il

***Abstract*—Multi-level memory cells are used in non-volatile memories in order to increase the storage density. Using multi-level cells, however, imposes lower read and write speeds, limiting their usability with high-performing applications. In this work we study the tradeoff between storage density and write/read speeds using codes. The contributions are codes that give high-performance write and read processes with minimal reduction in storage density. We describe the codes, give a detailed analytical treatment of their information rate and speed, provide encoding/decoding algorithms, and compare them with more basic access schemes and upper bounds. Using performance coding enables accessing the memory with variable access speeds, thus creating heterogenous storage devices serving a variety of applications with improved efficiency.**

***Index Terms*—Codes, Channel coding, Flash memory cells, Nonvolatile memory, Phase change memory.**

I. Introduction

MULTI-level non-volatile memories (NVMs) hold great promise to scale the storage capacity of mass-storage devices of the present and future. The main impediment of multi-level memories stems from the steep degradation of the read/write speed as the number of levels grows. This is because more levels mean longer processes to accurately set the cell levels and to measure them with precision upon read.

In order to achieve a viable storage device, the correct trade-off between storage density and read/write performance needs to be found. Toward this objective, we here propose and analyze coding schemes that efficiently balance between the rate of information storage and the time required to write and read information onto and from the media. With these codes, a storage device with a given number of physical levels q will be able to flexibly choose its encoding schemes to reach desired write and read speeds, with minimal resulting compromise of the information rate. This can enable creating heterogenous memory devices that can be accessed in a variety of speeds, using codes only and without any adjustments to the media and read/write circuitry.

In this paper we propose three codes that optimize write and read times, while keeping the expended redundancy to a minimum. In order to do so, the codes constrain the stored levels within a memory block in a way that is most beneficial to the write or read processes. Previous schemes like [1],[2] considered access-optimizing mappings for multi-level cells, but were more limited as they did not offer a tradeoff between storage rate and performance. Another relevant work [3] uses coding to improve the tradeoff between programming speed and the lifetime of flash memory cells in the rank modulation [4] paradigm. In Section II we introduce the first two codes C_1 and C_2 that address high-performance *write*. The codes differ by the write model they assume. The third code C_3, for high-performance *read*, is presented in Section III. As part of the analysis, we compare the codes to simpler natural alternatives and to newly derived upper bounds. We count this elaborate analytical study as the key contribution of this paper, which is an essential tool for choosing the optimal use of the codes in practical storage devices.

The best performance of the codes is achieved when the number of memory cells that can be written or read in parallel is not much higher than the number of memory levels. This is, for example, the typical case for NOR flash memory and for phase-change memory (PCM) [11], where commonly 2 to 16 cells are programmed in parallel. Toward their practical realization, we also provide low-complexity encoding/decoding algorithms for the codes. Beyond the applicability to practical storage, the results of this paper lay a basis for studying this important fundamental trade-off between density and speed in non-volatile storage devices.

Part of the results of this paper appeared in the proceedings of the 2014 International Symposium on Information Theory [12].

II. Codes for high-performance write

In the first part of the paper we look into ways to encode information when we wish to speed the *write* process up from the worst-case write time offered without coding.

A. Multi-level parallel breadth-first write model

In the commonly used *breadth-first write* process (see description in [13]), a number of cells are programmed in parallel to the same target level. For NAND technology, the memory level is reflected by the amount of electrical charge in each cell. Alternatively, for PCM cells the levels are represented by the electrical resistance values of the cells. The formal model we define next abstracts the specific physical mechanism used for writing, and thus can apply to any technology that uses breadth-first write.

Let the state of the storage cell be represented as a discrete **cell level** c, taken from the integer set $\{0,\ldots,q-1\}$. Let $\mathcal{N}$ be

a memory block of size n, and let $\boldsymbol{v} = (v_1, \ldots, v_n)$ be a vector of target levels we want to write into $\mathcal{N}$.

Definition 1. *Given a vector of cell levels $\boldsymbol{v} = (v_1, \ldots, v_n)$, with $v_i \in \{0, \ldots, q-1\}$, define the* **incidence set** *as the set $\mathcal{I}(\boldsymbol{v}) = \{s \in \{0, \ldots, q-1\} | \exists i,\ v_i = s\}$. The elements of $\mathcal{I}(\boldsymbol{v})$ are assumed to be ordered in increasing order.*

The breadth-first write scheme of the values $\boldsymbol{v}$ into the memory block $\mathcal{N}$ of size n is given by:

Algorithm 1.

$\boldsymbol{l} = \mathcal{I}(\boldsymbol{v})$
$\forall i, c_i = l_1$ // *program all cells to* $min(\boldsymbol{v})$
for k=2 to $|\boldsymbol{l}|$
  For all $c_i \in \mathcal{N}$ *with* $c_i < v_i$: $c_i = l_k$
  // *raising the level by* $l_k - l_{k-1}$
end

which means that first all cells are programmed (in parallel) to $min(\boldsymbol{v})$. Then the next higher level in $\boldsymbol{v}$ is found, and programmed into all cells that have target levels at least that value. The process continues until this is done with $max(\boldsymbol{v})$. Therefore, it is clear that the number of programming steps (which to first order is equivalent to write time) is proportional to $|\mathcal{I}(\boldsymbol{v})|$ and not to n.

B. C_1 - constrained number of levels code

From analyzing the write model it is obvious that reducing the number of levels used to store information in a memory block will also reduce the time needed to write this information (number of write steps in the "for" loop at Algorithm 1). However, reducing the number of occupied memory levels will also reduce the information storage rate of the memory block, creating an interesting tradeoff between storage efficiency and write-speed. Therefore, write optimization is achieved by constraining the number of different levels occupied in each vector of target levels $\boldsymbol{v}$. The basic most naïve write scheme supporting this principle is to use only up to ω *fixed* levels. For example, in a memory cell that includes $\{0, \ldots, q-1\}$ possible levels, the fixed scheme will be limited to the values $\{0, \ldots, \omega - 1\}$ only. However, by coding it is possible to improve the density/write-speed tradeoff.

Definition 2. *The constrained number of levels code $C_1(\omega)$ is a code in which every legal codeword $\boldsymbol{u}$ must fulfill $|\mathcal{I}(\boldsymbol{u})| \leqslant \omega$, where $0 < \omega \leqslant q$. In other words, the level subset is not fixed a priori, but any legal C_1 codeword must have up to only ω different occupied levels within the same block.*

It is clear that according to the model, in a worst-case scenario in which all of the possible levels are used, the write speed of the code C_1 is the same as that of the fixed scheme for the same ω.

Example 1. *Suppose $n = 8, q = 8$ and we set $\omega = q/2 = 4$. A legitimate codeword in the fixed model is for example $(3, 0, 2, 1, 0, 3, 1, 1)$ because it contains only levels in the set $\{0, 1, 2, 3\}$. In the code C_1, in addition to the vector above we may use the codeword $(7, 0, 4, 1, 0, 0, 7, 1)$, because it similarly includes 4 levels $\{0, 1, 4, 7\}$.*

C. Information rate analysis

In order to asses the storage efficiency of a code, we use the formal characterization of *information rate*.

Definition 3. *The* ***information rate*** *$\mathcal{R}$ of a code for n-cell, q-level memory array is defined as*

$$\mathcal{R} = log_q(A)/n,$$

where A is the number of legal combinations according to the code specification. For example, when we use the trivial code with all possible combinations, $A = q^n$, hence $\mathcal{R} = 1$.

Therefore, the information rate for the fixed write scheme is given by

$$\mathcal{R}_{Fixed}(n, q, \omega) = log_q(\omega^n)/n = log_q(\omega). \tag{1}$$

The code C_1, however, has a better information rate, which by definition is the optimal information rate that can be achieved by writing only ω different levels. To calculate this improved information rate we use the following proposition.

Proposition 1. *The number of combinations in which* ***exactly*** *$k \leqslant n$ levels are occupied in an n-cell q-level array, is given by*

$$C(n, q, k) = k! \cdot S(n, k) \cdot \binom{q}{k}, \tag{2}$$

where $S(n, k)$ is the Stirling number of the second kind [14]. For $k > n$ the number of combinations is clearly 0 by definition.

Proof: The number of combinations to choose k levels out of the q available levels is given by $\binom{q}{k}$. By multiplying it by the number of surjections from n-cell set to k-level set (which cells are assigned to which level), we obtain the desired combination count. The number of surjections from an n-set to a k-set equals $k! \cdot S(n, k)$ [15]. ∎

Theorem 2. *The* information rate *of the code C_1 is given by*

$$\mathcal{R}_{C_1}(n, q, \omega) = log_q\left[\sum_{k=1}^{min(n,\omega)} k! \cdot S(n, k) \cdot \binom{q}{k}\right]/n. \tag{3}$$

Proof: By Definition 2 and Definition 3 it is clear that the information rate of the code is given by

$$\mathcal{R}_{C_1}(n, q, \omega) = log_q\left[\sum_{k=1}^{min(n,\omega)} C(n, q, k)\right]/n.$$

Using the expression for counting combinations from Eq. (2) gives us claim (3). ∎

When analyzing and comparing the information rates of the fixed scheme and the code C_1 as a function of $log_2(q)$ it is possible to notice that the rate of C_1 is higher than the rate of the fixed scheme at all q values for a chosen n. In addition, when ω is a fixed fraction of q, $\omega = a \cdot q$ for $0 < a < 1$, the rate of C_1 increases as n becomes smaller and as q becomes larger, while the rate of the fixed model also increases as q becomes larger but does not depend on n.

Remark: from the expression in (3) it is clear that there is an explicit expression for counting the number of C_1 codewords for general parameters. Therefore, encoding and decoding of C_1 can be done using enumerative coding [5].

D. Worst-case write speed analysis

In order to assess the overall performance of the code, the write time, measured as the number of write steps, is now analyzed.

Definition 4. *The time T needed to write codeword $\boldsymbol{u}$ to an n-cell memory block is measured by the number of write steps given by $|\mathcal{I}(\boldsymbol{u})|$, (Conversion to write time given in [seconds] is achieved by multiplying T by the individual-write time in [seconds]).*

In the sequel, the subscript of T indicates the specific code used in the write process. This definition of T aims at the first-order objective of minimizing the number of program cycles. In Section II-F we refine this definition to also consider the dependence of time on the target levels.

In the worst-case write scenario exactly ω levels are occupied. Note that given a value of ω both C_1 and the fixed scheme have the same worst-case write time. To compare between C_1 and the fixed scheme, we will require identical information rates for given n and q, obtaining an equivalent number of occupied levels $\hat{\omega}$ for the fixed scheme. Equating $\mathcal{R}_{C_1}(n,q,\omega) = \mathcal{R}_{Fix}(n,q,\hat{\omega})$ yields

$$\hat{\omega} = \left\lceil \sum_{k=1}^{min(n,\omega)} k! \cdot S(n,k) \cdot \binom{q}{k} \right\rceil^{\frac{1}{n}}.$$

In other words, a C_1 codeword with given n, q and ω has the same information rate as n cells with $\hat{\omega}$ fixed memory levels (out of q). After equating the information rates it is now possible to compare the worst-case write times. Note that the value of $\hat{\omega}$ may be a non-integer. Therefore, we will make the following extended definition using the space-sharing concept:

Definition 5. *For any $\omega \in \mathcal{R}$, the function $T_{Fixed}(n,q,\omega)$ is defined as*

$$T_{Fixed}(n,q,\omega) = (\lfloor \omega+1 \rfloor - \omega) \cdot T_{Fixed}(n,q,\lfloor \omega \rfloor) + \\ +(\omega - \lfloor \omega \rfloor) \cdot T_{Fixed}(n,q,\lfloor \omega+1 \rfloor).$$

The practical motivation for this definition of space sharing is that for large memory arrays containing large number of n-cell blocks, it is possible to divide the array to two groups. The first group includes a $(\lfloor \omega+1 \rfloor - \omega)$ fraction of the total cell blocks, and uses $\lfloor \omega \rfloor$ levels. The second group includes a $(\omega - \lfloor \omega \rfloor)$ fraction of the total cell blocks, and uses $\lfloor \omega+1 \rfloor$ levels.

The worst-case *write-time ratio* (WTR) between the fixed scheme and the code C_1 is given by

$$WTR = \frac{T_{Fixed}}{T_{C_1}} = \frac{\hat{\omega}}{\omega}.$$

Since legal codewords of the fixed scheme are always legal codewords of C_1 (but not vice-versa), we have $\hat{\omega} > \omega$ and the write-time ratio for the worst-case scenario is always higher than 1. Fig. 1 presents the write-time ratio for the worst-case scenario with $\omega = q/2$. It can be observed that C_1 outperforms the fixed scheme, reaching significant improvement for lower values of n and higher values of q.

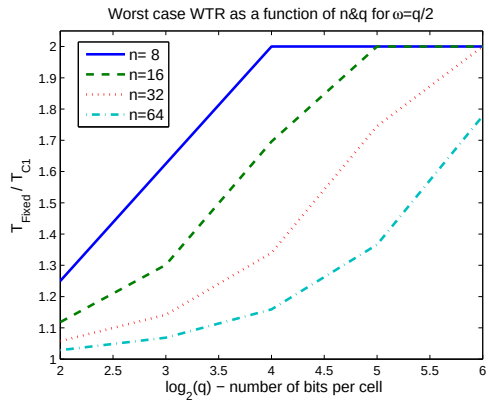

Figure 1. Worst-case write-time ratio for different values of n and q with $\omega = q/2$: $n = 8$ (solid), $n = 16$ (dashed), $n = 32$ (dotted), $n = 64$ (dash-dotted)

So far, we have compared the code C_1 with the fixed write scheme by demanding an equal information rate, and analyzing the resulting write time. However, another interesting way to compare between the two methods is to demand an equal write time, and analyzing the allowable information rate. When equalizing the worst-case write time we actually demand that ω will be identical for both methods. Fig. 2 presents the information-rate ratio $\mathcal{R}_{C_1}/\mathcal{R}_{Fixed}$ for equal worst-case write time and $\omega = q/2$. As can be noticed, C_1 gives better advantage for lower n values, with a typical $10-30\%$ higher information rate relative to the fixed write scheme.

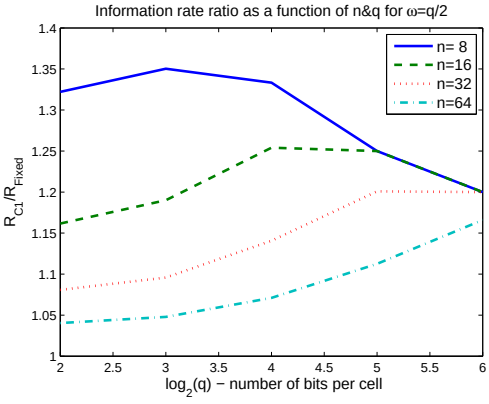

Figure 2. Worst-case information-rate ratio for different values of n and q with $\omega = q/2$: $n = 8$ (solid), $n = 16$ (dashed), $n = 32$ (dotted), $n = 64$ (dash-dotted)

E. Average write speed analysis

We now examine the performance of the *average* write time $\overline{T}$ when the written data are uniformly distributed.

Theorem 3. *The average time for writing a C_1 codeword of length n to a q-level memory block where only up to ω levels are occupied is given by*

$$\overline{T}_{C_1}(n,q,\omega) = \frac{\sum_{k=1}^{min(n,\omega)} k \cdot k! \cdot S(n,k) \cdot \binom{q}{k}}{\sum_{k=1}^{min(n,\omega)} k! \cdot S(n,k) \cdot \binom{q}{k}}. \quad (4)$$

Proof: When using the code C_1, the probability that in a codeword only k levels are occupied, $1 \leqslant k \leqslant \omega$, is given by

$$Prob(\boldsymbol{u} \in C_1, |\mathcal{I}(\boldsymbol{u})| = k) = \frac{C(n,q,k)}{\sum_{k'=1}^{min(n,\omega)} C(n,q,k')}.$$

Calculating the expectation of the number of occupied levels using Eq. (2) gives us claim (4). ■

Fig. 3 presents the tradeoff C_1 offers between information rate and write time. For fixed $q = 16$, we plot for each value of n both the information rate (solid) and the write time (dashed) of C_1. The write time is presented as the improvement relative to not using a performance code. The plots are given for two values of ω: $\omega = 8$ and $\omega = 12$. It can be observed that as n increases we get better write performance, but at the same time the storage rate decreases. These trends are faster for $\omega = 8$ than for $\omega = 12$. Therefore, ω serves as a constraining parameter that determines the trade-off between the information rate and write speed.

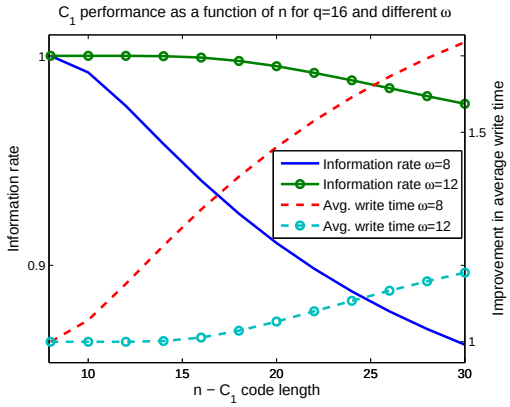

Figure 3. Performance of the C_1 code for $q = 16$ and two different values of ω - information rate and improvement in average write time as a function of n: information rate $\omega = 8$ (solid), information rate $\omega = 12$ (solid-circles), improvement in average write time $\omega = 8$ (dashed), improvement in average write time $\omega = 12$ (dashed-circles).

It is clear from Theorem 3 that for the fixed write scheme the average write time is

$$\overline{T}_{Fixed}(n,q,\omega) = \frac{1}{\omega^n} \sum_{k=1}^{min(n,\omega)} k \cdot k! \cdot S(n,k) \cdot \binom{\omega}{k}.$$

It is important to notice that by the way we define it, $\overline{T}_{C_1}$ and $\overline{T}_{Fixed}$ are actually the average number of the different levels occupied in C_1 and fixed write methods, respectively.

To compare between the code C_1 and the fixed scheme in terms of average write time, we will again require identical information rates for given n and q. Fig. 4 presents the average write-time ratio between the fixed model and C_1 as a function of n and q, with $\omega = q/2$. The entire plot reflects parameters where the code C_1 outperforms the fixed scheme (time ratios greater than 1). In addition, it is observed that it is possible to achieve up to $20-30\%$ speed improvement by using C_1.

For lower values of ω, the superiority of the average write time of C_1 is improved. For example, for $\omega = q/4$, the average write-time ratio is improved from $20-30\%$ to $40-50\%$ (for maximal improvement values).

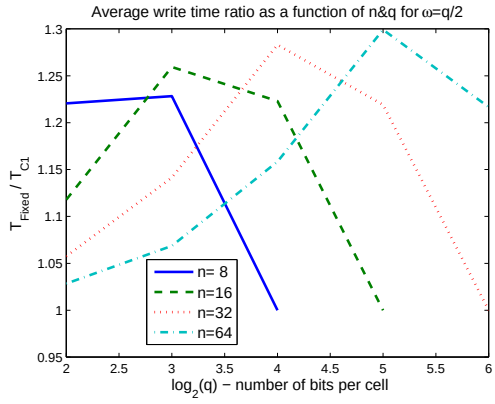

Figure 4. Average write-time ratio for different values of n and q with $\omega = q/2$: n=8 (solid), n=16 (dashed), n=32 (dotted), n=64 (dash-dotted)

F. Incremental step pulse program (ISPP) write model

So far in the section we assumed that the write time depends to first order on the *number* of distinct programmed levels within a cell block. This assumption is valid for technologies where the time spent on staging and verifying each programmed level dominates the program time. In other technologies, the time to program strongly depends on the target level itself, i.e., programming to a higher level takes more time than to a lower level. An example of such technology is the *incremental step pulse program* (ISPP) write method applied in NAND flash to mitigate cell variability [6].

In the ISPP method each program level induces a sequence of program pulses followed by a verification process to assure proximity to the target level. Higher levels require longer such sequences than lower levels. Hence we define the ISPP write model as assigning to each memory level $s \in \{0,\ldots,q-1\}$ a program time that grows linearly with s. More precisely, level s requires $m(s) = s+1$ units of time, where cells with identical target levels are being programmed simultaneously as in the previous breadth-first model. This model is realistic when different target levels have distinct program sequences, and hence program steps for lower levels do not significantly contribute to the cells with higher target levels. In this case the write time is dominated by the sum of levels. Given a write block of n cells, the total number of time units needed to program a word $\boldsymbol{v}$ is therefore equal to

$$\sum_{s \in \mathcal{I}(\boldsymbol{v})} (s+1),$$

where $\mathcal{I}(\boldsymbol{v})$ is the incidence set of $\boldsymbol{v}$ as in Definition 1. So for the remainder of the section we consider the *sum of levels* in the incidence set and not just its size as in previous subsections. To match this new definition of write time in the ISPP model, we define the code C_2.

Definition 6. *The constrained number of program pulses code*

$C_2(M)$ is a code in which every legal codeword $\boldsymbol{u}$ must fulfill

$$\sum_{s\in\mathcal{I}(\boldsymbol{u})}(s+1)\leqslant M, \tag{5}$$

where $0<M\leqslant q(q+1)/2$. In other words, any legal codeword in C_2 must be fully written in up to M time units.

The rate of the code C_2 is given in the following theorem.

Theorem 4. *Given a total write budget of M time units, the information rate of the code C_2 is given by*

$$\mathcal{R}_{C_2}(n,q,M)=\log_q\left[\sum_{m=1}^{M}\sum_{k=1}^{n}k!\cdot S(n,k)\,Q_k(m|q)\right]/n, \tag{6}$$

where $Q_k(m|q)$ is the number of partitions of m into k distinct parts, each smaller or equal to q [7].

Example 2. *The possible partitions of the integer 5 without restricting the part sizes are:* (5), (4, 1),(3, 2), (3, 1, 1), (2, 2, 1), (2, 1, 1, 1) *and* (1, 1, 1, 1, 1). *When we are interested only in partitions to 2 distinct parts we obtain $Q_2(5|5)=2$, because only* (4, 1) *and* (3, 2) *satisfy those restrictions. When we further limit the maximal part size to 3 we get $Q_2(5|3)=1$, because now only* (3, 2) *qualifies. As another example, $Q_3(5|5)=0$ because there are no three-part partitions with all-distinct parts.*

$Q_k(m|q)$ can be be calculated by the following generating function [7],[8]:

$$\begin{bmatrix}q\\k\end{bmatrix}_p p^{\binom{1+k}{2}}=\sum_m Q_k(m|q)\,p^m, \tag{7}$$

where $\begin{bmatrix}q\\k\end{bmatrix}_p$ are the Gaussian p-ary binomial coefficients [9].

Example 3. *In case we wish to calculate $Q_2(5|3)$, we can use the generating function and get*

$$\left(1+p+p^2\right)p^3=\sum_m Q_2(m|3)\,p^m. \tag{8}$$

Therefore, $Q_2(5|3)$ can be found as the coefficient of p^5, which equals 1.

Theorem 4*'s*

Proof: In order to calculate the information rate we need to calculate the number of C_2 codewords. Based on the proof of Proposition 1, it is clear that in order to calculate the number of $C_2(M)$ legal codewords, $k!\cdot S(n,k)$ in (2) must multiply the number of k-level combinations that can be programmed in up to M time units. The function $Q_k(m|q)$ counts all the combinations occupying exactly k different levels, each costing $\leqslant q$ time units, and which sum up to m. Summing for all $m\leqslant M$ and combining with Definition 3 gives (6). ∎

Similarly to the analysis of the code C_1, we now also compare the code C_2 with the fixed scheme. The fixed scheme mimics a common practice of reducing the representation of a memory level from $\{0,1,\ldots,q-1\}$ to $\{0,1,\ldots,\omega-1\}$. Recall from Section II-B that the fixed scheme uses ω fixed levels out of q. For a fair comparison with C_2 we set the ω of the fixed scheme in the ISPP model to

$$\frac{\omega(\omega+1)}{2}\leqslant M. \tag{9}$$

This value guarantees that (5) will be satisfied for the fixed scheme as well. We note that when the ω found in (9) is smaller than n, the condition (9) is in fact too restricting than needed. To overcome this issue, we limit ourselves in the comparison to only consider cases with $n\geqslant\omega$.

Theorem 5. *Given total write budget of M time units, the information rate of the fixed scheme using the ISPP write model is given by*

$$\mathcal{R}_{Fixed}(n,q,M)=\log_q\left(\left\lfloor\frac{\sqrt{8M+1}-1}{2}\right\rfloor\right), \tag{10}$$

for $n\geqslant\frac{\sqrt{8M+1}-1}{2}$. In particular, (10) holds for $n\geqslant q$.

Proof: When the total time budget is M units, the fixed scheme uses ω fixed levels by (9). When solving (9) as an equality, the only positive solution is $\omega=(\sqrt{8M+1}-1)/2$; due to the fact that ω must be an integer, we take the highest integer value fulfilling (9). Substituting it in (1) gives us (10). ∎

In order to asses the performance of the the code C_2, we compare it with the fixed scheme. Assuming a user limits the maximal number of program pulses in order to save time and power, we now compare the best information rate that can be achieved by both coding schemes. Fig. 5 presents the information rates of C_2 and the fixed scheme as functions of the write-time budget M. For all plots we get that $n\geqslant\omega$. We first notice that the information rate of the fixed scheme does not depend on n. The information rate of C_2, however, strongly depends on n, decreasing for high values of n. The plots of Fig. 5 reveal that the information rate of C_2 outperforms that of the fixed scheme for all values of $n\geqslant\omega$ and M.

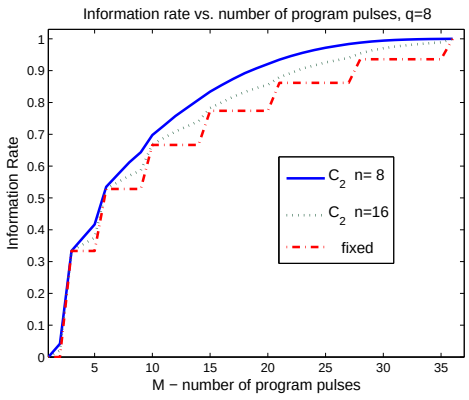

Figure 5. Information rates vs. write-time budget in the ISPP model for $q=8$ and different values of n: $n=8$ (solid), $n=16$ (dotted), fixed scheme (dash-dotted).

G. Example from the storage domain

To summarize this section, we give an example that illustrates the potential benefits of write speed-up codes in a realistic storage scenario. Let us examine a multi-level $1GB$ NVM device with $q = 8$ memory levels (hence $3b/cell$) and write speed of $5MB/s$. The device supports simultaneous write to $n = 16$ cells. Suppose we want to offer write-speed heterogeneity using the same media – so for $100MB$ (out of the $1GB$) we require a higher write speed. In this example we will assume that the write speed is inversely proportional to the number of program levels. By using C_1 with $\omega = q/2 = 4$ we obtain an information rate of $\mathcal{R}_{C_1}(n = 16, q = 8, \omega = 4) = 0.7934$. That means the designated $100MB$ of physical storage become $80MB$ for use by the customer. However, the average write time is now $T_{C_1}(n = 16, q = 8, \omega = 4) = 3.9917$ (as opposed to the original time $T_{original}(n = 16, q = 8, \omega = 8) = 7.0555$). Therefore, the write speed has increased to more than $7MB/s$, which amounts to 40% speed improvement relative to the original speed. Let us now examine the fixed write scheme. In order for the physical $100MB$ to similarly store $80MB$, the equivalent number of levels for the fixed scheme must be $\hat{\omega} \cong 5.2$. This number of levels imposes a write time of $T_{Fix}(n = 16, q = 8, \omega = 4) = 5.0273$, which yields a write-speed of $5.57MB/s$ – more than 20% slower than the speed achieved by using the code C_1 with the same storage cost.

III. Codes for high-performance read

In a similar way, coding can improve the *read* performance. We use the parallel threshold model (described in [10]), which means the cells are read by applying a sequence of threshold measurements, each applied to the n cells in parallel and returns n binary values of whether the cell levels are above or below the threshold.

Definition 7. *A* **threshold** τ *is an integer from the set* $\{1, \ldots, q-1\}$. *Given a threshold* τ, *a cell is said to be* **active** *with respect to* τ *if its cell level satisfies* $c \geqslant \tau$. *In the complementary case of* $c < \tau$, *the cell is said to be* **inactive** *with respect to* τ.

Definition 8. *A* **measurement** *is an operator acting on a block of cells* $\mathcal{N}$ *by applying a threshold of* τ, *and obtaining* $|\mathcal{N}|$ *binary values reflecting the activity of each cell in* $\mathcal{N}$ *with respect to* τ. *We denote the measurement as a vector* $\mu_\tau(\mathcal{N}) = (m_1, \ldots, m_{|\mathcal{N}|})$, *where* $m_i \in \{0, 1\}$. $m_i = 1$ *represents an active cell with respect to* τ, *and* $m_i = 0$ *represents an inactive cell.*

Example 4. *Let us measure the following block of cells* $\mathcal{N} = (7, 2, 0, 5)$ *with* $\tau = 4$. *The outcome of the measurement vector is therefore* $\mu_4(\mathcal{N}) = (1, 0, 0, 1)$.

The number of measurements M, needed to fully read a memory block, represents the effort (or time) required to complete the reading process. Therefore, in order to achieve high-performance read, M must be minimized while maintaining the information rate as high as possible.

A. Fixed number of consecutive levels code - C_3

We now introduce a code that has better read-performance than more naïve read speedup techniques. We first give some formal definitions.

Definition 9. *For a* q*-level memory cell, let us define the* i^{th} ω-**range** $r_i(\omega)$ *as the range of* ω *consecutive levels starting from* $i-1$. *Therefore,* $r_i(\omega) = [i-1, i-2+\omega]$, *for* $1 \leqslant i \leqslant q-\omega+1$.

From Definition 9 it is clear that there are $q - \omega + 1$ different ω-ranges in a q-level memory cell.

Definition 10. *For a* q*-level* n*-cell memory block, we define* $Z_i(\omega)$ *to be all possible combinations of length-*n *words where all* n *levels are taken to be in the range* $r_i(\omega)$.

Definition 11. *The fixed-number-of-consecutive-levels code* $C_3(\omega)$ *is defined as the code in which every* q*-level,* n*-cell memory block* $\boldsymbol{u}$ *satisfies* $\boldsymbol{u} \in Z_i(\omega)$, *for some* i.

In the sequel we omit the argument ω from $Z_i(\omega)$, $r_i(\omega)$ when its value is clear from the context.

Example 5. *Suppose* $n = 4$, $q = 8$ *and we set* $\omega = q/2 = 4$. *Legitimate* $C_3(4)$ *codewords are, for example,* $\boldsymbol{u} = (0, 3, 0, 1)$ *and* $\boldsymbol{v} = (3, 2, 4, 5)$ *because* $\boldsymbol{u}$ *and* $\boldsymbol{v}$ *are both* n*-cell combinations within the ranges* $r_{i=1}(4) = [0, 3]$ *and* $r_{i=3}(4) = [2, 5]$, *respectively.*

B. The information rate

Theorem 6. *The* information rate *of the code* $C_3(\omega)$ *is given by*

$$\mathcal{R}_{C_3}(n, q, \omega) = log_q\left[(q-\omega)\cdot(\omega^n - (\omega-1)^n) + \omega^n\right]/n. \quad (11)$$

Proof: For the proof, we make the following definition.

Definition 12. *Define the sets* S_i *as* $S_1 = Z_1$ *and* $S_i = Z_i \setminus (Z_i \cap Z_{i-1})$, $i > 1$.

To simplify the counting of codewords, we replace the sets $\{Z_i\}_{i=1}^{q-\omega+1}$ used to define the code with the *disjoint* sets $\{S_i\}_{i=1}^{q-\omega+1}$. There is no codeword that belongs to more than one set S_i, because belonging to S_i, $i > 1$ implies that the maximal level $i - 2 + \omega$ is occupied by at least one cell, and this level is not part of any set S_j with $j < i$.
Every set S_i, $i > 1$ includes the ω^n combinations of Z_i, excluding the combinations of $Z_i \cap Z_{i-1}$. By definition the joint combinations of Z_i and Z_{i-1} can only include the lower $\omega - 1$ levels of Z_i. Therefore, the number of combinations for each S_i, $i > 1$ is given by $\omega^n - (\omega-1)^n$. There are $(q - \omega)$ such sets in the code plus the extra set S_1 which includes ω^n combinations. Applying the total number of combinations to Definition 3 yields (11). ∎

An interesting special case is $\omega = q/2$, where the redundancy is less than one bit per q-ary cell, while the performance savings can be significant. For $\omega = q/2$ we get

$$\mathcal{R}_{C_3}\left(n, q, \frac{q}{2}\right) = log_q\left[\frac{q}{2}\cdot\left(\left(\frac{q}{2}\right)^n - \left(\frac{q}{2}-1\right)^n\right) + \left(\frac{q}{2}\right)^n\right]/n.$$

As in Section II, we compare C_3 with the natural read speedup technique in which for every memory block a *fixed*

interval of ω consecutive levels (e.g. $\{0,\ldots,\omega-1\}$) are used. We refer to this read scheme as "FixCons", and calculate its information rate to be

$$\mathcal{R}_{FixCons}(n,q,\omega) = log_q\left[\omega^n\right]/n.$$

For the case $\omega = q/2$ we get

$$\mathcal{R}_{FixCons}\left(n,q,\frac{q}{2}\right) = log_q\left[\left(\frac{q}{2}\right)^n\right]/n = 1 - log_q 2.$$

Due to the fact that $(q/2)^n - (q/2-1)^n > 0$ for all relevant n and q, it is easy to see that $\mathcal{R}_{C_3}(n,q,q/2) > \mathcal{R}_{FixCons}(n,q,q/2)$, which means that the rate of C_3 exceeds that of FixCons for all n and q.

C. Number of measurements in the read process

We now examine the benefits in read performance offered by the coding scheme defined in the previous sub-section. We assume that upon reading a block, the reader does *not* know if the levels are taken according to the code, or unconstrained from the entire set of q levels. This assumption is necessary if one wants to mix fast and regular reads without storing side information for the read circuitry.

We start with describing the reader for FixCons. In the FixCons scheme (assumed to use the lowest consecutive levels as its fixed interval), the reader will start a sequential threshold-measurements scan from level $\tau = 1$ and terminate when it reaches the first level that is higher than all stored levels, which is ω in the worst case. Thus the worst-case number of measurements needed to fully read a memory block in the FixCons scheme is $M_{FixCons} = \omega$, for any $\omega \leqslant q-1$ (note that $\omega = q-1$ does not give any advantage in that setup). We now show a read algorithm for the code C_3.

Algorithm 2.

ReadC$_3(\mathcal{N})$
- $\tau_0 = q/2$
- $\boldsymbol{m_0} = \mu_{\tau_0}(\mathcal{N})$
- $\tau = \tau_0$, $\boldsymbol{m} = \boldsymbol{m_0}$
- *while* $'1' \in \boldsymbol{m}$
  - $\tau = \tau + 1$ // *go upwards*
  - $\boldsymbol{m} = \mu_\tau(\mathcal{N})$
- *end*
- $\tau = \tau_0$, $\boldsymbol{m} = \boldsymbol{m_0}$
- *while* $'0' \in \boldsymbol{m}$
  - $\tau = \tau - 1$ // *go downwards*
  - $\boldsymbol{m} = \mu_\tau(\mathcal{N})$
- *end*

Theorem 7. *The number of threshold measurements needed to fully read a C_3 codeword is $M_{C_3} = \omega+1$ when $q/2 \leqslant \omega \leqslant q-2$.*

Proof: When reading a memory block using Algorithm 2 with $\omega \geqslant q/2$, there is at most one measurement that falls outside the set of levels allowed for codewords in S_i. To see this, observe that the center level $\tau_0 = q/2$ is allowed in all sets S_i, except possibly S_1. Starting at $\tau_0 = q/2$, Algorithm 2 will go upward until reaching τ such that all levels are $< \tau$. Then it will go downward until reaching τ such that all levels are $\geqslant \tau$. The total number of visited levels is thus at most $\omega+1$. When $i = 1$, the level $\tau_0 = q/2$ may be above the interval of levels in S_1, but the total number of measurements is still at most $\omega+1$ (in fact at most ω in this case). ■

Example 6. *Suppose $n = 4$, $q = 8$, we set $\omega = q/2 = 4$ and we wish to read the C_3 codeword $\boldsymbol{u} = (3,2,4,5)$. According to Algorithm 2, we start measuring with $\tau_0 = 4$ which gives $\mu_4(\boldsymbol{u}) = (0,0,1,1)$. Therefore, the algorithm continues measuring sequentially with higher threshold $\tau_1 = 5$ until measuring with $\tau_2 = 6$ which returns $\mu_6(\boldsymbol{u}) = (0,0,0,0)$. Then, the algorithm moves to measure with the lower thresholds $\tau_3 = 3$ and $\tau_4 = 2$ until all cells are read. Overall, Algorithm 2 requires $\omega+1 = 5$ measurements in order to fully read $\boldsymbol{u}$.*

It is important to notice that every read speed-up code must also include a matching read algorithm which effectively reads it. Only the combination of a code optimizing the data and a reading algorithm designated to read the specific structure of the data may lead to an effective read performance code.

D. Comparing C_3 vs. FixCons

To compare between the code C_3 and the FixCons scheme, we will require identical information rates for given n, q and ω, obtaining an equivalent number of allowed levels $\hat{\omega}(\omega)$ for the FixCons scheme. Equating $\mathcal{R}_{C_3}(n,q,\omega) = \mathcal{R}_{FixCons}(n,q,\hat{\omega})$ for $\omega = q/2$ yields

$$\hat{\omega}\left(\frac{q}{2}\right) = \left[\frac{q}{2}\cdot\left(\left(\frac{q}{2}\right)^n - \left(\frac{q}{2}-1\right)^n\right) + \left(\frac{q}{2}\right)^n\right]^{1/n}. \tag{12}$$

Definition 13. *For given n, q and ω, we define the **equal-information measurement ratio (EMR)** as the ratio between the number of measurements of FixCons and that of C_3. That is*

$$EMR(\omega) = \frac{M_{FixCons}}{M_{C_3}} = \frac{\hat{\omega}(\omega)}{\omega+1}, \tag{13}$$

in particular for $\omega = q/2$ we get

$$EMR(q/2) = \frac{\hat{\omega}(q/2)}{q/2+1}, \tag{14}$$

where $\hat{\omega}$ is given in (12).

In the sequel, for convenience we omit the ω argument from the $EMR(\omega)$ and $\hat{\omega}(\omega)$ functions, referring from now on to $\omega = q/2$. The maximal value of the EMR is attained when $n = 1$, yielding $\hat{\omega} = q$ and $EMR = q/(q/2+1)$, which approaches 2 for large q. Note that unlike the C_1 and C_2 write speedup codes, which always outperform the fixed scheme, at the other extreme of $n >> q$, C_3 may have $EMR < 1$ due to the extra measurement required for full read as indicated in Theorem 7. In intermediate cases of constant $n > 1$ and large q, we get the following

Theorem 8. *When $n > 1$ and $q >> n$ the asymptotic expression for the EMR ($\omega = q/2$) is given by*

$$EMR \cong \sqrt[n]{n+1}\cdot\left(1 - \frac{n-1}{n+1}\cdot\frac{1}{q}\right). \tag{15}$$

Proof: Due to the fact that $\hat{\omega} > q/2$ and $q >> n > 1$ the EMR can be approximated by

$$EMR \cong \frac{\hat{\omega}}{q/2} = \left[1 + \frac{q}{2} \cdot \left(1 - \left(1 - \frac{2}{q}\right)^n\right)\right]^{1/n}.$$

Approximating $(1 - 2/q)^n$ by $[1 - 2n/q + 2n(n-1)/q^2 + \ldots]$ and neglecting the higher-order terms gives

$$EMR \cong \left[1 + n - \frac{n(n-1)}{q}\right]^{1/n} = \sqrt[n]{n+1} \cdot \left[1 - \frac{n-1}{n+1} \cdot \frac{n}{q}\right]^{1/n}, \quad (16)$$

considering $n/q << 1$, further approximation by neglecting $O\left(1/q^2\right)$ terms from the Taylor series of (16) yields (15). ■

For small to moderate values of n, the EMR is larger than 1, representing advantage for C_3. By using (14), we get that for $(n = 16, q = 32)$, the number of measurements required by the FixCons scheme is 10% higher than with the C_3 code and matching read algorithm (with the same information rate). A more significant advantage is exhibited for $(n = 4, q = 64)$, where the number of measurements in the FixCons scheme is 43% higher than with C_3.

Values of $EMR(\omega)$ for different values of ω (not necessarily $\omega = q/2$), calculated by (13), are presented in Fig. 6. For the interesting values of n on the left end of the plot we see that the EMR is highest for $\omega = q/2$, and diminishes as ω approaches q.

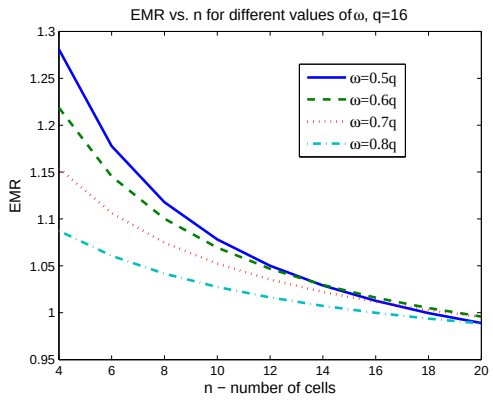

Figure 6. EMR vs. n for $q = 16$ and different values of ω: $\omega = 0.5q$ (solid), $\omega = 0.6q$ (dashed), $\omega = 0.7q$ (dotted), $\omega = 0.8q$ (dash-dashed).

E. Upper bound on information rate given read effort

After presenting constructive methods to lower the measurement count using redundancy, we now seek a converse result on limits for such measurement reduction given the expended redundancy. Specifically, we look for upper bounds on the best information rate achievable when up to M threshold measurements are applied. The upper bounds enable us to asses the performance of the FixCons and C_3 read schemes. We start with a simple upper bound, followed by a tighter, more involved one.

Proposition 9. *The information rate in a q-level memory block read by up to M threshold measurements is bounded from above by*

$$\mathcal{R}(q, M) \leqslant min\left(M \log_q 2, 1\right). \quad (17)$$

Proof: When applying a single threshold measurement we obtain one bit of information (above/below) from each of the n cells. Therefore, when taking M measurements, the number of combinations which can be read is not more than $2^{M \cdot n}$. Taking $\log_q$ and dividing by n yields the first expression in the min function of the upper bound in (17). In addition, by definition the information rate cannot be higher than 1, hence the second expression inside the min function. ■

We refer to the upper bound presented in Proposition 9 as the *naïve upper bound*, since it does not take into account the structure by which the read process obtains the nM bits. We now derive a significantly tighter upper bound.

Theorem 10. *The information rate in a n-cell, q-level memory block read by up to M threshold measurements is bounded from above by*

$$\mathcal{R}(n, q, M) \leqslant \log_q \left[\sum_{m=1}^{M} \sum_{(k,L,j) \in \psi(m)} k! \cdot S(n,k) \cdot D_j(q,k,L)\right] / n, \quad (18)$$

where $\psi(m) = \{(k, L, j) \mid k + L - j = m\}$ and $1 \leqslant k \leqslant n$, $1 \leqslant L \leqslant k$, $j \in \{0, 1, 2\}$ and the expressions for $D_j(q, l, L)$ are [17]:

$$D_0(q,l,L) = \binom{l-1}{L-1}\binom{q-l-1}{L},$$

$$D_1(q,l,L) = 2\binom{l-1}{L-1}\binom{q-l-1}{L-1},$$

$$D_2(q,l,L) = \binom{l-1}{L-1}\binom{q-l-1}{L-2} + \Delta[l = q; L = 1].$$

The function $\Delta[]$ is 1 when all of its arguments are true and 0 otherwise.

Proof: In order to calculate the upper bound we need to find a necessary condition for being able to read an information vector in up to M measurements. For a given codeword $\boldsymbol{c}$, the minimal number of measurements needed to fully read $\boldsymbol{c}$ is given by [10] $\hat{M}(\boldsymbol{c}) = |\mathcal{I}(\boldsymbol{c}) \cup \mathcal{I}^*(\boldsymbol{c})|$, where $\mathcal{I}^*(\boldsymbol{c})$ is the **shifted incidence set** defined as $\mathcal{I}^*(\boldsymbol{c}) = \{s \in \{1, \ldots, q-1\} | \exists i,\ c_i + 1 = s\}$. Therefore, in order to find all the combinations which can be read by at most M measurements we need to find all codewords $\boldsymbol{c}$ that fulfill $\hat{M}(\boldsymbol{c}) \leqslant M$. In order to do so, we recall from [10] that there are $k! \cdot S(n,k) \cdot D_j(q,k,L)$ level combinations with k used levels, falling into L consecutive runs, with j levels in the two extremes 0 and $q-1$. Each such combination requires at least $k + L - j$ measurements. Hence the number of combinations that can be measured by M measurements are not more than those that satisfy $k+L-j \leqslant M$. This is exactly the expression inside the log in (18). ■

The information rates of the upper bounds, the code C_3 and the FixCons scheme are presented in Fig. 7. The code C_3 is presented for the higher more practically interesting information rates. C_3 outperforms the FixCons scheme for all relevant values of M, however, the superiority of C_3 (relative to the FixCons scheme) decreases with M. In addition, for middle-range values of M, the upper bound (18) is much

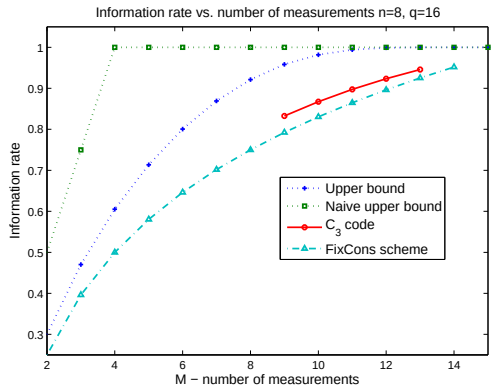

Figure 7. Information rates as a function of number of measurements: Upper bound (pluses), Naive upper bound (squares), C_3 (circles), FixCons scheme (triangles).

tighter than the naïve upper bound. Note that by Theorem 7, C_3 must have $\omega \geqslant q/2$, therefore, C_3 can only be presented for $M \geqslant q/2+1$.

F. C_3 additional high-performance write capabilities

Aside from enhancing read performance, note that restricting the C_3 codewords to ω consecutive levels, trivially restricts the number of different levels to ω, and hence we obtain the following proposition.

Proposition 11. *For all $\omega < q$, a C_3 codeword is also a C_1 codeword,*

$$C_3 \subset C_1. \tag{19}$$

When $\omega = q$, C_3 and C_1 are identical and contain all possible n-cell q-ary combinations.

Therefore, when using the C_3 code for read speed, we also obtain optimal write speed.

G. C_3 encoding & decoding schemes

We now describe encoding and decoding procedures for C_3, which are relatively simple, easy to implement, only have elementary arithmetics and simple enumeration, and in particular have $O(n)$ complexity. Let $\mathcal{D}(n,q) : \{\mathbb{Z}_q\}^n \to \mathbb{Z}_{q^n}$ be the trivial decoder of an n-cell, q-level array to an integer in the range 0 to q^n-1. We now describe the decoder for the code C_3, $\mathcal{D}_{C_3}(n,q,\omega)$ which maps a C_3 codeword to an integer $\mathcal{X}$ in the range 0 to $(q-\omega)\cdot(\omega^n-(\omega-1)^n)+\omega^n-1$. The C_3 decoder $\mathcal{D}_{C_3}(n,q,\omega)$ is implemented by using $\mathcal{D}(n,q)$ and simple arithmetic operations. The main idea behind $\mathcal{D}_{C_3}(n,q,\omega)$ is that it first determines the set S_i (defined in Definition 12) which contains the codeword; then it determines the corresponding integer by an internal enumeration within the set S_i.

Definition 14. *Let $\lambda\left(n,\hat{j}\right)$ be an enumeration function mapping the integers $1,\ldots,\binom{n}{\hat{j}}$ to size-$\hat{j}$ subsets of $\{1,\ldots,n\}$.*

Example 7. *For $n = 4$ and $\hat{j} = 2$ we have the following enumeration where ✓ and − denote, respectively, coordinates in and out the size-$\hat{j}$ subset.*

TABLE I
EXAMPLE FOR $\lambda\left(n,\hat{j}\right)$

$\lambda\left(n,\hat{j}\right)$	subsets			
1	✓	✓	−	−
2	✓	−	✓	−
3	✓	−	−	✓
4	−	✓	✓	−
5	−	✓	−	✓
6	−	−	✓	✓

The first preliminary decoding step of $\mathcal{D}_{C_3}(n,q,\omega)$ is to determine the highest occupied level ℓ. In case $\ell = \omega-1$, the decoding is done by $\mathcal{D}(n,\omega)$. Otherwise, $\mathcal{D}_{C_3}(n,q,\omega)$ continues with the following decoding sequence:

1) Determine the set S_i containing the codeword by $i = \ell-\omega+2$
2) Find $\hat{j}$, the number of times ℓ appears
3) Obtain the index of $\lambda\left(n,\hat{j}\right)$ corresponding to the $\hat{j}$ coordinates with level ℓ
4) Decode all the cells which are not set to ℓ by $\tilde{\mathcal{X}}_{r3} = \mathcal{D}\left(n-\hat{j},\omega-1\right)$
5) Return outcome $\mathcal{X}$:

$$\mathcal{X} = \omega^n + (i-2)\left(\omega^n-(\omega-1)^n\right) + \sum_{j=1}^{\hat{j}-1}\binom{n}{j}(\omega-1)^{n-j} + \left(\lambda\left(n,\hat{j}\right)-1\right)(\omega-1)^{n-\hat{j}} + \tilde{\mathcal{X}}_{r3}. \tag{20}$$

All in all the decoding procedure consists of only elementary counting and arithmetic operations.

We now turn to describe an encoder for C_3. Let $\mathcal{E}(n,q) : \mathbb{Z}_{q^n} \to \{\mathbb{Z}_q\}^n$ be the trivial encoder of an integer in the range 0 to q^n-1 to an n-cell, q-level array. We now describe the encoder for the code C_3, $\mathcal{E}_{C_3}(n,q,\omega)$, which maps an integer $\mathcal{X}$ ranging from 0 to $(q-\omega)\cdot(\omega^n-(\omega-1)^n)+\omega^n-1$ to a C_3 codeword. Similarly to the decoder, the C_3 encoder is implemented by using $\mathcal{E}(n,q)$ and simple arithmetic operations. In the trivial case of $\mathcal{X} < \omega^n$ we encode $\mathcal{E}_{C_3}(n,q,\omega) = \mathcal{E}(n,\omega)$. Otherwise we start the following sequence of operations. Let us first define $\tilde{\mathcal{X}} = \mathcal{X}-\omega^n$; we can determine the range $r_i(\omega)$ of the codeword (corresponding the the set S_i) by calculating

$$i = \left\lceil \frac{\tilde{\mathcal{X}}+1}{\omega^n-(\omega-1)^n} \right\rceil + 1.$$

For example $\mathcal{X} = \omega^n$ is mapped to $r_2(\omega)$. After determining the range, let us now define $\tilde{\mathcal{X}}_{r1} = \mod\left(\tilde{\mathcal{X}}, \omega^n-(\omega-1)^n\right)$, so the encoding problem is now reduced to encoding the integer $\tilde{\mathcal{X}}_{r1}$ to the n-cell, $r_i(\omega)$ range array. The next encoding step is finding the minimal value of j ($1 \leqslant j \leqslant n$), $\hat{j}$ that fulfills $\tilde{\mathcal{X}}_{r1} < \sum_{j=1}^{\hat{j}}\binom{n}{j}(\omega-1)^{n-j}$. By definition, a C_3 codeword in range $r_i(\omega)$ must have its level $\ell = i-2+\omega$ occupied. We denote by $\hat{j}$ the number of cells set to this level. In the next step we look to encode the residual integer

$\tilde{\mathcal{X}}_{r2} = \tilde{\mathcal{X}}_{r1} - \sum_{j=1}^{\hat{j}-1} \binom{n}{j} (\omega-1)^{n-j}$. The next step is to choose which $\hat{j}$ cells out of the n are set to level $\ell = i - 2 + \omega$; we denote these cells by ψ. We obtain ψ by indexing the subset enumerator $\lambda\left(n, \hat{j}\right)$ with the integer

$$\left\lfloor \frac{\tilde{\mathcal{X}}_{r2} + 1}{(\omega-1)^{n-\hat{j}}} \right\rfloor. \tag{21}$$

The final step is then encoding the integer $\tilde{\mathcal{X}}_{r3} = \mod\left(\tilde{\mathcal{X}}_{r2}, (\omega-1)^{n-\hat{j}}\right)$, with the trivial encoder $\mathcal{E}\left(n-\hat{j}, \omega-1\right)$ to all cells except ψ and adjusting it to the range of $r_i(\omega)$ by simply adding $i-1$ to all the cells.

Example 8. *Let us assume we use a $n = 5$, $q = 8$ memory block, we fix ω to be $q/2 = 4$. The total number of C_3 codewords is* 4148*, enough to accept* 12*-bit information words as inputs to the encoder. We take as example the input integer $\mathcal{X} = 2963$ which we wish to map to a C_3 codeword. Due to the fact that $\mathcal{X} \geqslant \omega^n = 1024$ we get $\tilde{\mathcal{X}} = 1939$, hence $i = 4$ which means the codeword is in the range $r_4(4)$. At the next step, given that $\tilde{\mathcal{X}}_{r1} = \mod(1939, 781) = 377$, we can find that the number of times the level $\ell = i - 2 + \omega = 6$ is occupied is given by $\hat{j} = 1$. As a result, we get $\tilde{\mathcal{X}}_{r2} = \tilde{\mathcal{X}}_{r1} = 377$. The $\hat{j} = 1$ cell in level $\ell = 6$ is found as index* 5 *of the subset enumerator $\lambda(5,1)$. The next step is to encode $\tilde{\mathcal{X}}_{r3} = 53$ to the remaining* 4 *cells using the trivial encoder $\mathcal{D}\left(n - \hat{j} = 4, \omega - 1 = 3\right)$, obtaining* $\{1,2,2,2\}$*, which are then adjusted to* $\{4,5,5,5\}$ *because $i = 4$. The final encoded vector is the C_3 codeword* $\{4,5,5,5,6\}$.

Example 9. *In this example we wish to decode the C_3 codeword* $\{4,5,5,5,6\}$ *from the previous example. We follow the steps of the decoding process described above:*

1) *The highest level ℓ is* 6*, so $i = 4$*
2) *ℓ appears $\hat{j} = 1$ times*
3) *ℓ corresponds to the fifth index of the codeword, so we need to use $\lambda(5,1) = 5$.*
4) $\{4,5,5,5\}$ *is converted to* $\{1,2,2,2\}$ *and trivially decoded to $\tilde{\mathcal{X}}_{r3} = 53$*
5) *By* (20)*, the outcome is $\mathcal{X} = 2963$*

H. Example from the storage domain

Let us examine a multi-level $1GB$ NVM device with $q = 16$ memory levels and worst-case read speed of $15MB/s$. The device supports simultaneous read from $n = 8$ cells. Let us now assume that for $100MB$ (out of $1GB$) we need a higher worst-case read speed, for which we are willing to reduce the amount of available storage by $20MB$. Using the code C_3 with information rate of 0.8 decreases the worst case number of threshold measurements from 15 to 8, enabling higher read speed of $22.5MB/s$. With the FixCons scheme, in comparison, we get for the same redundancy $\sim 11\%$ lower speed of $20MB/s$.

IV. Conclusion

Using coding within multi-level NVMs can significantly improve read and write performance while keeping the expended redundancy to a minimum. For write speed-up the codes C_1 and C_2 were presented and analyzed. An interesting future direction is to propose similar constructions and analyses for other realistic write models. For read speed-up, the code C_3 was introduced with a matching read algorithm. It is shown that C_3 outperforms a fixed scheme, and can be encoded and decoded efficiently. Given the gap between C_3 and the provided upper bound, it is an interesting open question whether there exist better codes with matching read algorithms. Another interesting future direction is combining performance codes with error-correcting codes, simultaneously improving access times and data reliability.

Acknowledgment

This work was partly done under a joint ISF-UGC project. In addition, it was supported by a Marie Curie CIG grant, by the Israel Ministry of Science and Technology, and by an Intel ICRI-CI grant. We also wish to thank the anonymous reviewers that helped to improve this paper.

References

[1] K. Takeuchi, et-al. "A multipage cell architecture for high-speed programming multilevel NAND flash memories," IEEE Journal of Solid-State Circuits, vol. 33.8, pp.1228-1238, Aug 1998.

[2] A. Berman, Y. Birk, "Minimal Maximum-Level Programming: Faster Memory Access via Multi-Level Cell Sharing," in Proc. of IEEE Global Communications Conference (GLOBECOM), pp.2705-2710, 9-13 Dec. 2013.

[3] Q. Minghai, A. Jiang, P.H. Siegel, "Parallel programming of rank modulation," in Proc. of IEEE Int. Symp. on Information Theory (ISIT), pp.719-723, 7-12 July 2013.

[4] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," IEEE Trans. on Information Theory, vol. 55, no. 6, pp.26592673, June 2009.

[5] T.M. Cover, "Enumerative source encoding," IEEE Trans. on Information Theory, vol.19, no.1, pp.73-77, Jan 1973.

[6] J. Brewer, M. Gill, *Nonvolatile Memory Technologies with Emphasis on Flash*. IEEE Press Series on Microelectronic Systems, 2001.

[7] W. Chu, L. Di Claudio, *Classical Partition Identities and Basic Hypergeometric Series*. Department of Mathematics, University of Lecce, Italy 2004.

[8] G. H. Hardy, E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford Science Publications, fifth edn, Oxford University Press, 1979.

[9] G. E. Andrews, *The theory of partitions*, Vol. 2. Cambridge University Press, 1998.

[10] E. Hemo, Y. Cassuto, "Adaptive threshold read algorithms in multi-level non-volatile memories," IEEE Journal on Selected Areas in Communications, vol.32, no.5, pp.847-856, May 2014.

[11] S. Kang et al., "A 0.1-m 1.8-V 256-Mb phase-change random access memory (PRAM) with 66-MHz synchronous burst-read operation," IEEE Journal of Solid-State Circuits, vol. 42.1, pp.210-218, Jan 2007.

[12] E. Hemo, Y. Cassuto, "Codes for High Performance Write and Read Processes in Multi-Level NVMs,"in Proc. of IEEE Int. Symp. on Information Theory (ISIT), pp.2092-2096, June 29 July 4, 2014.

[13] A. Berman, Y. Birk, "Constrained flash memory programming,"in Proc. of IEEE Int. Symp. on Information Theory (ISIT), pp.2128-2132, July 31 - Aug 5, 2011.

[14] J. van Lint and R. Wilson, *A Course in Combinatorics, second edition*. Cambridge University Press, 2001.

[15] A. Mohr, T.D. Porter, "Applications of Chromatic Polynomials Involving Stirling Numbers", Department of Mathematics Southern Illinois University, 2008.

[16] C. Trinh et al., "A 5.6 MB/s 64 Gb 4 b/Cell NAND ash memory in 43 nm CMOS," in Proc. of IEEE Int. Solid-State Circuits Conference (ISSCC), vol. 52, pp.246-247, Feb. 2009.

[17] Y. Cassuto, M. Blaum, "Codes for symbol-pair read channels", IEEE Trans. on Information Theory, Vol 57, No. 12, December 2011.