Minimal Maximum-Level Programming—Combined Cell Mapping and Coding for Faster MLC Memory

Amit Berman and Yitzhak Birk, *Senior Member, IEEE*

***Abstract*— In multi-level-cell memory, such as flash and phase-change memory, shrinking cell size and the growing number of levels per cell worsen the access rate to capacity ratio and even reduce access rate. We present minimal maximum-level programming, a scheme for expediting cell programming by sharing physical cells among multiple data sectors and exploiting the fact that making moderate changes to a cell's charge level is faster than making large ones. In particular, we encode the data such that in the kth writing of data to a cell, only the lowest $k+1$ levels are utilized. Unlike in previously proposed cell-sharing schemes, different same-size data sectors occupy different numbers of physical cells, and a cell may hold a fraction of a bit of a given data sector. Nevertheless, the exposed sector size remains unchanged. Data are encoded, but without redundancy. In a four-level cell example, we achieve up to 75% reduction in write latency. Read latency may be degraded, depending on the percentage of utilized capacity.**

***Index Terms*— Memory architecture, cache storage, flash memory cells, phase-change memory, system performance, signal design, modulation coding.**

I. Introduction

NAND Flash is currently the most prominent non-volatile semiconductor memory technology, used mostly for storage [1]. Phase-Change Memory (PCM) is viewed by some as a possible replacement for DRAM [2]. Both Flash and PCM employ multi-level cells (MLC) [1], [2], and designers strive to increase density by reducing cell size and increasing the number of levels.

A. Performance Implications of MLC

Flash MLC programming (writing) entails several steps: first, a data page (which comprises one or more data sectors) is transferred from the host to an on-chip memory buffer; next, a high voltage pulse (program pulse) is applied to the cells being programmed. A program pulse's impact on different cells may vary due to manufacturing variations. Also, decreasing a cell's level entails applying voltage to the bulk, so it cannot be performed to individual cells. Consequently, over-programming of a cell must be avoided. Programming is therefore carried out via a sequence of small pulses, each followed by read in order to verify the cell's level. The program-verify cycle is repeated until the desired levels are achieved [1].

TABLE I
Latency of SLC and 4-Level MLC in PCM and Flash [3]–[6], [40]

		PCM	NAND Flash
Read Latency	SLC	10 ns	25 μs
	MLC	44 ns	50 μs
Write Latency	SLC	100 ns	200 μs
	MLC	395 ns	900 μs

Write latency increases with an increase in the number of levels. As seen in Table 1, it increases faster than the increase in the number of levels, e.g., from 200μs for 2-level cells to 900μs for 4-level cells.

A cell's level is determined by applying a reference voltage to the cell and comparing its threshold voltage to the reference. While each read-verify (during Write) entails a single reference comparison, the determination of a cell's level during read requires multiple reference comparisons, each with a different reference voltage. Therefore, read latency also increases with an increase in the number of levels [3] (Table 1).

The move to MLC, while beneficial in terms of storage capacity and cost per bit, comes with a performance penalty. Moreover, with increased capacity and reduced performance, the normalized performance drop is dramatic. In this paper, we propose a scheme to improve the write performance, while the read performance is reduced only if more than 75% of MLC storage capacity is utilized.

Before presenting our contribution, we next briefly survey relevant related work. Additional (seemingly) related work will be mentioned later.

B. Related Work

The key to all access acceleration schemes is a critical observation whereby if the maximum (over cells being accessed) current cell level (for read) and cell target level (for write) is known, then one can save time. For example, if the maximum target level is 2 then one need not spend the time for reaching level 3 or above. Similarly, if (when reading), it is known

Manuscript received May 1, 2016; accepted August 17, 2016. Date of publication August 26, 2016; date of current version October 11, 2016. This work was supported in part by the HPI Institute for Scalable Computing and in part by the Technion through a Mitchell Innovation Grant. *(Corresponding author: Amit Berman.)*

A. Berman is with the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 3200003, Israel, and also with Samsung Corporation, Ramat Gan 52522, Israel (e-mail: bermanam@tx.technion.ac.il).

Y. Birk is with the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 3200003, Israel (e-mail: birk@ee.technion.ac.il).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2016.2603791

0733-8716 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

that all cells are at one of the first two levels, the number of reference comparisons can be reduced accordingly.

In FlexFS [7], the file system dynamically decides whether to use any given wordline as SLC or MLC. Use in SLC mode increases endurance and accelerates access. In all modes, any given cell contains data belonging to a single data page. The number of cells per data page varies with the number of levels being used, reflecting the change in cell capacity and keeping a fixed logical page size. Therefore, a page (and thus its entire block) must be erased when switching its mode.

In Multipage Programming (MP) [8], each 4-level cell is shared among two pages. A wordline's capacity equals twice that of a data page. The two pages sharing a physical page are typically written one at a time. The content of the first page being written determines one bit in the level number of a cell, and the second page determines the value of the other bit. When writing the second page, one must first read the cell to determine its current level, as the cell's final level is determined by the values of the both pages' bits. MP has several salient features: 1) when writing the first of the two "partner" pages, only the two lower levels are used, so writing is as fast as for SLC; 2) as long as the second page has not been written, reading of the first one is also fast; 3) no erasure is required when switching from SLC to MLC; and 4) Once the second page has been written, this slows down the reading of both pages, as one must determine the exact level of the cell, which may be any of the four levels.

It is important to note that both MP and our new scheme, *MMLP*, are fundamentally different from various coding schemes that are used to permit multiple writes to MLC pages between erasures. (Examples of the latter include WOM codes [9] and Rank Modulation [10].) In the other schemes, the old content is lost, whereas both MP and *MMLP* add information without harming the old one.

In the zero cell-to-cell interference architecture [38], [42], several sequences of adjacent cells in a single wordline are grouped to a single logical page and programmed together to limit cell-to-cell interference (that stems from cells being programmed after their neighbors had reached their target level). In order to handle edge interference effect, a dummy bitline is inserted in between. In addition, the MSB page is programmed to "x0" state that is below levels 2 and 3 (00 and 10). The interference from programming of adjacent wordlines shifts the "x0" state towards levels 2 and 3 and reduces the program pulses that were required for those cells. Hence, this method uses coupling interference to accelerate program operation. There are three issues regarding our paper.

1) Speedup vs. Coupling Interference: The amount of program acceleration depends on the amount of coupling interference (high interference leads to high speedup and vice versa). However, in modern charge-trap NAND flash and PCM such coupling is negligible [35], [36] and the scheme performance is same as multi-page architecture. Moreover, the impact of interference varies according to manufacturing process (e.g. insertion of insulating air gap between cells) and accurate dimensions of the floating gate device vertical layers. Therefore, such benchmark is avoided.

2) Apriori Knowledge of Wordline Data: In order to use the zero cell-to-cell interference scheme, the MSB page has to be known in order to program the corresponding cells to state "x0". In contrast, in *MMLP* the data can be programmed immediately, even when a single depth-queue 1 (DQ1) with single 4KB sector has write request.

3) A complementary Technique for Planar NAND: The zero cell-to-cell interference scheme is independent of *MMLP* and the two can be combined for further program acceleration in high-interference planar NAND devices. An example for such combination is that pages 1,2 are programmed to physical cells that were grouped by the zero cell-to-cell interference scheme. When page 3 (or 4) is applied, it is programmed first to level "0x", then returning to program neighboring wordlines and returning to move "0x" to target level.

C. Our Contributions

We propose and evaluate minimal maximum-level programming (*MMLP*), a scheme to accelerate MLC memory access. *MMLP* encodes the data such that in the kth writing of data to a cell, only the lowest k+1 levels are utilized. Therefore, cell levels are used gradually, which leads to fewer programming pulses and read reference comparisons. Unlike in previously proposed cell-sharing schemes, different data sectors are stored in different numbers of physical cells, and a cell may hold a fraction of a bit of a given data sector. Nevertheless, the exposed sector size remains unchanged and data is encoded without redundancy, so no capacity is lost. For facility of exposition, the discussion will focus on Flash. We also discuss PCM in Section V, and *MMLP* may also be adaptable to other MLC memory technologies.

D. MMLP Applications

In enterprise storage systems (and also in many cloud and server environments), the host receives an acknowledgement only when data is stored in non-volatile media, since data loss can be catastrophic. In order to handle this requirement, NAND-based SSD is often used as a write cache proxy to HDD or to All-Flash array platforms. In write-cache SSD, the bit density and write performance are the important requirements. The data is scheduled for sequential read and re-program to main storage array. *MMLP* is effective for write cache, as it has a small negative effect on the sequential read (where all cell levels are read) while improving the write performance and enabling to provide bigger cache size and a cost-effective solution. Moreover, in client SSD, a small part of the NAND is used as write cache (Turbo-write in Samsung SSDs [41]). Although DRAM is used as write buffer in client environment (where data loss is possible and an acknowledgement is sent to the host after DRAM write), it is limited due to cost issues (about 1/100 of the NAND bits). Therefore, *MMLP* is also useful in client settings.

In case that the SSD contains sufficient over-provisioning, *MMLP* can be used gradually over all wordlines and achieve sustained improvement of write and read operations. Hot-cold data separation can improve performance even further, as *MMLP* accelerates the read of part of the pages at the expense of others.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

The remainder of the paper is organized as follows. Section II presents *MMLP*, and it is evaluated in Section III. In Section IV we discuss error handling. In Section V we analyze *MMLP* benefits for PCM. Section VI offers concluding remarks.

II. Minimal Maximum-Level Programming (MMLP)

In this section we present *MMLP*. We begin with our taxonomy, move on to describe the overall approach, and then describe the exact write flow for both 4-level and 8-level cells.

A. Taxonomy

- *Sector* - a vector of binary data as viewed by the host's file system. Its size is fixed, typically 4kB. It is denoted with *D*. In the following examples, we divide the sector into small chunks (e.g. 2 bits) and encode each one independently.
- *C* - a set of selected cells of a given wordline. Those cells jointly store a given data sector (and possibly additional sectors or parts thereof). The cells that are associated with a particular address are determined by an address-to-cells (ATC) mapping function, as explained in Section B.
- *P* - the set of current (charge) levels of the cells in *C*.
- *E* - the sector's data as encoded by *MMLP*. *E* is a vector of integers that represents the levels of the sector's cells, and whose cardinality equals the number of cells in *C*.
- *MaxLevel(C)* - the current highest level of any cell in *C*.
- *Address* - host's file system address, as provided to storage device.

B. MMLP Overview

MMLP comprises address-to-cells mapping, encoder and decoder components:

- *Address-to-cells (ATC) mapping* - Given an address (as defined), ATC determines the set of memory cells (out of the wordline's cells) to which that address is mapped.
- *Encoder* - Given data sector (*D*), address, and the current levels of the mapped cells (*P*), the encoder transforms the data such that writing the encoded data *E* into the target cells causes only minimal level changes in them. A sector is stored across an address-dependent number of cells, so encoder output has variable length. (The encoder's output is the desired levels of the target cells, reflecting both the new sector being written and the existing information in those cells, which is not lost.)
- *Decoder* - Given *E*=*P* (the levels of the cells containing the sector being read) and the address, the decoder reconstructs *D* that was stored in that address. (The address is used to determine the decoder that should be used.)

Fig. 1 depicts the data flow among system's components. In each programming of a given cell, we limit its maximum target level. The 1st writing may only use levels 0 and 1; the 2nd may only use up to level 2, etc. The encoder, decoder and address-to-cells mapping are determined by the wordline and sector sizes, and by the total number of levels. We next describe the *MMLP* flow.

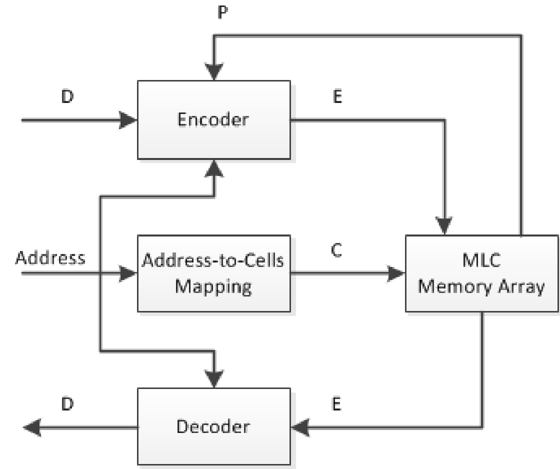


Fig. 1. Data flow among encoder, decoder, address-to-cells mapping and MLC memory array.

MMLP write flow (*D*, Address)
(1) $C \leftarrow$ ATC(Address)
(2) $P \leftarrow$ levels of memory cells C
(3) $E \leftarrow$ Encoder(D, P, Address)
(4) Write E to memory cells C
(5) Update *MaxLevel(C)* // (optional) stored metadata,
// used to accelerate reading

Fig. 2. Pseudo-code for MMLP write.

MMLP read flow (Address, *MaxLevel*)
(1) $C \leftarrow$ ATC(Address)
(2) *MaxLevel* $\leftarrow$ *MaxLevel(C)*
(3) $E \leftarrow$ perform (*MaxLevel*-1) reference comparisons
on cells C
(4) $D \leftarrow$ Decoder(E, Address)

Fig. 3. Pseudo-code for MMLP read.

C. MMLP Flow

The pseudo-code of write flow is shown in Fig. 2. In step (1), address determines target cells *C*. In step (2) *C* is read from the memory. Next, in step (3) the cell levels *P* (current content of the target physical cells *C*) along with sector data *D* (data that is to be written) and address is input to the encoder. The encoder determines the new levels of *C*'s cells. Finally, in step (4) the encoded data *E* is written to cells *C*. Note that information is added to the target cells; previously stored data is not lost.

The pseudo-code of read flow is shown in Fig. 3. In step (1), the cells to be read *C* are determined based on the address. In step (2), the maximum level of the wordline cells containing the desired sector is read (metadata). In step (3), (MaxLevel-1) reference comparisons are used to determine the cells' levels. (These are sector-wide reference comparisons, so binary search is irrelevant.) Finally, in step (4), the decoder reconstructs the original sector *D* from *E* and address. Note that a faster read scheme with fewer reference comparisons is described in subsection F. Memory erasure is not affected

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

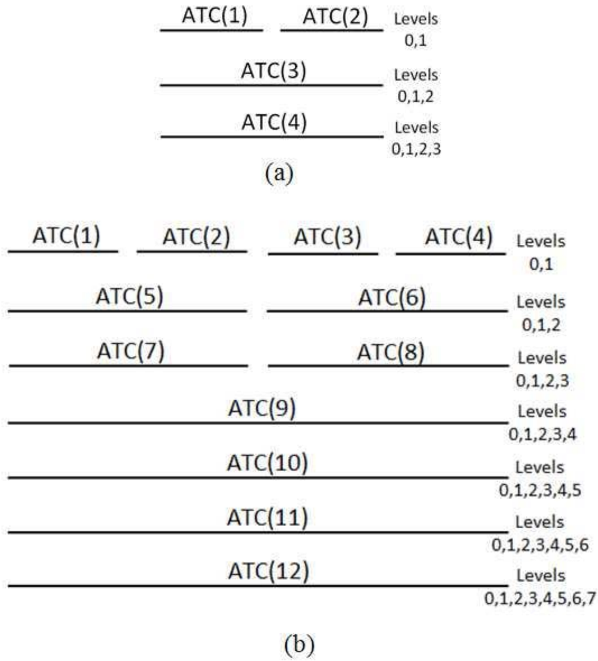


Fig. 4. Address-to-cells (ATC) mapping of data sectors to a single wordline; (a) 4-level cells (2-bits per cell, a.k.a MLC). (b) 8-level cells (3-bits per cell, a.k.a TLC).

by *MMLP*. Before proceeding to the general formulation, we next provide an example for 4-level cells with 2-bit data encoding chunks.

D. MMLP for 4-Level Cells

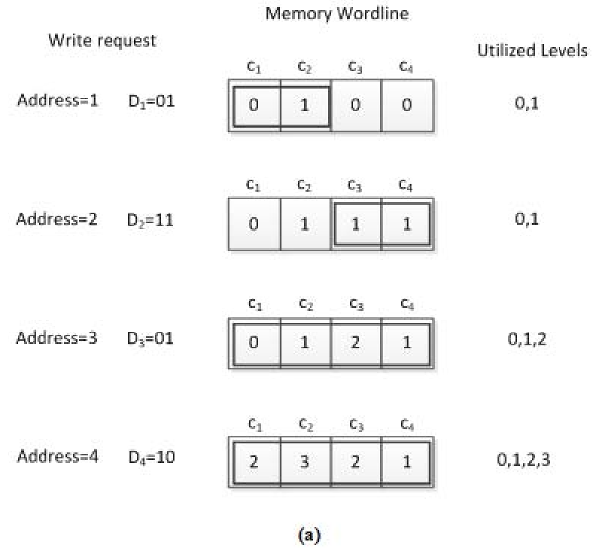
We use the following parameters, referring to a single wordline:

- 4-level MLC cells
- 2-bit sectors, and 2-bit encoding chunk size.
- Wordline (4 cells): $\{c_1,c_2,c_3,c_4\}$
- Four logical addresses (each referring to a sector), mapped to specific cells as follows: ATC(1) $= \{c_1, c_2\}$, ATC(2) $= \{c_3, c_4\}$, ATC(3) $= \{c_1, c_2, c_3, c_4\}$, ATC(4) $= \{c_1, c_2, c_3, c_4\}$.

Fig. 4 depicts the 4- and 8-level *MMLP* address to cells mapping, as well as the levels that may be used when writing each sector. Fig. 5 depicts the encoding/decoding tables, as well as a specific example of writing four 2-bit sectors of data into four 4-level (2-bit) cells.

Mapping and Writing: The first two sectors D_1 and D_2, are not encoded. Each is stored in a distinct part of the wordline, ATC(1) or ATC(2) (see rectangular frames in Fig. 5(a)), using levels {0,1}. Each bit of the 3rd sector D_3 is mapped to a distinct pair of cells, so D_3 is spread across 4 cells; it is written using levels {0,1,2}. Finally, each bit of D_4 is again mapped to a distinct pair of cells, and is written using levels {0,1,2,3} over 4 cells. Fig. 5(b) depicts the encoding tables (output cell levels according to sector data) for sectors 3 and 4. The mappings are all injective, and are thus reversible.

Any given cell is thus affected by three data bits, each from a different sector (it contains a full bit of sector 1 or 2, and half a bit of each of sectors 3 and 4).



	Page 3 data	
Cell-1/Cell-2	0	1
0-0	0-0	1-2
0-1	0-1	0-2
1-0	1-0	2-0
1-1	1-1	2-1

	Page 4 data	
Cell-1/Cell-2	0	1
0-0	0-0	2-2
0-1	0-1	2-3
1-0	1-0	3-2
1-1	1-1	3-3
1-2	1-2	1-3
0-2	0-2	0-3
2-0	2-0	3-0
2-1	2-1	3-1

(b)

Fig. 5. Example of MMLP. 4-level (2-bit) cells and D size of two bits. Each wordline has four cells, and can store four sectors. (a) Mapping of addresses to cells (see frames), utilized cell levels, and cell levels following the writing of each sector with specific data. (b) Encoding tables of addresses 3 and 4 (encoder inputs and outputs are given in cell levels): the left column depicts the current cell contents, and the others depict the new cell contents. Note that cell levels are either raised or remain unchanged.

Remark: For larger sectors and wordlines, the scheme is simply replicated for all chunks.

Example: Consider specific data being stored (Fig. 5(a)). The first data chunk is $D_1 = 01$, and is stored as is in the first two cells (one bit per cell). $D_2 = 11$, and is stored as is in the next two cells. $D_3 = 01$; it is encoded to four cells, adding 0.5 bit per cell, using only levels 0,1,2. Prior to writing it, ATC(3) cells are read ($P = 0111$), as their values affect the encoding of the new data. Using the sector-3 encoding table (Fig. 5(b)), stored data 01 and input data 0 are encoded to 01. Similarly, stored 11 and input data 1 are encoded to 21, and the cells are programmed to 0121. $D_3 = 10$, which encodes to 2321.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

Reading entails determination of the levels of all relevant cells, and decoding in descending address order. (This will be fully explained and improved in a later subsection.)

Example: Reading of 2321 would decode to $D_4 = 10$ and address 3 levels 0121 (using the sector-4 encoding table in Fig. 5(b)).

We now proceed to present the *MMLP* components, starting with the address-to-cells (ATC) mapping.

E. Address-to-Cells(ATC) Mapping

The mapping depends on the number of levels per cell. The construction is recursive.

1) 2-Level Cells (Baseline): A wordline's storage capacity is a single (data) sector. Each sector is thus stored in its own wordline.

Doubling the number of levels per cell (from *L'*/2 to *L'*)

1) Double the number of cells per wordline.
2) Double the number of levels per cell.
3) Duplicate the ATC of the pre-doubling number of levels, with each "copy" of the ATC using half of the wordline.
4) Map *L'*/2 additional sectors to the wordline such that each of those is stored across all the cells constituting this wordline.
5) Number the addresses (from scratch) in ascending order of the number of mapped cells across which a sector is spread.

Fig. 4 depicts the resulting mapping for (a) 4-level cells and (b) 8-level cells. A wordline comprising 2-level cells would only have a single sector and only ATC(1). With 4-level cells (Fig. 5(a)), a wordline would be able to store four sectors because the number of cells per wordline was doubled and each cell can now store two bits. The first two sectors, each residing in a different part of the wordline, are mapped to the locations marked ATC(1), ATC(2) (step 2). Finally, the two remaining sectors are mapped to ATC(3) and ATC(4), with each of them stored across the entire wordline (step 3).

Comparing Fig. 4(a) and (b), one can readily see that the construction for 8-level cells was obtained by placing two copies of the 4-level cell mapping side by side, then adding 8/2=4 sectors, each stored across the entire wordline (4 sectors at this point), and renumbering the sector's cells locations.

If the number of levels is not an integer power of 2, the mapping is constructed for the next integer power of two, but the number of sectors mapped in step 3 is such that the number of "layers" equals the number of levels minus one. With 6-level cells, for example, ATC(11) and ATC(12) are dropped.

F. Writing and Reading

Writing to any cell that is shared by multiple sectors must take place in ascending address order. (For simplicity, one can impose the stricter requirement of always writing sectors in ascending address order. This does not restrict the order in which data sectors are written, because there is a logical-to-physical table that can map any sector to any available wordline cells.) Also, as indicated in Fig. 5, the kth writing of data to a cell may only raise it to levels up to k. Of course, the level cannot be reduced. Consequently, writing to low addresses is much faster than to high ones, and mean writing time is reduced relative to that with a conventional mapping. Finally, note that sectors in all but the base level (low addresses) are stored across more cells than the number of bits per sector, so each cell stores a fraction of a bit of such sectors.

Reading: Naive decoding of a data sector requires the determination of the levels of all the cells across which it is stored, as well as the prior decoding of all sectors that share those cells and were written after the writing of the sector of interest. The maximum possible cell level, *MaxLevel(C)*, is optionally known for each wordline (optionally stored metadata in DRAM or elsewhere). Whenever a wordline is not fully utilized, this can reduce the required number of reference comparisons. E.g., if only sectors 1 and 2 were stored, a single reference comparison would suffice. With 8-level cells, reading addresses 8 and 10 requires at least two and four reference comparisons, respectively. Decoding employs combinational logic, so its latency is negligible (nano-seconds) relative to that of reference comparisons (tens of micro-seconds).

MMLP Read Acceleration: In sequential read, all data is required to be read from the target wordline, and sensing is performed to all cell levels. In this type of workload, there is no difference in read performance between the conventional methods and *MMLP*. In random read workload, random sectors are required to be read out of all wordline's data, and there is a performance gap between Mutli-Page and *MMLP* if the used storage space comprises over 75% of the memory capacity (while storing 3 sectors using *MMLP* in wordline that contains 4 sectors, the first two have single reference sensing, and the third has two reference sensing, similar to conventional technique). Data can be decoded by using multiple reference comparisons such that all margins between cell levels are sensed (total 3 sensing operations). However, 4-level MLC has on average 1.5 sensing operations per sector (1 for the LSB and 2 for the MSB). We suggest an improved read scheme that reduces the average number of sector's sensing operations. In the 4-level example, if only sectors 1 and 2 are stored, only one sensing between 0 and 1 is required. If the maximum level is 2, two sensing operations (reference between 0 and 1, and levels 1 and 2) for sectors 1 and 2. Sector 3 can be sensed with one sensing operation between levels 1 and 2 (note that when 2 is detected the bit of sector 3 is 1, else the bit is 0). If the maximal level is 3, sectors 3 and 4 can be decoded by two sensing operations: between levels 1 and 2, and 2 and 3. Sectors 1 and 2 would require sensing all possible margins (3 sensing operations). In summary, storage of up to two sectors per wordline result in a single sensing for read. Storing 3 sector requires $(1 + 1 + 2)/3 = 1.25$ sensing operations per sector. By using all wordline's capacity and storing all 4 sector, the random read performance degrades to $(2 + 2 + 3 + 3)/4 = 2.5$ reference comparisons.

In summary, if memory utilizes up to 75% of its storage capacity (first three sectors in each 4-sectors 4-levels-per-cell wordline) the read latency does not change in comparison to conventional mutli-page architecture. However, if 50%-75% of the storage capacity is used, the number of reference

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

comparisons does not change but read of the third sector in each wordline doubles the number of bit transmissions. If more than 75% of the memory volume is utilized, the average number of sensing operations during read increases from 1.5 to 2.5, and 50% of the sectors (sectors 3 and 4 in each wordline) require twice as many bit transmissions. Since bus speed is about 600Mbps (6.8μs per 4KB sector in 8-bit bus), the degradation is small comparing to the reference sensing latency, which is estimated to be above 50μS.

The additional bit transmissions when reading page 3 or 4 affect the available bandwidth between the Flash controller and the flash devices. It would have been unnecessary if the SSD used the industry standard technique for storing data in MLC cells. It should also be noted that since some of the sectors in the proposed scheme can span many more cells than in the industry standard scheme, this effect (known as read amplification) will be exacerbated further. A possible solution to mitigate this issue is to implement *MMLP* decoding within the NAND chip. The complexity is low (as discussed in Section III.E., VLSI implementation and overhead). In fact, MP also affects the read bandwidth, and indeed it is currently implemented on-chip in MLC NAND.

In addition, most of SSDs contain over-provisioning (7% to 37%) [44], namely extra storage space that is used for endurance enhancement and erase time hiding. *MMLP* can harness over-provisioning for partial-levels programming of memory cells while exposing full storage capacity to the host, thereby achieving sustained read acceleration. Hot-cold data separation can increase performance even further.

G. MMLP Properties

So far, we presented the ATC algorithmically, and gave the rule for level use. Also, we provided a comprehensive example, including encoder/decoder for the case of 4-level cells in the form of a small lookup table. We next establish the feasibility of *MMLP* in terms of the available storage capacity for each additional sector, discuss feasibility subject to the one-way level change constraint, and establish *MMLP*'s optimality in terms of the mean (over sectors) writing time.

Theorem 1: The available capacity of a cell for storing the data of any sector being written to it equals or exceeds the amount of data that this sector needs to store in that cell.

Proof: By induction on the number of levels per cell.

With 2-level cells, a single data sector is stored in a wordline, placing a single bit per cell. This is clearly feasible.

Now, consider $L' = (L+1)$-level cells ($\log_2(L')$ bits per cell) with $L' = 2n$, where n is natural, and assume that the theorem holds true up to $L'/2$-level cells. The first step of the extension of *MMLP* to L' levels is to double the number of cells and to duplicate the $L'/2$-level mapping, with the two replicas using disjoint sets of cells. Obviously, the capacity for storing twice the number of sectors using $L'/2$ levels is available. Next, $L'/2$ additional sectors are stored, each across the entire wordline, Since, based on the *MMLP* ATC construction, the mapping for L'-level cells uses a total of $L'/2$ sectors, it follows that each of these $L'/2$ sectors places $2/L'$ bits of information in each cell. So, storing all of them using a total of L levels is also feasible, as it requires one additional bit per cell, and this is exactly the effect of doubling the number of levels. The question, however, is whether it is possible to add these $L'/2$ sectors one by one, allowing each to use only one additional charge level.

The addition of a single level to a cell, say from m to m+1, increases its storage capacity by $\log_2(m+1) - \log_2(m)$ bits. This difference is monotonically decreasing in m. This, combined with the fact that $\log_2(L') - \log_2(L'/2) = 1 = (L'/2)\cdot(2/L')$, ensures that for every $0 < i < L'/2$, $\log_2(L'/2+i) - \log_2(L'/2) > i \cdot 2/L'$, so the total cell capacity with any number of levels exceeds that required for storing the sectors that are only allowed to use those levels. ∎

Theorem 2: MMLP minimizes the mean (over sectors) number of program pulses required for writing a data sector.

Proof: In order to minimize writing time, the use of the higher levels should be minimized. Consider the situations wherein levels $\{0, \ldots, k\}$ are in full use; i.e., their full expressive power has been exhausted. To add data, an additional level must be used. Therefore, to achieve minimum writing time, the next data must be programmed using levels $\{0, \ldots, k, k+1\}$, which lasts $T_{PROG} = T_1 + T_2 + +T_k + T_{k+1}$. This implies that the optimal algorithm must increase the number of utilized levels by one whenever data of another sector is programmed to a cell. Indeed, *MMLP* does exactly this.

It remains to be shown that it is impossible to write more than one additional sector using the single additional level. Based on the fact that $d(ln x)/dx = 1/x$, which is monotonically decreasing in x, it follows that $\log_2(L'/2+1) - \log_2(L'/2) < \log_2 e/(L'/2) < 1.5/(L'/2)$. Since each additional sector must store at least $1/(L'/2)$ bits in each of the $L'/2$ cells across which it is stored, it follows that there isn't sufficient capacity for storing more than one sector using a single additional level. It has already been established in Theorem 1 that the residual capacity after storing a single such sector decreases as one stores more sectors, so one can never store more than one additional sector using one additional level and the same number of cells. ∎

III. Evaluation

This section quantifies the performance of *MMLP* for 4-level Flash cells with 4-sectors per wordline: read latency, write latency and energy consumption. A VLSI design then serves for overhead estimation, and the benefits are estimated by way of trace-based simulation.

A. Parameter Values and Basic Expressions

Program and read time comprise, respectively, the required time to raise a cell's level from i to j, $Tp_{i\to j}$, and to sense the cell's voltage and compare it with a reference value. $Tp_{i\to j}$, equals the number of required pulses, $Np_{i\to j}$, times the sum of the durations of the program pulse (T_{pulse}) and the subsequent verification (T_{vfy}):

$$Tp_{i\to j} = Np_{i\to j}\left(T_{pulse} + T_{vfy}\right) \tag{1}$$

NAND Flash vendors do not provide data about the required number of pulses per level. We extract this data as follows:

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

TABLE II

EXPERIMENTAL PARAMETERS [4]. $Np_{i\to j}$ DENOTES THE REQUIRED NUMBER OF PULSES FOR RAISING A CELL FROM LEVEL i TO j. T_{pulse} AND T_{vfy} ARE THE DURATIONS OF PROGRAM PULSE AND SINGLE REFERENCE COMPARISONS

Parameter	Value
$Np_{0\to1}$	10 [pulses]
$Np_{0\to2}$	20 [pulses]
$Np_{0\to3}$	40 [pulses]
$Np_{1\to2}$ (=$Np_{0\to2}$-$Np_{0\to1}$)	10 [pulses]
$Np_{1\to3}$ (=$Np_{0\to3}$-$Np_{0\to1}$)	30 [pulses]
$Np_{2\to3}$ (=$Np_{0\to3}$-$Np_{0\to2}$)	20 [pulses]
T_{pulse}	10 [μS]
T_{vfy}	10 [μS]

In [4], the mean program latency of SLC is 200μS. That of MLC is 900μs (as shown in table 1). The latency is determined according to the slowest cell, which moves from the erased state (level 0) to the highest level (1 in SLC and 3 in MLC). In order to derive the duration of level 0 to 2 transition, we used the data in [39]: in its measurements reported in Fig. 1(b), MLC programming was applied and was interrupted with sudden power-off. Then, the bit-error-rate (BER) was checked. We observed major drops at 200μs and 400-500μs (fits to level 1 and 2). We assumed 10μs for program and verify pulses according to [1] and derived the corresponding number of pulses shown in Table 2. Address multiplexing delay is about 1000x smaller than program and verify times, so it is neglected.

The time required for read (T_{read}) equals T_{vfy} times the required number of reference comparisons N_r:

$$T_{read} = N_r \cdot T_{vfy}. \tag{2}$$

B. Read Latency

The tested benchmarks comprise both sequential and random read requests. Random sector read latency depends on the number of utilized levels, so it varies with capacity utilization. When all the first 3 levels are utilized, the read latency is similar for all methods. When computing averages, we assume that the low levels of all wordlines are used before beginning to use higher levels.

C. Write Latency

In Conventional Programming (CP) [11], all cells destined for level $\geq$ 1 are first programmed to level 1. Then, all cells destined for $\geq$ 2 are programmed to level 2, etc. One verification step follows each program pulse. Mean sector write latency is:

$$T_{CP} = (Np_{0\to1} + Np_{1\to2} + Np_{2\to3})\left(T_{pulse} + T_{vfy}\right) \tag{3}$$

In Multipage Programming (MP) [8], each cell stores one bit of each of two pages. The level-to-values mapping: $0 \leftrightarrow 11$, $1 \leftrightarrow 10$, $2 \leftrightarrow 00$, $3 \leftrightarrow 01$. The first page sets the least significant bit (levels 0 and 1). The second page sets the most significant bit utilizing concurrent programming to levels 2,3 with two reference comparisons (see section I.B). Prior to second page programming, MP reads the cells (one reference comparison) in order to retain the correct LSB value.

The programming time of the 1st page is $Np_{0\to1}(T_{pulse} + T_{vfy})$; that of the subsequently written 2nd page is $T_{read} + max\{Np_{0\to3}, Np_{1\to2}\}(T_{pulse}+2T_{vfy})$. The mean (over pages) write time is:

$$T_{MP} = \frac{1}{2}\big[Np_{0\to1}(T_{pulse} + T_{vfy}) + T_{read-MP} + max\{Np_{0\to3}, Np_{1\to2}\}(T_{pulse} + 2T_{vfy})\big]. \tag{4}$$

Remark: Due to limited-magnitude errors (most of the errors are caused by single-level drift, e.g. 3 to 2), the levels-to-bits translation is performed with gray code. Although in the original paper [8] the second programming is 0→2 and 1→3, there does not exist such gray code that LSB page uses levels 0 and 1 (11 and 01) and second programming results in 0→2 and 1→3. The level-to-bit mappings for such transitions are not specified in [8] or any related paper, and thus prevents the evaluation and discussion in such transitions. In practice, the MP scheme is used when MSB page is programmed with 0→3 and 1→2. See for example [38].

In *MMLP*, each cell is shared among four pages (sectors). Writing each of the first two pages takes $Np_{0\to1}(T_{pulse}+T_{vfy})$. The 3rd page has level transitions 0→1, 0→2, and 1→2, and The 4th page has level transitions 0→2, 1→3 and 2→3 (Fig. 5(b)). When writing the 3rd or 4th page, cells must first be read. This takes $T_{read} = 1 \cdot T_{vfy}$ prior to 3rd page (one reference comparison) and $T_{read} = 2 \cdot T_{vfy}$ prior to 4th. Total programming times are therefore $T_{read-p2} + max\{Np_{0\to1}, Np_{0\to2}, Np_{1\to2}\}(T_{pulse} + 2T_{vfy})$ for the 3rd page, and $T_{read-p3} + max\{Np_{0\to2}, Np_{1\to3}, Np_{2\to3}\}(T_{pulse} + 2T_{vfy})$ for the 4th one. The mean (over pages) page writing time is:

$$T_{MMLP} = \frac{1}{4}\big[2Np_{0\to1}\left(T_{pulse} + T_{vfy}\right) + T_{read-p2} + T_{read-p3} + max\left\{Np_{0\to1}, Np_{0\to2}, Np_{1\to2}\right\}\left(T_{pulse} + 2T_{vfy}\right) + max\left\{Np_{0\to2}, Np_{1\to3}, Np_{2\to3}\right\}\left(T_{pulse} + 2T_{vfy}\right)\big] \tag{5}$$

Results (Table 3): MMLP achieves 40% and 32% reduction in average program latency (speedup of 1.65x and 1.5x) over Conventional and MP, respectively. For more levels, the gap increases; E.g., a 56.75% reduction relative to MP for 8-level (3-bit) cells.

D. Trace-Based Evaluation

We estimate the expected performance of *MMLP* relative to Conventional and Multipage for actual I/O traces, using storage traces from PC-MARK [12]. See Table 4 for trace descriptions.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

TABLE III
WRITE LATENCY FOR MMLP AND PRIOR ART (CONVENTIONAL, MULTIPAGE); 4-LEVEL CELLS

Architecture	Page Write Latency [μS] 4-level cells	Average Write Latency [μS] 4-level cells
Conventional	All pages: 800	800
Multi-page	1st page: 200 2nd page: 1210	705
MMLP	1st page: 200 2nd page: 200 3rd page: 610 4th page: 920	482.5

TABLE IV
STORAGE TRACES

Trace	Description
windefend	Windows defender
gaming	Playing a game
Impacts	Importing pictures
Vistastart	Windows Vista start
videdit	Video Editing
medcent	MS media center
medplayer	Playing music
Appload	Application loading

1) Methodology: Our methodology is as follows. First, we use our previously obtained analytical results to express speedup vs. R/W ratio for a given set of parameter values. Next, for each trace, we count the numbers of reads and writes to obtain its R/W ratio. Finally, we use the analytical results for that R/W ratio as our estimate.

The speedup in average memory access time is given by:

$$Speedup = \frac{N_R T_R^{old} + N_W T_W^{old}}{N_R T_R^{new} + N_W T_W^{new}} \quad (6)$$

Where:

- N_R, N_W – Numbers of read and write operations.
- T_R^{old}, T_W^{old} – Conventional's durations.

The energy reduction factor (larger is better) is given by:

$$Speedup = \frac{N_R P_R^{old} T_R^{old} + N_W P_W^{old} T_W^{old}}{N_R P_R^{new} T_R^{new} + N_W P_W^{new} T_W^{new}} \quad (7)$$

Where:

- P_R^{old}, P_W^{old} – Read and write power of Conventional.
- P_R^{new}, P_W^{new} – Read and write power of MP or *MMLP*.
- T_R^{new}, T_W^{new} – The duration of read and write in MP or *MMLP* architectures, as summarized in Table 3, with adjustments to 2x (for MP) and 4x (for *MMLP*) shorter bitlines.

The speedup gain is with respect to conventional settings. *MMLP* achieves on average 800/482=1.65 write speedup improvement (see table 3). When comparing to MP, speedup is 705/482=1.46. *MMLP* can achieve up to 4x improvement in read-speedup if memory is not fully utilized (in conventional settings, all references are read with additional processing) and for shorter bitlines (same as MP [8] due to increased cell parallelism) as explained in Section III.D. As expected, the speedup ranges up to 2x in MP and 1.65x to 4x in *MMLP*, according to read/write ratio. The degree of device utilization (and corresponding read speedup) was according to the selected workload. We used 32GB SSD with 7% over-provisioning in our simulator. In benchmarks with small write volume such as gaming and imppict, storage utilization is less than 75% of total capacity, whereas benchmarks with heavy writes such as video-edit reach full capacity utilization but also fewer have small or sequential read to a instructions. Note that heavy writes lead to full utilization but also to a small amount of degraded reads, whereas fewer writes did not reach full utilization and was able to exploit read acceleration. There is an additional case wherein the device is fully utilized and random read-only workloads would lead to performance degradation, but we did not find such workload in PC-Mark package. Note that sequential read of all cell levels when the SSD is fully utilized has about the same performance in *MMLP* and MP.

Fig. 6 depicts speedup and energy vs. R/W ratio, showing the location of each trace on the curve based on its R/W ratio. Consider 4-level-cell NAND Flash. With Conventional, write is x18 slower than read (900μs vs. 50μs). Write energy is 10-630 times that of read [4]. In our comparison, we assume write power (not energy) to be 2x that of read (for all schemes), which matches most of the samples in [4].

MP's wordlines are 2x longer and bitlines are 2x shorter than Conventional's [8], This becomes 4x for *MMLP* due to increasingly parallel cell access. Write time is unaffected, but read duration is reduced commensurately due to reduced bitline precharge and discharge times during reference comparison.

Results show *MMLP*'s advantage over MP to be 1.5x – 2x in performance, and 2x – 2.7x in energy consumption.

a) Array-level and wordline/bitline implications: MMLP increases the number of cells that may be accessed in parallel in order to write (or read) a given data sector. This by itself is of little value, because commensurately more cells must be accessed in order to write or read a data sector. However, as pointed out for Multipage [8], the ability to meaningfully lengthen the wordline (so as to increase the number of bitlines and access all cells in parallel) permits a commensurate shortening of the bitlines. Shortening a bitline reduces its capacitance, permitting faster read access. In other words, this accelerates reading, as well as the verification step when writing. The available on-chip parallelism is thus put to good use in order to expedite access to even a single data sector.

b) Delay and energy parameters: Both programming and reading of Flash/PCM occur as a sequence of discrete steps (program pulses + verification for write, and reference comparisons for Read). Both latency and energy are linear in the number of steps, and their relative values are largely indepen-

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

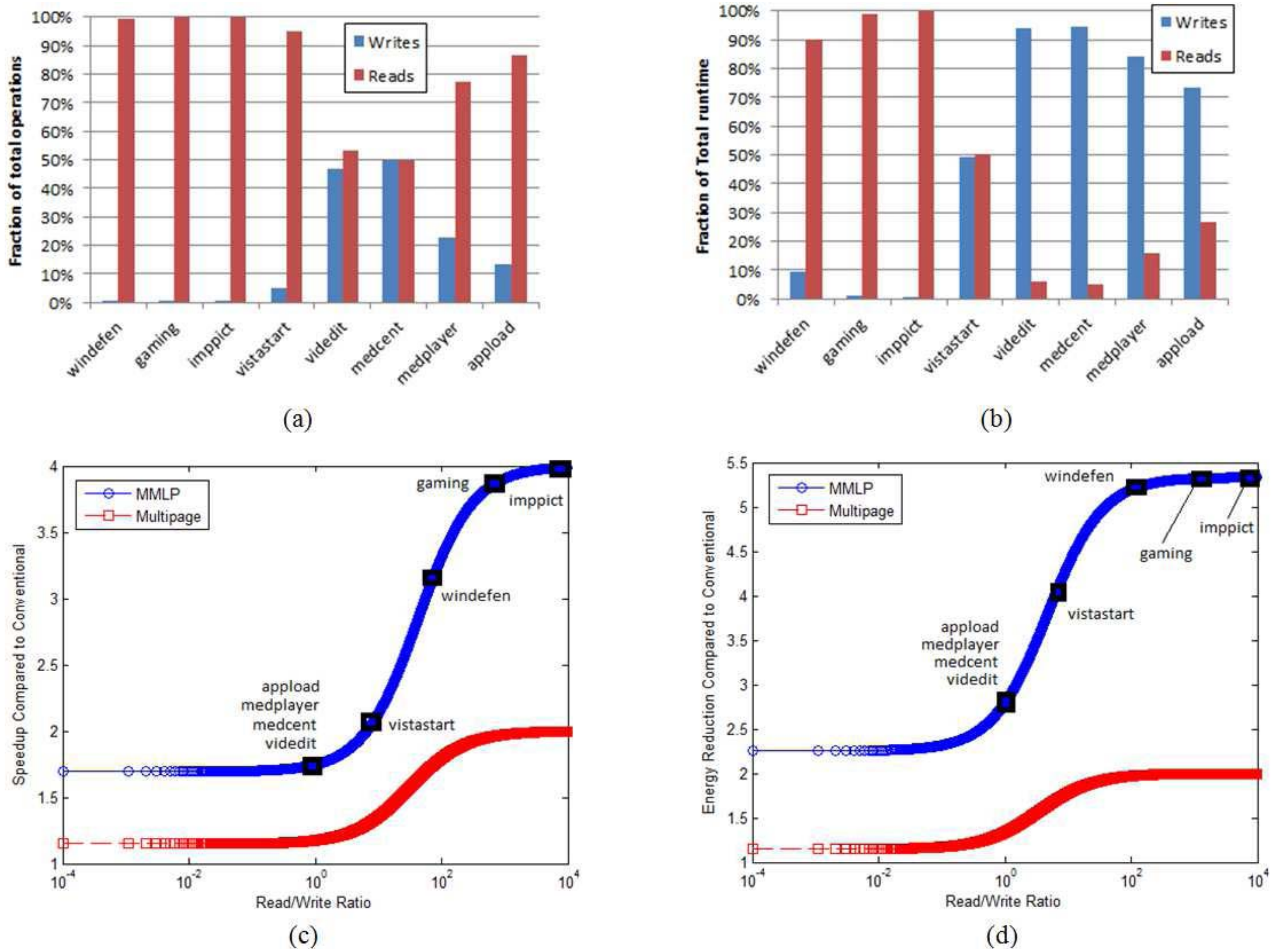


Fig. 6. Top: (a) Read and write fraction out of total benchmark operations. (b) Read and write duration fraction out of total runtime. Write is 18x slower than read. Bottom: Speedup (c) and energy reduction (d) of Multipage and *MMLP* relative to Conventnional vs. R/W ratio. Benchmarks are placed based on their R/W ratio. (baseline write is 18x slower than read, and consumes 2x power than read.)

dent of circuit parameters. Therefore, our evaluation relative to Conventional and Multi-page architectures is meaningful. We must nonetheless use actual time values to determine the relative weights for read and write.

c) Results: In Fig. 6, one can see a dramatic advantage of both MP and *MMLP* over Conventional, which becomes more pronounced as the number of levels per cell increases. The difference between *MMLP* and MP grows as R/W ratio increases, expressing the fact that the indirect benefit of shortening the bitlines is apparently very substantial. *MMLP* performance advantage over Multipage ranges from 1.5x to 2x in performance, and from 2x to 2.7x in energy consumption.

E. VLSI Implementation and Overhead

MMLP comprises encoder, decoder, and address-to-cell mapping modules, and a small table storing *MaxLevel(C)* for each group of sectors. We implemented the modules in Verilog HDL, and synthesized them with Synopsis Design Compiler in IBM 65nm process technology. The latency of each module is about 0.5ns in 65nm technology, which is negligible relative to write/read latency. *MMLP*'s circuit area is $625\mu m^2$, 0.003% of a typical 144mm^2 die. Total power consumption, including leakage and dynamic power is $584\mu W$, nearly 1% of a typical 50mW average program power. Overhead, both latency and area, is thus negligible.

F. Energy Savings

The reduction in write/read latency leads to corresponding energy savings. A typical MLC Flash has power consumption of 50mW for write and 30mW for read. While read energy reduction depends on memory occupancy, write energy reduction is expressed when writing any erased block. With 4-level cells, energy reduction relative to Multipage is 32%-50%.

IV. Error Handling and Endurance

A. Inter-Cell Interference

Traditional planar NAND devices exhibit inter-cell coupling interference which may render re-write schemes impractical [34]. However, new NAND products (since 2014) incorporate 3D structure, and are based on charge-trap similarly

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

to SONOS structure [35]. The inter-cell interference in those products is reduced by 84% [36], which enables the usage of *MMLP*. Moreover, in emerging memory technologies such as PCM and ReRAM, the inter-cell interference phenomena have not been observed, which makes *MMLP* feasible for them [37].

Furthermore, *MMLP* can be combined with other interference-mitigation programming techniques such as zero cell-to-cell interference programming [38], [42]). See Sec. I.B. for detailed explanation of the combination.

Sequential order of page programming is also used in planar NAND, where inter-cell interference is common. However, in modern 3D Flash with a charge-trap storage mechanism, the wordlines do not have to be written row-by-row. Furthermore, in emerging memory technologies such as PCM, such requirements are unlikely [37].

B. Data Reliability During Read-Back

Flash data errors are characterized as low-magnitude shifts, which cause the level of a cell to change to an adjacent level. They are often caused by continuous charge leakage or program overshoots.

Due to the storage of partial bits per cell, a reduction of a cell's level by one may result in multiple data bit errors. Nonetheless, any given cell is affected by at most one bit of any given sector, so a single cell error causes at most a single erroneous bit in each data sector.

We analyzed the reliability impact of *MMLP* in page data read. We tested all possible single-level drifts and calculated the average number of affected bits. Our results show that random read of *MMLP* pages results in 1.18 bit errors for single cell level drift (18% increase in BER) if all four pages are stored or 1.09 bit errors (9% error increase) if three pages are stored. We provide a comprehensive analysis in Fig. 7. When all four pages are stored, there are transitions that lead to a single error in more than a single page. However, this can be mitigated by using symbol-level ECC as described in the next section. Observing the encoding table in Fig. 5(b), the resulting level distribution in the cells when full storage capacity is used is equal for all levels (i.e., the number of cells in any specific level is equal to the number in other levels). Therefore, the average level-error rate is not affected by *MMLP*.

C. Mitigation of Read Errors in MMLP

MMLP is used to accelerate write. Typically, existing SSDs have part of their volume dedicated to serve has a write cache (Turbo-write [41]) or are used in their entirety as cache. In such configuration, cache is scheduled to be sequentially read and re-written to the main array. When *MMLP* is used in write cache storage area, sequential read for re-program performs read to all pages in each wordline, hence the controller can decode the exact cell levels and therefore cell-wise ECC (Reed Solomon (RS) code) can be used. RS code is meant for correcting level (symbol) errors and is more efficient than bit-wise ECC (BCH or LDPC codes). RS is an MDS code, and has small implementation area and complexity and also can use the same decoder resources of BCH (BM decoder). Furthermore, write-cache is not expected to have retention errors, which are most of the errors observed in NAND flash.

Another approach is to protect the data of any given sector by using well known ECC. Also, assuming that no errors occurred until the final sector was written to a given wordline, ECC that protects the cell states following the writing of the final sector will guarantee correct decoding of all sectors. Thus, all sectors but the last can be programmed without ECC protection, and the last sector is programmed with ECC, thereby reducing the required amount of ECC redundancy. Integrating ECC with last-sector programming is a topic for future research.

D. Endurance

Endurance is defined as permissible number of erasure cycles that a cell may undergo before its data becomes unrecoverable. The scheme proposed in this paper does not affect endurance directly. However, whenever *MMLP* is used gradually across all memory cells in a storage system and there is no over-provisioning, garbage collection (GC) and write amplification may be affected due to possible change in the number of invalid wordlines per victim block before erase. The exact impact is highly dependent on the workload. When *MMLP* is utilized in the recommended write-cache SSD configuration (as whole SSD in storage server or part of the SSD as Turbo-write [41], see Section I.D). the GC collects full blocks and re-programs them to the main array, so GC and write amplification are not affected.

V. Phase-Change Memory

In this section, we assess the benefits of *MMLP* to PCM programming. We first briefly review PCM's salient features.

A. PCM Programming Algorithm

PCM can be changed on a bit basis, unlike Flash that has to be block erased [13]. The programming algorithm of MLC PCM includes program pulses and also read verify in some cases, similarly to Flash. The programming duration of MLC PCM depends on the cell's current level. As in Flash, the impact of a given program pulse on different cells may vary, both due to process variation and to the cell's current state. When programming many cells concurrently, the worst case is likely to exist and determines the latency.

MLC PCM programming is usually either "RESET to SET" (R2S) or "SET to RESET" (S2R) [14] (Fig. 8). In R2S, an initial reset pulse is applied, bringing the cell to its lowest level, and subsequent programming pulses raise its level. Similarly, S2R comprises an initial set pulse that raises the cell level to the maximum, and subsequent pulses lower the level to the target one. S2R and R2S exhibit a speed-reliability trade-off. While S2R is faster than R2S, the margin between adjacent levels is narrower, making S2R more error prone. Therefore, R2S is more common. Prior to writing, it has been proposed to perform data read [15] in order to obviate the need for additional pulses to cells that already contain data. However, the benefit of so doing when programming an entire sector is

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

			page errors				
Original	Error	#cell drifts	page 1	page 2	page 3	page 4	#bit errors
0-0	0-1	1	0	1	0	0	1
0-0	1-0	1	1	0	0	0	1
0-0	1-1	2	1	1	0	0	2
0-1	0-2	1	0	0	1	0	1
0-1	0-0	1	0	1	0	0	1
0-1	1-1	1	1	0	0	0	1
0-1	1-2	2	0	1	1	0	2
0-1	1-0	2	1	1	0	0	2
1-0	1-1	1	0	1	0	0	1
1-0	2-0	1	0	0	1	0	1
1-0	0-0	1	1	0	0	0	1
1-0	2-1	2	0	1	1	0	2
1-0	0-1	2	1	1	0	0	2
1-1	1-2	1	1	1	1	0	3
1-1	1-0	1	0	1	0	0	1
1-1	2-1	1	0	0	1	0	1
1-1	0-1	1	1	0	0	0	1
1-1	2-2	2	1	1	0	1	3
1-1	2-0	2	0	1	1	0	2
1-1	0-2	2	1	0	1	0	2
1-1	0-0	2	1	1	0	0	2
1-2	1-3	1	0	0	0	1	1
1-2	1-1	1	1	1	1	0	3
1-2	2-2	1	0	0	1	1	2
1-2	0-2	1	0	1	0	0	1
1-2	2-3	2	0	1	1	1	3
1-2	2-1	2	1	1	0	0	2
1-2	0-3	2	0	1	0	1	2
1-2	0-1	2	0	1	1	0	2
0-2	0-3	1	0	0	0	1	1
0-2	0-1	1	0	0	1	0	1
0-2	1-2	1	0	1	0	0	1
0-2	1-3	2	0	1	0	1	2
0-2	1-1	2	1	0	1	0	2
2-0	2-1	1	0	1	0	0	1
2-0	3-0	1	0	0	0	1	1
2-0	1-0	1	0	0	1	0	1
2-0	3-1	2	0	1	0	1	2
2-0	1-1	2	0	1	1	0	2
2-1	2-2	1	1	1	1	1	4
2-1	2-0	1	0	1	0	0	1
2-1	3-1	1	0	0	0	1	1
2-1	1-1	1	0	0	1	0	1
2-1	3-2	2	0	1	1	1	3
2-1	3-0	2	0	1	0	1	2
2-1	1-2	2	1	1	0	0	2
2-1	1-0	2	0	1	1	0	2
2-2	2-3	1	0	1	0	0	1
2-2	2-1	1	1	1	1	1	4
2-2	3-2	1	1	0	0	0	1
2-2	1-2	1	0	0	1	1	2
2-2	3-3	2	1	1	0	0	2
2-2	3-1	2	1	1	1	0	3
2-2	1-3	2	0	0	1	0	1
2-2	1-1	2	1	1	0	1	3
2-3	2-2	1	0	1	0	0	1
2-3	3-3	1	1	0	0	0	1
2-3	1-3	1	0	1	1	0	2
2-3	3-2	2	1	1	0	0	2
2-3	1-2	2	0	1	1	1	3
3-2	3-3	1	0	1	0	0	1
3-2	3-1	1	0	1	1	0	2
3-2	2-2	1	1	0	0	0	1
3-2	2-3	2	1	1	0	0	2
3-2	2-1	2	0	1	1	1	3
3-3	3-2	1	0	1	0	0	1
3-3	2-3	1	1	0	0	0	1
3-3	2-2	2	1	1	0	0	2
1-3	1-2	1	0	0	0	1	1
1-3	2-3	1	0	1	1	0	2
1-3	0-3	1	0	1	0	0	1
1-3	2-2	2	0	0	1	0	1
1-3	0-2	2	0	1	0	1	2
0-3	0-2	1	0	0	0	1	1
0-3	1-3	1	0	1	0	0	1
0-3	1-2	2	0	1	0	1	2
3-0	3-1	1	0	1	0	0	1
3-0	2-0	1	0	0	0	1	1
3-0	2-1	2	0	1	0	1	2
3-1	3-2	1	0	1	1	0	2
3-1	3-0	1	0	1	0	0	1
3-1	2-1	1	0	0	0	1	1
3-1	2-2	2	1	1	1	0	3
3-1	2-0	2	0	1	0	1	2
	Total	120					142

			page errors			
Original	Error	#cell drifts	page 1	page 2	page 3	#bit errors
0-0	0-1	1	0	1	0	1
0-0	1-0	1	1	0	0	1
0-0	1-1	2	1	1	0	2
0-1	0-2	1	0	0	1	1
0-1	0-0	1	0	1	0	1
0-1	1-1	1	1	0	0	1
0-1	1-2	2	0	1	1	2
0-1	1-0	2	1	1	0	2
1-0	1-1	1	0	1	0	1
1-0	2-0	1	0	0	1	1
1-0	0-0	1	1	0	0	1
1-0	2-1	2	0	1	1	2
1-0	0-1	2	1	1	0	2
1-1	1-2	1	1	1	1	3
1-1	1-0	1	0	1	0	1
1-1	2-1	1	0	0	1	1
1-1	0-1	1	1	0	0	1
1-1	2-2	2	1	1	1	3
1-1	2-0	2	0	1	1	2
1-1	0-2	2	1	0	1	2
1-1	0-0	2	1	1	0	2
1-2	1-1	1	1	1	1	3
1-2	2-2	1	0	0	0	0
1-2	0-2	1	0	1	0	1
1-2	2-1	2	1	1	0	2
1-2	0-1	2	0	1	1	2
0-2	0-1	1	0	0	1	1
0-2	1-2	1	0	1	0	1
0-2	1-1	2	1	0	1	2
2-0	2-1	1	0	1	0	1
2-0	1-0	1	0	0	1	1
2-0	1-1	2	0	1	1	2
2-1	2-2	1	1	1	0	2
2-1	2-0	1	0	1	0	1
2-1	1-1	1	0	0	1	1
2-1	1-2	2	1	1	0	2
2-1	1-0	2	0	1	1	2
	Total	52				57

Fig. 7. Cell-level drift vs. Page bit-errors in 4-level MMLP. Left: all four pages are programmed. Right: three pages are programmed.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

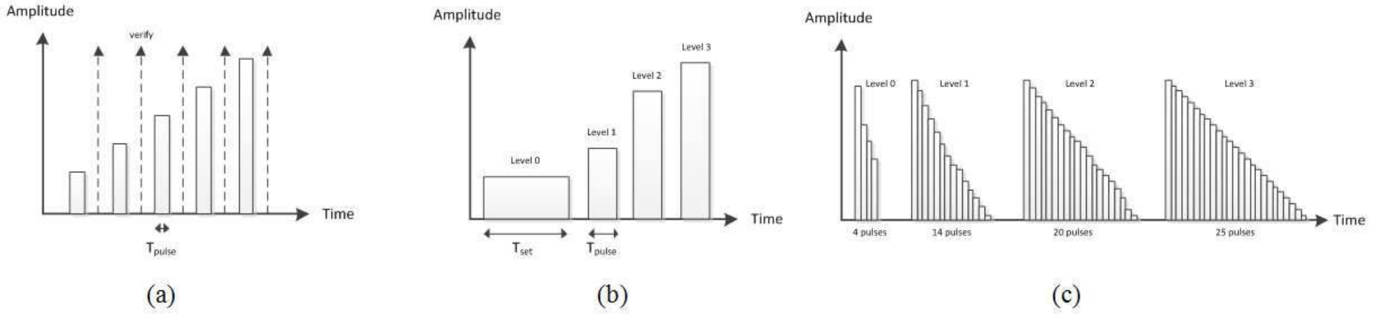


Fig. 8. Pulse duration and magnitude in Flash and PCM MLC programming. (a) Constant pulse width, with read verify after each pulse in Flash (ISPP - Incremental Step Pulse Programming). (b) Set to Reset (S2R) accumulated pulses in PCM. While programming a cell to high level, previous levels have to be programmed. (c) Reset to Set (R2S) pulses to program for each level in PCM. R2S are more accurate due to shorter pulses and are more popular [14]. Fig. (b) and (c) were adopted from [14].

TABLE V
RESET TO SET (R2S) PCM PROGRAMMING

Level	Value [ns]
00 – Reset	75
01 – State	210
11 – State	270
10 – Set	420

TABLE VI
SET TO RESET (S2R) PCM PROGRAMMING

Level	Value [ns]
00 – Set	200
01 – State	250
11 – State	300
10 – Reset	350

questionable because the worst case (over cells) matters, and at least one cell is likely to require the full swing.

Further PCM programming optimization schemes include pulse optimization [16], [17], mapping [18] and write suspension [19]. Pulse optimization can complement *MMLP* and enhance it. Write suspension and cancellation can mitigate the write bottleneck by using additional buffers, but are limited to buffer size and can cause data loss in the event of power failure. Since there is no standard PCM write algorithm yet, we chose the basic R2S and S2R approaches. There are many others [2], and their consideration is a topic for future work.

B. PCM Programming Latency

Tables 5, 6 provide typical timings for R2S and S2R, respectively [14]. Program time to high level includes the programming to initial and intermediate levels.

Let $Tp_{i\to j}$ denotes the program time from level i to to level j. Tp_0 denotes the programming to the base level (reset in R2S and set in S2R). Programming time is determined by worst-case cell transition, which is likely to be full transition from the base level to the highest level when programming a sufficient amount of data (e.g. 32 bits). With a Conventional four-level cell and Conventional scheme:

$$T_{Conv.} = Tp_0 + max\,\{Tp_{0\to 1}, Tp_{0\to 2}, Tp_{0\to 3}\} \tag{8}$$

which is 420ns for R2S and 350ns for S2R. Assuming T_{read}=50 nS, Multipage programming (MP) yields:

$$T_{MP} = \frac{1}{2}\left[Tp_0 + Tp_{0\to 1} + T_{read} + Tp_0 + max\,\{Tp_{1\to 2}, Tp_{0\to 3}\}\right] \tag{9}$$

This is 340ns (325ns) for R2S (S2R), a 25% (14%) improvement over Conventional. With *MMLP*:

$$T_{MMLP} = \frac{1}{4}\left[Tp_0 + Tp_{0\to 1} + Tp_0 + Tp_{0\to 1} + T_{read} + Tp_0 + max\,\{Tp_{0\to 1}, Tp_{0\to 2}, Tp_{1\to 2}\} + T_{read} + Tp_0 + max\,\{Tp_{0\to 2}, Tp_{1\to 3}, Tp_{2\to 3}\}\right] \tag{10}$$

This is 265ns (300ns) for R2S (S2R), a 24% (8%) and 37% (15%) improvement over Conventional and MP, respectively.

C. Partial Overwriting of Data in PCM Applications

When programming 4-level MLC each one of levels is mapped to two bits from different pages (e.g. 11, 01, 00, 10). Re-writing LSB page would result in moving levels 0↔1 and 2↔3. Re-writing MSB moves levels 0↔3 and 1↔2.

In *MMLP*, once all pages have been programmed, re-writing is possible according to the following:

- Re-writing of page 4 includes switching levels 3↔2, 3↔1 and 2↔0 according to the table in Fig. 5(b).
- Re-writing of page 3 includes reading page 4 (and page 3 which is also performed in the conventional MLC case) and re-encoding. In cells whose page 4 bit is 0, rewriting page 3 is straightforward (0↔1, 0↔2 and 1↔2). Else, additional level transformations are: 2↔3, 1↔3. The difference from conventional is that 0↔3 transition is not required, which is the slowest transition.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

- Re-writing of pages 1 or 2 includes reading pages 4,3 and the page to be updated. Unlike conventional MLC rewrite (where all wordline cells are updated and have level transitions), in *MMLP* only half of the wordline cells are modified (either page 1 or 2), but modification includes all possible level transitions (6 transitions) as opposed to 2 transitions in MLC.

In summary, *MMLP* enables update of pages 3,4 with partial level transitions (comparing to full write) or updates of pages 1,2 with cell level transitions in only half of the wordline (comparing to all wordline cells in conventional settings).

The number of level transitions during rewrite of cells that are encoded in *MMLP* is higher than conventional MLC update, but it is lower than full write in pages 3,4 and affects fewer cells in pages 1,2. Moreover, *MMLP* enables gradual filling of the memory levels, and increases the probability that page update would occur in a wordline that is partially programmed, which results in faster programming and less cell degradation.

VI. Conclusions

We proposed *MMLP* – minimal maximum level programming, a memory architecture that enhances write and read performance while saving energy in MLC memory. *MMLP* Minimizes the mean sector-writing time, shortening it by at least 32% relative to prior art for 4-level NAND Flash cells. Whenever the memory is underutilized (or with high over-provisioning), Read is accelerated by reducing the number of reference comparisons, based on a priori knowledge of the highest programmed level in a sector.

Our focus here has been on NAND Flash, with an outline of adaptation to Phase-Change memory. However, *MMLP* may be beneficially adaptable to additional memory technologies.

We proved that *MMLP* is feasible in terms of storage capacity, and moreover verified that at least for cells with up to 16 levels, an encoding exists such that, when programmed in address order, writing an additional data sector never requires the lowering of any cell's charge level. However, the determination whether this holds for any number of levels is left for future research. Also, the encoding tables that we generated grow rapidly with an increase in the number of levels. The construction of optimal or even slightly sub-optimal (in terms of cell capacity utilization) compact encoders is another interesting research topic.

MMLP results in variability of write/read time between sectors. Exploiting this for performance optimization and hot-cold data separation is a topic for future research. Additional research directions include low-complexity encoding/decoding for a large number of levels, combination with ECC, and further combination with high-speed programming techniques.

References

[1] J. Brewer and M. Gill, Eds., *Nonvolatile Memory Technologies With Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices* (IEEE Press Series on Microelectronic Systems). Hoboken, NJ, USA: Wiley, 2008.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. ISCA*, 2009, pp. 2–13.

[3] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of NAND flash memory," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*, 2012, p. 2.

[4] L. M. Grupp *et al.*, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. MICRO*, Dec. 2009, pp. 24–33.

[5] *K9NBG08U5M 4Gb*8 Bit NAND Flash Memory Data Sheet*, Samsung Electronics, Suwon, South Korea, 2013.

[6] *K9GAG08U0M 2Gb*8 Bit NAND Flash Memory Data Sheet*, Samsung Electronics, Suwon, South Korea, 2013.

[7] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "FlexFS: A flexible flash file system for MLC NAND flash memory," in *Proc. USENIX Annu. Tech. Conf.*, 2009, pp. 1–14.

[8] K. Takeuchi, T. Tanaka, and T. Tanzawa, "A multipage cell architecture for high-speed programming multilevel NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 33, no. 8, pp. 1228–1238, Aug. 1998.

[9] R. L. Rivest and A. Shamir, "How to reuse a 'write–once' memory," *Inf. Control*, vol. 55, nos. 1–3, pp. 1–19, Oct./Dec. 1982.

[10] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[11] K.-D. Suh *et al.*, "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," in *Proc. ISSCC*, Feb. 1995, pp. 128–129.

[12] *PCMARK-VANTAGE, White Paper V1.0*. Futuremark, Finland, 2013.

[13] F. Bedeschi *et al.*, "A bipolar-selected phase change memory featuring multi-level cell storage," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 217–227, Jan. 2009.

[14] M. Joshi, W. Zhang, and T. Li, "Mercury: A fast and energy-efficient multi-level cell based phase change memory system," in *Proc. IEEE High Perform. Comput. Archit. (HPCA)*, Feb. 2011, pp. 345–356.

[15] J. Hu, W.-C. Tseng, C. J. Xue, Q. Zhuge, Y. Zhao, and E. H.-M. Sha, "Write activity minimization for nonvolatile main memory via scheduling and recomputation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 4, pp. 584–592, Apr. 2011.

[16] T. Nirschl *et al.*, "Write strategies for 2 and 4-bit multi-level phase-change memory," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2007, pp. 461–464.

[17] J.-T. Lin, Y.-B. Liao, M.-H. Chiang, and W.-C. Hsu, "Operation of multi-level phase change memory using various programming techniques," in *Proc. IEEE Int. Conf. IC Design Technol.*, May 2009, pp. 199–202.

[18] H. Yoon, N. Muralimanohar, J. Meza, O. Mutlu, and N. P. Jouppi, "Data mapping for higher performance and energy efficiency in multi-level phase change memory," in *Proc. NVMW*, 2012, pp. 1–2.

[19] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montaño, "Improving read performance of phase change memories via write cancellation and write pausing," in *Proc. IEEE High Perform. Comput. Archit. (HPCA)*, Jan. 2010, pp. 1–11.

[20] G. J. Hemink, T. Tanaka, T. Endoh, S. Aritome, and R. Shirota, "Fast and accurate programming method for multi-level NAND EEPROMs," in *Proc. Symp. VLSI Technol.*, Jun. 1995, pp. 129–130.

[21] M. Grossi, M. Lanzoni, and B. Ricco, "Program schemes for multilevel flash memories," *Proc. IEEE*, vol. 91, no. 4, pp. 594–601, Apr. 2003.

[22] H. Kim *et al.*, "A 159 mm^2 32 nm 32 Gb MLC NAND-flash memory with 200 MB/s asynchronous DDR interface," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2010, pp. 442–443.

[23] T. Tanaka *et al.*, "A quick intelligent page-programming architecture and a shielded bitline sensing method for 3 V-only NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 29, no. 11, pp. 1366–1373, Nov. 1994.

[24] T. Hara *et al.*, "A 146 mm^2 8 Gb NAND flash memory with 70 nm CMOS technology," in *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2005, pp. 44–45.

[25] S.-H. Chang *et al.*, "A 48 nm 32 Gb 8-level NAND flash memory with 5.5 MB/s program throughput," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2009, pp. 240–241.

[26] A. Berman and Y. Birk, "Constrained flash memory programming," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul./Aug. 2011, pp. 2128–2132.

[27] K. Takeuchi *et al.*, "A 56 nm CMOS 99 mm^2 8 Gb multi-level NAND flash memory with 10MB/s program throughput," in *Proc. ISSCC*, Feb. 2006, pp. 507–516.

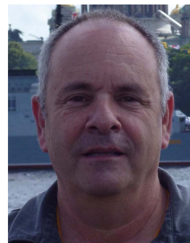
[28] C. Trinh *et al.*, "A 5.6 MB/s 64Gb 4b/cell NAND flash memory in 43 nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2009, pp. 246–247.

[29] J. Hwang *et al.*, "A middle-1X nm NAND flash memory cell (M1X-NAND) with highly manufacturable integration technologies," in *Proc. IEEE Int. Electron Device Meeting (IEDM)*, Dec. 2011, pp. 9.1.1–9.1.4.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.

[30] K. Imamiya *et al.*, “A 130 mm^2 256 Mb NAND flash with shallow trench isolation technology,” in *Proc. ISSCC*, Feb. 1999, pp. 112–113.
[31] T. Futatsuyama *et al.*, “A 113 mm^2 32 Gb 3b/cell NAND flash memory,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2009, pp. 242–243.
[32] Y. Li *et al.*, “128 Gb 3b/cell NAND flash memory in 19 nm technology with 18 MB/s write rate and 400 Mb/s toggle mode,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2012, pp. 436–437.
[33] A. Berman and Y. Birk, “Minimal maximum-level programming: Faster memory access via multi-level cell sharing,” in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 2705–2710.
[34] J.-D. Lee *et al.*, “Effects of floating-gate interference on NAND flash memory cell operation,” *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, May 2002.
[35] J. Jang *et al.*, “Vertical cell array using TCAT(Terabit Cell Array Transistor) technology for ultra high density NAND flash memory,” in *VLSI Symp. Tech. Dig.*, Jun. 2009, pp. 192–193.
[36] K.-T. Park *et al.*, “Three-dimensional 128 Gb MLC vertical NAND flash-memory with 24-WL stacked layers and 50 MB/s high-speed programming,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 334–335.
[37] A. Athmanathan, M. Stanisavljevic, N. Papandreou, H. Pozidis, and E. Eleftheriou, “Multilevel-cell phase-change memory: A viable technology,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 1, pp. 87–100, Mar. 2016.
[38] K.-T. Park *et al.*, “A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories,” *IEEE J. Solid-State Circuits*, vol. 43, no. 4, pp. 919–928, Apr. 2008.
[39] H.-W. Tseng, L. Grupp, and S. Swanson, “Understanding the impact of power loss on flash memory,” in *Proc. Design Autom. Conf. (DAC)*, 2011, pp. 35–40.
[40] X. Dong and Y. Xie, “AdaMS: Adaptive MLC/SLC phase-change memory design for file storage,” in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2011, pp. 31–36.
[41] “TurboWrite technology,” Samsung Electronics, Suwon, South Korea, White Paper, 2013. [Online]. Available: http://www.samsung.com/cz/business-images/resource/white-paper/2014/02/Whitepaper-Samsung-SSD-TurboWrite-0.pdf
[42] L. B. Fernandes, “Dynamically configurable MLC state assignment,” U.S. Patent 8 467 242 B2, Jun. 18, 2013.
[43] C. Yu, O. Mutlu, E. F. Haratsch, and K. Mai, “Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation,” in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 123–130.
[44] K. Smith, “Understanding SSD over provisioning,” in *Proc. Flash Memory Summit Conf.*, 2012, pp. 1–16.

Amit Berman received the Ph.D. degree in electrical engineering from the Technion–Israel Institute of Technology, Haifa, Israel, in 2013. He is currently with Samsung Electronics Memory Division as an Algorithms Team Manager, and also serves as an Adjunct Lecturer with the Electrical Engineering Department, Technion–Israel Institute of Technology. He has authored over 20 research papers in leading venues and holds over 10 U.S./PCT issued, and pending patents. He is a recipient of several awards, including the Hershel Rich Innovation Award, the Mitchell Grant, the HPI Fellowship and International Solid–State Circuits Conference Recognition.



Yitzhak Birk (M’82–SM’02) received the B.Sc. (*cum laude*) and M.Sc. degrees from the Technion–Israel Institute of Technology, Haifa, Israel, in 1975 and 1982, respectively, and the Ph.D. degree from Stanford University, Stanford, CA, in 1987, all in electrical engineering. He was a Research Staff Member with IBM’s Almaden Research Center. He has been with the Faculty of the Electrical Engineering Department, Technion–Israel Institute of Technology, since 1991, and heads its Parallel Systems Laboratory.

His research interests include computer and communication systems, with much attention to the storage subsystem and to the interplay between storage and communication. The true application requirements are considered in each case. The judicious exploitation of redundancy, coding and randomization for performance enhancement, and cross-disciplinary approaches, have been recurring themes in much of his work.

Authorized licensed use limited to: Technion Israel Institute of Technology. Downloaded on January 20,2021 at 09:24:04 UTC from IEEE Xplore. Restrictions apply.