Minimal Maximum-Level Programming: Faster Memory Access via Multi-Level Cell Sharing

Amit Berman, Yitzhak Birk
Electrical Engineering Department, Technion – Israel Institute of Technology
{bermanam@tx, birk@ee}.technion.ac.il

***Abstract*-In multi-level-cell (MLC) 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 (*MMLP*), a scheme for expediting cell writing by sharing physical cells among multiple data pages and exploiting the fact that making moderate changes to a cell's level is faster than making large ones. Reading is also expedited by requiring fewer reference comparisons. In a four-level cell example, we achieve a 32% reduction in write/read latency relative to prior art with negligible area overhead.**

Keywords - Multi-Level Cell (MLC), Flash Memory, Phase-Change Memory, Read/Write Performance

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 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].

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 it and comparing the cell's threshold voltage to it. 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).

		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

Table 1. Latency of SLC and 4-level MLC in PCM and Flash memories [3,4,5,6].

The move to MLC, while beneficial in terms of storage capacity and cost per bit, comes at a performance penalty. Moreover, with an increase in capacity and a reduction in performance, the "normalized" performance drop is dramatic. There is therefore a true need for schemes that can somehow mitigate the performance drop, and this is our goal in this paper.

Another problem with MLC is endurance, namely the permissible number of erasure cycles that a cell may undergo before it degrades. Endurance can be 10x lower for 4-level cells than for 2-level cells. This paper does not address endurance, and the scheme proposed here does not affect it.

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 mitigation 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 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 physical page as SLC or MLC. Use in SLC mode increases endurance and accelerates access. In all modes, any given cell contains data of 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 physical page's capacity equals twice that of a logical page. The two logical 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.

C. *Our Contributions*

We propose and evaluate minimal maximum-level programming (*MMLP*), a scheme to accelerate MLC memory access. *MMLP* places and encodes the data such that in the k^{th} 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 pages use different numbers of physical cells, and a cell may hold a fraction of a bit of a given data page. Nevertheless, the exposed page size remains unchanged and data is encoded without redundancy, so no capacity is lost. For facility of exposition, the discussion will focus on Flash. *MMLP* may also be adaptable to other MLC memory technologies.

The remainder of the paper is organized as follows. Section 2 presents *MMLP*, and it is evaluated in Section 3. In Section 4 we discuss error handling, and Section 5 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*

D - logical or *data page* (or simply *page*) - a page of data as viewed by the host. Its size is fixed, typically 2-4kB.

physical page - a set of cells jointly storing one or more entire data pages and only those. Data of any given logical page may be spread across a subset of cells whose joint capacity exceeds the logical page size.

C - a set of cells jointly storing a given data page (and possibly additional pages or parts thereof).

P - the set of levels of the cells in *C*.

E - the data as encoded by *MMLP*.

MaxLevel(C) - the current highest level of any cell in *C*.

Address - By abuse of notation, this merely refers to the location of a logical page in its physical page, as if there is only one physical page. With MMLP, the (already existing) mapping tables would map a logical page number to a physical_page.

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 *C* to which that address is mapped, as well as *MaxLevel(C*).
- ***Encoder.*** Given *D*, *P* and an address (as defined), the encoder transforms the data such that writing the encoded data *E* into the target cells causes only minimal level changes in them. A page 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 page being written and the existing information in those cells, which is not lost!)
- ***Decoder.*** Given *E=P* (the levels of the cells containing the page 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.

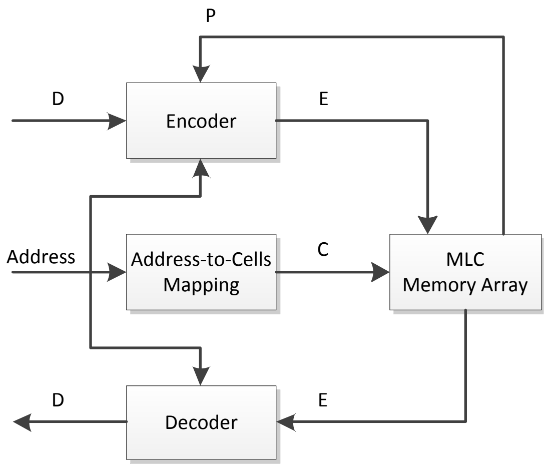


Fig. 1. Data flow among encoder, decoder, address-to-cells mapping and MLC memory array.

In each programming operation of a given physical page, we limit the maximum target cell level. Writing the 1st (logical) page may only use levels 0 and 1. Writing the 2nd page may only use up to level 2, etc. The encoder, decoder and address-to-cells mapping are determined by the physical and logical page sizes, and by the total number of levels. We next describe the *MMLP* flow.

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 page *C*) along with page data *D* (data that is to be written) and address are input to the

```
MMLP write flow (D, Address)
(1) C ← ATC(Address)
(2) P ← Current levels of memory cells C
(3) E ← Encoder(D, P, Address)
(4) Write E to memory cells C
(5) Update MaxLevel(C) // stored metadata
```

Fig. 2. Pseudo-code for MMLP write.

encoder. The encoder transforms the data such that moderate level changes would be made to 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, but the data already stored in them is not lost.

The pseudo-code of read flow is shown in Fig. 3.

```
MMLP read flow (Address, MaxLevel)
(1) C ← ATC(Address)
(2) MaxLevel ← ATC-MaxLevel(Address)
(3) E ← perform (MaxLevel-1) reference comparisons
        on cells C
(4) D ← Decoder(E, Address)
```

Fig. 3. Pseudo-code for MMLP read.

In step (1), the cells to be read C are determined based on the address. In step (2), the maximum level of the pages containing the desired page is read (metadata). In step (3), (*MaxLevel*-1) reference comparisons are used to determine the cells levels. (These are page-wide reference comparisons, so binary search is irrelevant.) Finally, in step (4), the decoder reconstructs original page D from E and address.

Memory erasure is not affected by *MMLP*.

We next provide the details of MMLP for 4-level cells and for 8-level cells with 2-bit pages. Larger pages are handled by commensurately increasing the physical page and simply handling two bits at a time in the encoder and decoder.

D. *MMLP for 4-Level Cells*

We use the following parameters:

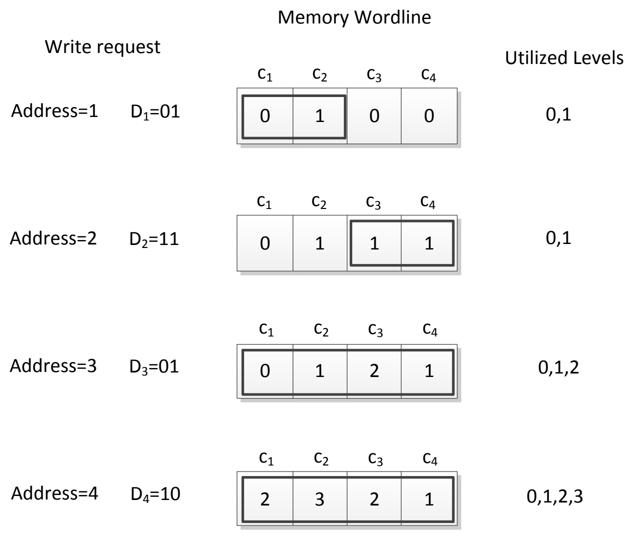
- D (logical page size): 2 bits
- Wordline (row) contains four cells $\{c_1,c_2,c_3,c_4\}$. There are four addresses for each wordline (physical page). Address-to-cells mapping: ATC(1)=$\{c_1,c_2\}$, ATC(2)=$\{c_3,c_4\}$, ATC(3)=$\{c_1,c_2,c_3,c_4\}$, ATC(4)=$\{c_1,c_2,c_3,c_4\}$.

Fig. 4 depicts a 4-level *MMLP* encoding/decoding table (a), as well as a specific example of writing four 2-bit pages of data into four 4-level (2-bit) cells (b).

The first two data pages, D_1 and D_2, are not encoded. They are stored in distinct cells ACT(1) and ACT(2) (see rectangular frames in Fig. 4a). Each bit of the 3rd page D_3 is mapped to a distinct pair of cells using levels {0,1,2}, and is thus spread across four cells. Finally, each bit of the 4th page D_4 is again mapped to a distinct pair of cells, this time using levels {0,1,2,3}. The encoding tables of pages 3 and 4 are given in Fig. 4(b). The mappings are all injective, and are thus reversible.

Any given cell is affected by three data bits, each from a different logical page (each cell contains a full bit of one page and half a bit of each of two additional pages).

Consider the specific data being stored (Fig. 4(a)). The 1st-page data is D_1=01, and is stored as is in the first two cells (one bit per cell). The 2nd-page data is D_2=11, and is stored as is in the next two cells. The third page data is 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, ACT(3) cells are read (P=0111) as their values affect the encoding of the new data. Using the page-3 encoding table (Fig. 4(b)), stored data 01 and input data 0 is encoded to 01. Similarly, writing data 1 over the cell pair 11 is



(a)

	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. 4. Example of MMLP. 4-level (2-bit) cells and D size of two bits. Each wordline has four cells, and can store four logical pages. (a) Mapping of addresses to cells (see frames), utilized cell levels, and cell levels following the writing of each page 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 other raised or remain unchanged.

encoded to 21, and the cells are programmed to 0121. Page 4 data is D_4=10, which encodes to 2321.

Consider reading of an address. The maximum possible cell level, MaxLevel(C), is known (stored metadata). For read of addresses 1 and 2, a single reference comparison suffices (the one between levels 0 and 1). Reading addresses 3 and 4 require two and three reference comparisons, respectively.

Data decoding is performed by using the encoding tables in reverse, starting from the last address. Reading of 2321 would decode to D_4=10 and address 3 levels 0121 (using page 4 encoding table in Fig. 4(b)). Decoding latency consists of combinatorial logic, and is negligible (nano-seconds) relative to reference comparison (tens of micro-seconds).

E. 8-Level Cell MMLP

Consider 8-level (3-bit) cells, also known as "TLC". Fig. 5 depicts the ATC, mapping 12 data pages into a single physical page, as well as the utilized levels after programming each page. The first four addresses (pages) are mapped to distinct cells, with one bit per cell, so no encoding is required. Addresses 5-8 are each mapped to twice as many cells as page size, so each cell stores an additional 0.5 bit of each page (a pair of cells stores an additional bit). Similarly, an address in the range 9-12 has 0.25 bit stored in each cell.

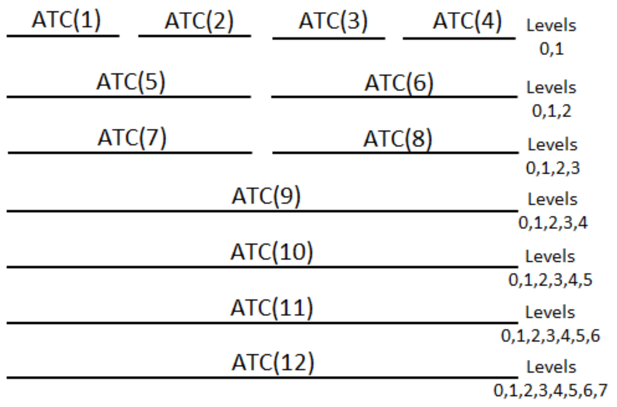


Fig. 5. Address-to-cells (ATC) mapping of data pages to a single wordline; 8-level cells.

Encoding tables for 8-level cells are omitted for brevity. Instead, we next show that such an encoding exists.

Proposition 1: An encoding per Fig. 5 exists; i.e., one requiring at most a single additional level per page write.

Proof: In the first four pages, each bit is programmed in a single cell using levels 0,1. When allowing only three levels 0,1,2, each cell can store $\text{Log}_2(3)$=1.585 bits. Therefore, each pair of cells can store 3 bits - the two bits of the previous page and one bit of page 5 or 6. Similarly, when allowing first n levels, each cell can store $\text{Log}_2(n)$ bits. We multiply the number of cells until data page can be added on top of previously programmed pages. This approach can be generalized to any number of levels per cell. □

In summary, *MMLP* has only a small fraction of data pages programmed to high levels, thereby accelerating the writing of most pages as well as their reading so long as subsequent pages have not been written. Yet, full storage capacity is utilized without redundancy. We next turn to quantify the benefits.

Remark. *MMLP* assumes that addresses are programmed in order. (This refers to the actual physical pages. It has no logical implications, given the use of mapping tables.)

III. Evaluation

This section quantifies the performance of *MMLP* for 4-level Flash cells: write latency, read latency and energy consumption. A VLSI design then serves for overhead estimation.

A. Parameter Values and Basic Expressions

Program and read time comprise 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}):

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

Table 2 provides numbers based on reported measurements [4]. Address multiplexing delay is some 1000x smaller than program and verify times, so it is omitted.

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 [μSeconds]
T_{vfy}	10 [μSeconds]

Table 2. 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.

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

$$T_{read} = Nr \cdot T_{vfy} \tag{2}$$

B. Write Latency

We derive this for *MMLP*, and compare it with prior art: Conventional and Multi-Page programming techniques.

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

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

In *Multipage Programming (MP)* [8], each cell stores one bit of each of two pages. The level-to-values mapping: 0↔11, 1↔10, 2↔00, 3↔01. The 1st page sets the least significant bit (levels 0 and 1). The 2nd page sets the most significant bit utilizing concurrent programming to levels 2,3 with two reference comparisons. Prior to 2nd 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 $T_{readmp}+\max\{Np_{0\to3}, Np_{1\to2}\}(T_{pulse}+2T_{vfy})$. The mean (over pages) write time is:

$$T_{MP}=\frac{1}{2}\Big(Np_{0\to1}\left(T_{pulse}+T_{vfy}\right)+T_{readmp}+ +\max\left\{Np_{0\to3},Np_{1\to2}\right\}\left(T_{pulse}+2T_{vfy}\right)\Big) \quad (4)$$

In *MMLP*, each cell is shared among four pages. It is the maximum possible parallelism with 4 levels. Writing each of the 1st two pages takes $Np_{0\to1}(T_{pulse}+T_{vfy})$. The 3rd page has level transitions 0→1, 0→2, and 1→2 (Fig. 4(b)), and cells are read T_{readp2} prior to program, taking $T_{readp2}+\max\{Np_{0\to1}, Np_{0\to2}, Np_{1\to2}\}(T_{pulse}+2T_{vfy})$. The 4th page has level transitions 0→2, 1→3 and 2→3 (Fig. 4(b)), and read T_{readp3} prior to program, taking $T_{readp3}+\max\{Np_{0\to2}, Np_{1\to3}, Np_{2\to3}\}(T_{pulse}+2T_{vfy})$. The read $T_{readp2}=1\cdot T_{vfy}$ prior to 3rd page program requires one comparison. The $T_{readp3}=2\cdot T_{vfy}$ prior to 4th page has two comparisons. The average (over pages) page writing time is:

$$T_{MMLP}=\frac{1}{4}\Big(2Np_{0\to1}\left(T_{pulse}+T_{vfy}\right)+T_{readp2}+T_{readp3} +\max\left\{Np_{0\to1},Np_{0\to2},Np_{1\to2}\right\}\left(T_{pulse}+2T_{vfy}\right) +\max\left\{Np_{0\to2},Np_{1\to3},Np_{2\to3}\right\}\left(T_{pulse}+2T_{vfy}\right)\Big) \quad (5)$$

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 3. Write Latency for MMLP and prior art (Conventional, Multipage); 4-level cells..

Results (Table 3). *MMLP* achieves 40% and 32% reduction in average program latency (speedup of x1.65 and x1.5) over Conventional and *MP*. For more levels, the gap increases; E.g., a 56.75% reduction relative to MP for 8-level (3-bit) cells,

C. *Read Latency*

Read latency depends on the number of utilized levels, so it varies with capacity utilization. When all levels are utilized, the read latency is similar for all methods: $T_{read}=3T_{vfy}$. When computing averages, we assume that the low levels of all physical pages are used before beginning to use higher levels.

In *Conventional* Reading, three reference comparisons are used without considering level utilization.

With *Multipage*, for utilization of up to 50%, $T_{read}=T_{vfy}$. Beyond that, any page uses all levels, so $T_{read}=3T_{vfy}$.

With *MMLP*, for utilization of up to 50%, $T_{read}=T_{vfy}$. Each page that is programmed between 50% to 75% capacity uses levels (0,1,2), and reading it requires 2 reference comparisons $T_{read}=2T_{vfy}$. Beyond that, any page uses all levels, so $T_{read}=3T_{vfy}$.

Once a “later” page (higher address in the same physical page) uses a high level, the larger number of comparison is also required when reading an “earlier” page.

MMLP’s relative reduction in the number of required reference comparisons grows as the number of levels increases. Fig. 6 depicts read latency vs. memory occupancy for 4- and 8-level *MMLP* and *Multipage* techniques. *MMLP* is as fast as Multipage in some occupancy ranges, and significantly faster in others. E.g., with 4-level cells, for 50%-75% occupancy, *MMLP* is 1.5x faster than Multipage (20µs vs. 30µs).

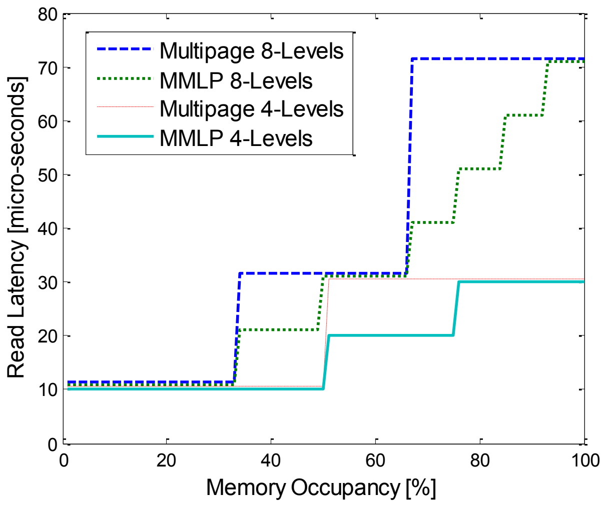


Fig. 6. Read Latency vs. Memory Occupancy of *MMLP* and *Multipage* with 4- and 8-level cells.

Finally, *MMLP* increases the number of cells accessed in parallel to the maximum possible. In some NAND Flash array designs, the wordline length has to be extended while commensurately shortening the bitline to keep a fixed number of cells. So doing further reduces read latency due to reduced bitline precharge and discharge duration. Program time is not affected by wordline extension, since program pulse duration is several magnitudes longer than wordline signal propagation (micro-seconds vs. nano-seconds). The above analysis did not incorporate this additional advantage, and is thus conservative.

D. *Trace-Based Evaluation*

We estimated the expected performance of MMLP relative to Conventional and Multipage for actual I/O traces, using storage traces from PC-MARK [12].

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

Consider 4-level-cell NAND Flash. With Conventional, write is 18x slower than read (900µs vs. 50 µs). Write energy consumption ranges between 10x and 630x of read [4]. In our comparison, we assume write energy to be 36x 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

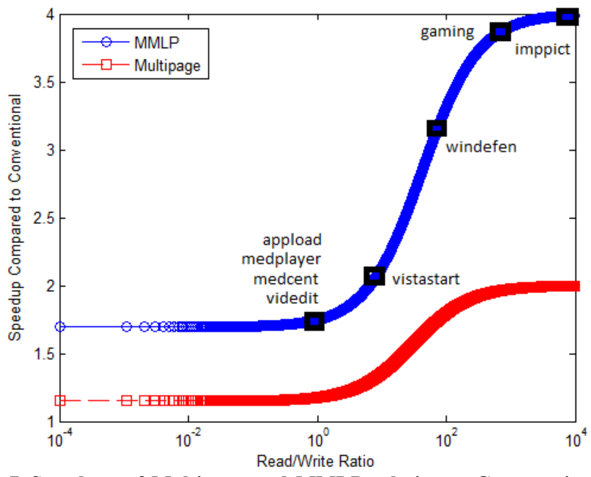


Fig. 7. Speedup of Multipage and MMLP relative to Conventnional vs. R/W ratio. Benchmarks are placed based upon their R/W ratio. (baseline write is 18x slower than read, and consumes 2x power than read).

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 a 1.5x-2x performance advantage of *MMLP* over Multipage (Fig. 7). A 2x - 2.7x reduction in energy consumption is also observed (figure omitted for lack of space).

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 pages. 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 625µm^2, 0.003% of a typical 144mm^2 die. Total power consumption, including leakage and dynamic power is 584µWatt, 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 savings relative to Multipage is 32%-50%.

IV. Error Handling and Endurance

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. However, any given cell is affected by at most one bit of any given page, so at most one error may result in each data page.
The data of any given page can be protected using conventional ECC. Also, assuming that no errors occurred until the final page was written to a given physical page, ECC that protects the cell states following the writing of the final page will guarantee correct decoding of all pages. Thus, all pages but the last can be programmed without ECC protection, and the last page is programmed with ECC, thereby reducing the required amount of ECC redundancy. Integrating ECC with last page programming is a subject for future research.

Endurance is the number of possible erasures, (not writes,) to a cell. In *MMLP*, encoding is done with no redundancy, so overall storage capacity remains unchanged. *MMLP* does write to more cells in parallel, but causes a smaller level increment in each cell, so the number of erasures does not change.

V. Conclusions

We proposed *MMLP* – minimal maximum level programming, a cell-sharing scheme that enhances write and read performance while saving energy in MLC memory. *MMLP* Minimizes the mean page-writing time, shortening it by at least 32% relative to prior art for 4-level cells. Whenever the memory is underutilized, Read is accelerated by reducing the amount of reference comparisons, based on a priori knowledge of the highest programmed level in a page.

MMLP results in variability of write/read time between pages. Exploiting this for performance optimization 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.

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

References

[1] J. Brewer, M. Gill, "Nonvolatile memory technologies with emphasis on flash", IEEE Press Series on Microelectronic Sys., 2008.

[2] B. Lee, E. Ipek, O.Mutlu, and D. Burger. "Architecting phase change memory as a scalable DRAM alternative". In ISCA-36, 2009.

[3] Laura M. Grupp et-al., "The Bleak Future of NAND Flash Memory", 10th USENIX conf. on file and storage technologies (FAST), 2012.

[4] Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, "Characterizing Flash Memory: Anomalies, Observations, and Applications", MICRO'09.

[5] Samsung Electronics, "K9NBG08U5M 4Gb*8 Bit NAND Flash Memory Data Sheet".

[6] Samsung Electronics, "K9GAG08U0M 2Gb*8 Bit NAND Flash Memory Data Sheet".

[7] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "FlexFS: A Flexible Flash File System for MLC NAND Flash Memory", USENIX Annual Technical Conference, 2009.

[8] K. Takeuchi, et-al. "A multipage cell architecture for high-speed programming multilevel NAND flash memories", Journal of Solid-State Circuits (JSSC), 1998.

[9] R.L. Rivest and A. Shamir, "How to reuse a write-once memory," Information and Control, vol. 55, nos. 1–3, pp. 1–19, 1982.

[10] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank Modulation for Flash Memories", IEEE Transactions on Information Theory, vol. 55, no. 6, pp. 2659-2673, June 2009.

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

[12] PCMARK-VANTAGE, White paper v1.0.