Write Amplification Reduction in NAND Flash through Multi-Write Coding

Ashish Jagmohan, Michele Franceschini, Luis Lastras
IBM T.J. Watson Research Center, {ashishja, franceschini, lastrasl}@us.ibm.com

***Abstract*—The block erase requirement in NAND Flash devices leads to the need for garbage collection. Garbage collection results in write amplification, that is, to an increase in the number of physical page programming operations. Write amplification adversely impacts the limited lifetime of a NAND Flash device, and can add significant system overhead unless a large spare factor is maintained. This paper proposes a NAND Flash system which uses multi-write coding to reduce write amplification. Multi-write coding allows a NAND Flash page to be written more than once without requiring an intervening block erase. We present a novel two-write coding technique based on enumerative coding, which achieves linear coding rates with low computational complexity. The proposed technique also seeks to minimize memory wear by reducing the number of programmed cells per page write. We describe a system which uses lossless data compression in conjunction with multi-write coding, and show through simulations that the proposed system has significantly reduced write amplification and memory wear.**

I. Introduction

NAND Flash is a solid-state memory technology, which is of increasing prominence in technology applications in both consumer and enterprise systems. Its main advantages include non-volatility, high density, reduced latency (compared to hard disks) and reduced power. Fundamental constraints that frequently arise in the use of NAND Flash memories include their limited endurance (5-10K program-erase cycles for the latest multi-level cell (MLC) memories), the large latency for programming, and the necessity of block erases.

The block erase requirement has led to the popular use of log-structured file systems (LFS) [1] for NAND Flash memory systems [2], [3]. In an LFS, a translation layer maintains a mapping between logical and physical block addresses, and other meta-data. Logical pages are written in a sequential log fashion to erased physical pages in the Flash memory. When the data on a logical page is updated, it is written to a new, erased physical page, the previous physical page corresponding to the logical page is marked invalid, and the translation layer logical to physical page mapping is updated.[1] In an example implementation [4] physical Flash blocks are partitioned into two pools, a pool of occupied blocks and a pool of free blocks (with one or more erased pages), both of which are maintained as queues. A logical page to be updated is written to the first erased page available in the free block at the head of the free block queue. When all pages in a free block have been programmed, the block is moved to the tail of the occupied queue.

[1]The Flash translation layer may maintain a mapping of logical to physical 'blocks' where a block is smaller than a page. For example, a block may be 512B while a page is 4KB. This 'block' is different from the physical Flash block which consists of multiple pages (e.g. 64 or 128 pages). For simplicity, and to avoid confusion, we will assume throughout that the smallest unit of data for which the translation layer maintains a mapping is a Flash page.

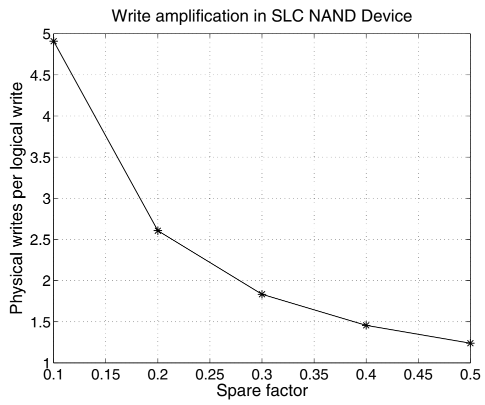
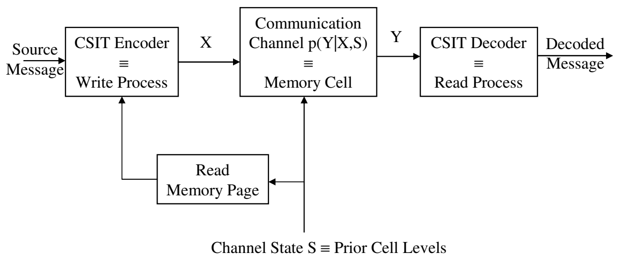


Fig. 1. Write amplification in an SLC NAND Flash device, with garbage collection implemented as in the base system described in Section IV.

The NAND Flash LFS structure requires the use of periodic garbage collection, to recover memory space by erasing previously programmed, invalid physical pages. Garbage collection may be invoked when the number of erased pages falls below a threshold, or when the system is not busy. In the example implementation [4], when the number of free blocks falls below a threshold, an occupied block is erased. Valid pages from the occupied block are written to erased pages from a free block, and the occupied block is erased and moved to the tail of the free queue. One block reclamation algorithm is the greedy algorithm wherein the block with the least valid pages is freed.

Garbage collection causes an increase in the number of physical page programming operations, over and above the page writes necessitated strictly by logical page updates. If W_p physical pages are required to be programmed for W_F logical page writes, *write amplification* A is defined as $A \triangleq W_P/W_F$. Figure 1 shows write amplification in an example system, described in greater detail in Section IV. As can be seen, when the spare factor (defined as $1 - \frac{\text{logical address space size}}{\text{physical address space size}}$) is 10%, there is a write amplification of more than five. This phenomenon further impacts the limited endurance of a NAND Flash device, and can add significant system overhead [5] unless a large spare factor is maintained.

The key idea underlying this paper is that write amplification can be significantly reduced by the use of a multi-



Source Message
CSIT Encoder ≡ Write Process
X
Communication Channel p(Y|X,S) ≡ Memory Cell
Y
CSIT Decoder ≡ Read Process
Decoded Message
Read Memory Page
Channel State S ≡ Prior Cell Levels

Fig. 2. Channel coding with side-information at the transmitter, and the eraseless reprogramming problem.

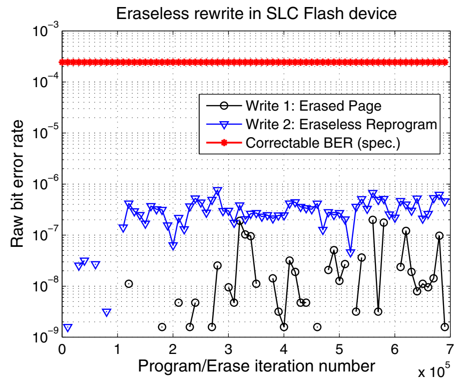


Fig. 3. Eraseless reprogramming of an SLC NAND FLash device. In the second write only erased cells are converted to programmed cells.

write coding technique which allows a NAND Flash page to be written more than once without requiring a block erase. Multi-write codes operate on the principle that it is possible to reprogram a Flash cell without an intervening erase, as long as the new program level requires the cell's floating-gate charge to be increased [6], [7]. Thus, a NAND Flash page can be 'eraselessly' reprogrammed as long as each cell on the Flash page only preserves or increases its floating-gate charge.[2] The problem of eraselessly reprogramming a NAND Flash page is closely related to the problem of *write-once* or *permanent* memories in information theory. Theoretical capacity results and practical code constructions for these and related memories have been previously presented [6], [7], [8], [9], [10], [11].

In this paper we propose a NAND Flash system which uses multi-write coding to reduce write amplification and memory wear. We present a novel two-write coding technique which achieves linear coding rates with low coding complexity, and minimizes the number of programmed cells. We then describe a system which uses lossless data compression in conjunction with multi-write coding. We show through simulations that the proposed multi-write system has significantly reduced write amplification, even with the availability of just one additional eraseless reprogram. There is also a significant reduction in the number of programmed cells (and hence memory wear), thereby improving system lifetime. For simplicity we will mostly focus on single-level cell (SLC) devices, but the principles described are readily extensible to MLC devices.

II. Multi-write Flash Coding

Consider a Flash page wherein each cell supports data levels from a discretized set of levels $\mathcal{L}$. Define a relation $\prec$ such that for levels $l_i, l_j \in \mathcal{L}$, $l_i \prec l_j$ denotes that a cell programmed with level l_i can be eraselessly reprogrammed to level l_j. The problem of eraselessly reprogramming a currently programmed page of this type, is an instance of the information-theoretic channel coding with transmitter side-information (CSIT) paradigm [12], shown in Figure 2.

In the CSIT paradigm, the encoder wishes to communicate data over a communication channel with a state-dependent transition probability, wherein the state is known to the encoder but not the decoder. In the problem at hand, the encoder is the memory reprogramming process, the communication channel is the memory storage channel, the state of the memory is the sequence of previously programmed cell levels, and the decoder is the memory read process. The memory reprogramming process reads the memory prior to reprogramming and is, hence, aware of the prior cell levels. Since the act of reprogramming changes the cell levels, the memory read process is agnostic of the prior cell levels. For a noiseless n-cell page with contents $l_0, l_1, \ldots, l_{n-1}$, it can be shown [12] that the maximum per-cell number of bits which can be stored with a single eraseless reprogramming operation is (asymptotically) $\frac{1}{n}\sum_{i=0}^{n-1}\log|\mathcal{L}_i|$ bits, where $\mathcal{L}_i \triangleq \{l \in \mathcal{L} : l_i \prec l\}$, and the logarithm is base-2. Interestingly, there is no asymptotic capacity loss compared to the case where the read process also knows the prior cell levels.

Such memories have been studied in the information theory literature as write-once memories or permanent memories [8], [9]. Several code constructions for these have been previously proposed, for example in [6], [7], [8], [9], [10], [11]. These are either restricted to small block lengths or have sub-linear rates.

TABLE I
Eraseless reprogramming for a 4-level MLC device.

Start Level	0	1	2	3
Reprogrammable Levels	0,1	2,3	2,3	2,3

A. *Eraseless Reprogramming in NAND Flash*

SLC Flash devices have two levels, an erased level (conventionally, logic level '1') and a programmed level (logic level '0'). In accordance with the above discussion it may be expected that an SLC Flash page is eraselessly reprogrammable as long as this operation does not require any programmed cell to be erased. In terms of the defined notation an SLC device has $\mathcal{L} = \{0, 1\}$ with $1 \prec 0, 1$ and $0 \prec 0$.

Figure 3 shows experimental data for an SLC NAND Flash device (rated endurance 100K cycles), showing the raw program bit-error rate (BER) for one such eraseless reprogram ('Write 2') in each program/erase iteration. The reprogram causes a higher BER, but the BER is well below the maximum correctable rate specified in the device's ECC recommendations (1 bit error per 512 bytes). Table 1 shows

[2]Some devices may have characteristics that additionally constrain reprogrammability.

experimentally obtained allowable multi-write transitions for a 4-level MLC device. For this device, $\mathcal{L} = \{0, 1, 2, 3\}$ with $0 \prec 0, 1$ and $1, 2, 3 \prec 2, 3$. Figure 3 and Table 1 demonstrate the possibility of performing eraseless reprogramming on NAND Flash devices. We note that more detailed device characterization, including retention testing, will be crucial in practice. Further, a higher BER for eraseless rewrites may require a stronger error-control code (ECC).

We now present a novel two-write coding technique which allows information rates which are linear in the block length to be encoded. The coding technique assumes SLC Flash but can be extended to MLC Flash memories.

B. Coding Technique

The theme underlying CSIT code constructions is the partition of the space of encoded sequences into mutually exclusive sets, and a mapping of data messages to the indices of these sets. Thus, a particular data message corresponds to one of the partition sets, and is communicated by selecting a sequence from that set based on the state of the channel. Specifically, for eraseless reprogramming of an n-cell programmed SLC NAND Flash page, the rewrite code consists of a partition of the sequence space $\{0, 1\}^n$ into mutually exclusive sets, and a mapping of possible data messages onto the partition set indices. Reprogramming is done by first determining the set index which corresponds to the data to be stored, and then selecting a sequence from the set which can be eraselessly reprogrammed given the current contents of the Flash page. The stored data can be recovered by reading this sequence (assuming the absence of noise) and determining the partition set to which the sequence belongs. We now present a computationally efficient two-write technique which uses short block codes in conjunction with enumerative coding [13]. The technique allows a programmed Flash page to be eraselessly reprogrammed once and additionally seeks to minimize the number of programmed cells in order to maximize endurance.

Consider an SLC Flash device with page size n bits. We assume, for now, that the device is noiseless. We consider the two-write case, wherein the first write is done on an erased page, and the second, subsequent write is done by eraselessly reprogramming the page. Assume that R_1 bits of data are to be stored in the page in the first write, and R_2 bits are to be stored in the second write (R_1, R_2 ¡ n). We think of the page as consisting of b-bit sized sub-pages, where n is a multiple of b, with page contents represented by the sub-page symbol sequence $s_0 s_1 \ldots s_{\frac{n}{b}-1}$ where $s_i \in \overline{\mathcal{L}} \triangleq \{0, \ldots, 2^b - 1\}$, $i \in \{0, \ldots, \frac{n}{b} - 1\}$. The physical contents of the sub-page cells are given by the binary expansion of the sub-page symbol.

For the first write the sub-page symbols are constrained to belong to a subset $\mathcal{L}_1 \subset \overline{\mathcal{L}}$. Each sub-page symbol $s \in \mathcal{L}_1$ has an associated programming cost c_s equal to the average number of programmed cells required by the symbol. A frequency distribution $\{p_j : j \in \{0, \ldots, |\mathcal{L}_1| - 1\}$ ($\sum_j p_j = 1$) on the sub-page symbols is found, such that the number of $\frac{n}{b}$-symbol sequences satisfying this symbol frequency distribution is larger than 2^{R_1}. Enumerative source coding [13] is then used to encode to map the data to be stored onto a unique $\frac{n}{b}$-symbol sequence $s_0^{(1)} s_1^{(1)} \ldots s_{\frac{n}{b}-1}^{(1)}$ with symbol frequency distribution $\{p_j\}$, which is programmed to the page. Conceptually, this consists of a lexicographic enumeration of all sequences with symbol frequency distribution $\{p_j\}$; during encoding, the data to be stored is used to index this enumeration, and the indexed sequence is the codeword representing the data. Decoding consists of computing the index from the codeword. Such coding can be efficiently implemented using known techniques [13], [14].

For the second write, the set $\overline{\mathcal{L}}$ is partitioned into mutually exclusive subsets $\{\mathcal{P}_i\}_{i=0}^{m-1}$. A frequency distribution $\{q_k\}_{k=0}^{m-1}$ on the index set $\{0, \ldots, m - 1\}$ is found, such that the number of $\frac{n}{b}$-symbol sequences satisfying this distribution is larger than 2^{R_2}. As in the first write, enumerative coding is used to map the data to be stored onto a unique $\frac{n}{b}$-element sequence $s_0^{(2)} s_1^{(2)} \ldots s_{\frac{n}{b}-1}^{(2)}$, where $s_i^{(2)} \in \{0, \ldots, m - 1\}$, with symbol frequency distribution $\{q_k\}$. The sequence to be reprogrammed onto the page is found as follows. To obtain the symbol for sub-page k, a symbol is selected from the partition set $\mathcal{P}_{s_k^{(2)}}$ which can be eraselessly reprogrammed from the prior symbol $s_k^{(1)}$. If no such symbol can be found an encoding error is declared. In practice, the set $\mathcal{L}_1$ and the partitions $\{\mathcal{P}_i\}$ are selected such that an error never occurs. The sequence of selected symbols is then programmed onto the Flash page. An example two-write code uses the following parameters (similar to a length-3 code in [9]): $b = 3$, $\mathcal{L}_1 = \{111, 110, 101, 011\}$, $\mathcal{P}_0 = \{000\}$, $\mathcal{P}_1 = \{101, 010\}$, $\mathcal{P}_2 = \{011, 100\}$ and $\mathcal{P}_3 = \{001, 110, 000\}$.

Next, we return to the selection of $\{p_j\}$ and $\{q_k\}$. We would like to minimize the average number of programmed cells, in addition to satisfying the rate constraints. Define the cost c_{jk} as the average number of additional cells programmed when a symbol from partition set $\mathcal{P}_k$ is eraselessly reprogrammed onto a sub-page containing prior symbol $j \in \mathcal{L}_1$. The selection

$$p_j = e^{-\lambda(c_j + \sum_k q_k c_{jk})} / Z_1, \; q_k = e^{-\mu(\sum_j p_j c_{jk})} / Z_2 \qquad (1)$$

can be shown to be asymptotically optimal when R_1 and R_2 are both known prior to the first write. Here λ and μ are selected so as to satisfy $\sum p_j \log p_j \geq bR_1/n$ and $\sum q_k \log q_k \geq bR_2/n$, and Z_1, Z_2 are normalization constants. If only R_1 is known at the time of the first write, $\{p_j\}$ can be greedily optimized as $p_j = e^{-\delta c_j}/Z$ with δ selected to meet the rate constraint. In practice, the optimization is complicated by finite codeword length, and distributions corresponding to appropriately discretized programming rates are precomputed.

Finally, we briefly discuss the issue of providing robustness against noise. Error-control coding can be performed in two ways. First, the spare area can be partitioned, with a separate partition used to store parity bits for each eraseless rewrite. An alternative is to use a Reed-Solomon [15] code with alphabet size tied to the sub-page size. An appropriately constructed code can ensure that an error only affects one symbol in the multi-write code, while also ensuring that the parity symbols can be eraselessly reprogrammed. We omit details due to space constraints, but note that both alternatives lead to an increase in the spare area required to store parity bits.

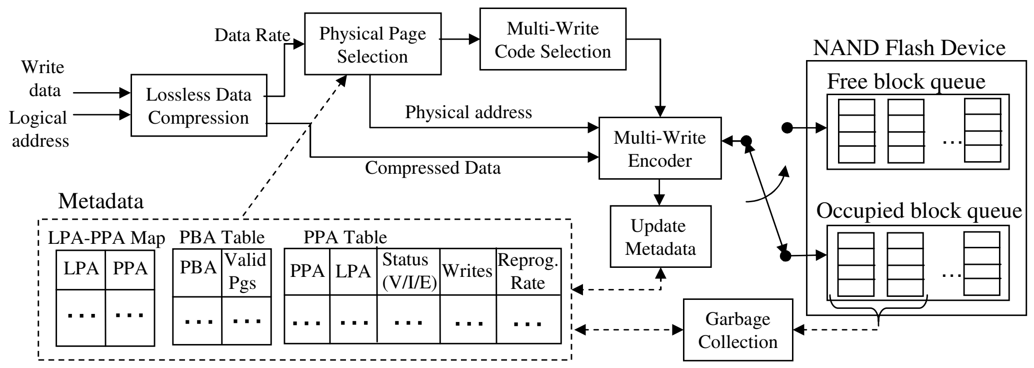


Fig. 4. Proposed Nand Flash system which uses multi-write codes during programming.

III. System Description

Figure 4 shows an overview of the proposed NAND Flash system which uses the multi-write coding technique presented in Section II to reduce write amplification. The conventional single-write LFS described in [4] is used as a base. Key additional components include lossless compression, write page selection based on compression rate and on meta-data information describing invalid pages, multi-write coding, and modified garbage collection. We omit wear-levelling considerations since our focus is on write amplification; previous work on the combined problem of wear-levelling and write amplification can be found in [16], [17].

Multi-write codes generally result in an expansion of data. If the size of the coded data is not a multiple of the physical page size, the result is a loss of page alignment which, in turn, may lead to increased meta-data and programming latency overhead. This issue can be resolved in multiple ways. One way is to fix logical and physical write unit sizes such that page alignment is maintained (i.e. the physical write size is always a multiple of the device page size) and to design appropriately parametrized multi-write codes. An alternative approach, for compressible data, is to apply lossless compression and to subsequently *adapt* the multi-write code rate to the compressed data rate. This method ensures page alignment, for example by fixing the physical write unit size to the device page size, and does so in a manner which can be completely transparent to system layers above the memory controller. The decision of which of the above methods to use will depend strongly on the precise system application; for the remainder of the paper, we will assume that the write data is compressible, and the second method is used.

The system maintains two pools of blocks organized into queues, namely a free block queue and an occupied block queue. The free block queue consists of blocks with least one erased page, and the occupied block queue consists of all other blocks. The controller maintains a logical page address (LPA) to physical page address (PPA) mapping. The controller also maintains other meta-data information including physical page status information, the number of times a non-erased page has been eraselessly reprogrammed, and the available reprogramming rate, i.e. the amount of data that can be reprogrammed on a non-erased page (this is dependent on the data rates programmed during previous page writes). Each page can be in one of three states: *erased*, *valid* (i.e. the page contains valid programmed data), and *invalid* (i.e. the page is non-erased, but contains invalid data). Erased pages can be programmed with information up to the page size, while invalid pages can be reprogrammed with information up to the available reprogramming rate as long as the number of reprograms is less than the maximum allowed. Compared to a single-write system such as [4], the number of page reprograms and available reprogramming rate constitute additional meta-data which must be stored. For a two-write system, the former requires one bit per page while the latter requires a few bits per page, for a system with a small number of allowable programming rates. Further optimizations can be performed to reduce this meta-data storage requirement; for example, since this meta-data is primarily used in write page selection, it does not need to be stored for all physical pages. Instead, a cache-like structure can be used to store such information for a small number of invalidated pages, and the write selection process only considers these pages as candidates for reprogramming. The cache can be periodically updated during downtime, or after programming operations, with replacement on the basis of criteria such as available reprogramming rate and number of reprograms. These and other meta-data optimizations are deferred to future work. Finally, as in [4], the controller also tracks the number of valid pages at each physical block address (PBA), for use in garbage collection.

Compressed data is programmed as follows. Given the compression rate, a search is done for an invalid reprogrammable page with available reprogramming rate greater than compression rate. The search is done over a small subset of previously programmed pages, for example, the pages in a small number of blocks at the head of the occupied block queue (or from a cache as described above). If only one such page is found, an appropriate multi-write code is used to encode the data and eraselessly reprogram the page. If no page is found the data is written to the next available erased page in the block at the head of the free block queue. If there is more than one candidate invalid page, the page with the maximum available reprogramming rate is selected. As can be seen from (1), such a selection greedily minimizes the endurance cost of reprogramming. Note that the endurance cost could be minimized by always choosing an erased page to program, but that would eliminate the write amplification gain. Optimizing this trade-off is an interesting open theoretical and practical problem.

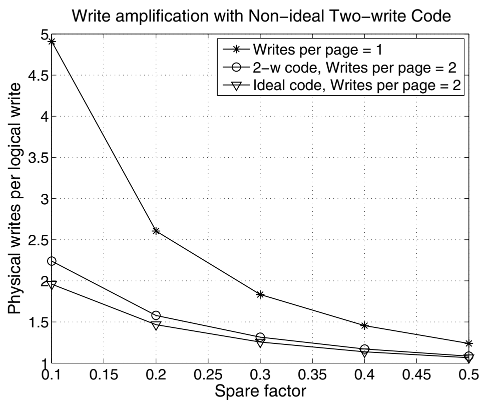
When the number of free blocks falls below a threshold, garbage collection is done. We use the same greedy strategy as in [4], where the block with the minimum number of valid pages in a window positioned at the head of the garbage queue is selected for erasure. Further optimizations can be done on the basis of the total number of available page reprograms in the block, or the total endurance cost of erasing each block. The valid pages in the selected block are written to other programmable pages, and the block is erased and placed at the tail of the free queue.

IV. Results

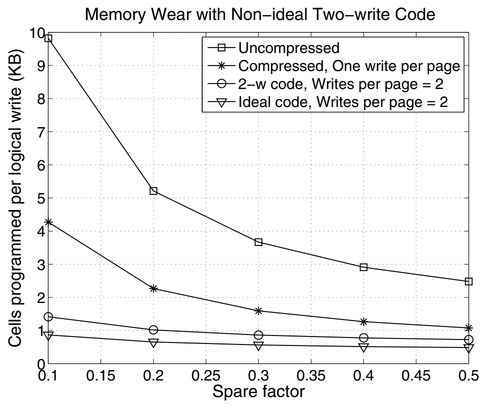
We present simulation results showing the effect of multi-write coding on write amplification and memory wear in the NAND Flash system shown in Figure 4. Memory wear is quantified by the number of cells programmed per logical page write. The simulated device is an SLC device with 4000 blocks, each consisting of 64 pages of size 4K bytes. Reprogramming page search, and block reclamation for garbage collection are both done using windowed searches at the occupied block queue head; the window consists of 25 blocks for the reprogramming search, and 500 blocks for block reclamation. The number of reserve free blocks is set to 10 as in [4]; once the free block queue has less than these many blocks remaining, garbage collection is initiated. The spare factor ($= 1 - \frac{\text{logical address space size}}{\text{physical address space size}}$) is varied from 0.1 to 0.5. In general, write amplification and memory wear decrease as the spare factor is increased. The input to the system is assumed to be compressible data, with logical addresses selected uniformly at random from the logical address space. Write amplification and memory wear are measured over a trace of length 100 times the physical address space size. We ignore the effects of error-control coding for these simulations.

Figure 5(a) shows the cumulative distribution of the compressibility of the input data; this is taken from a sample archive (*mozilla*) from the standard Silesia corpus [18], and is obtained by losslessly compressing 4KB segments of the archive. The data compressibility varies over a range, with significant frequency mass at compression rates roughly 0.25, 0.5 and 1 (i.e. 4:1 compressibility, 2:1 compressibility, and incompressible data), with mean compression rate around 0.5. In our simulation, the compressibility of each page is randomly selected according to the distribution in the figure. Figures 5(b) and 5(c) show the effect on write amplification and memory wear of using ideal capacity-achieving multi-write codes. Figure 5(b) shows that write amplification can be substantially reduced with just 2 writes per page. The write amplification factor [4] (equal to the excess write amplification over one) undergoes a four-fold reduction at 0.1 spare factor. Alternatively, two writes allow the same amplification with 0.1 spare factor, as achieved with roughly 0.3 spare factor using conventional programming. The figure also shows the decreasing returns with larger numbers of multi-writes. In particular, there is almost no performance difference between 4 and 8 multi-writes, due to the decreasing reprogram rate availability with each additional multi-write. Figure 5(c) shows the effect of multi-write coding on memory wear, quantified as the number of cells programmed per page write. It is assumed that uncoded data consists of roughly an equal number of 0 and 1 bits, on average. As expected, compression alone gives a reduction by a factor of about 2, corresponding to the mean compressibility of the data. Multi-write coding with 2 writes yields an additional four-fold reduction in the number of cells programmed per page write, thereby yielding almost an order of magnitude reduction over uncompressed, uncoded data.

Finally, Figure 6 shows corresponding write amplification and memory wear improvements for the two-write coding technique presented in Section II. The code is based on the 3 bit sub-page code described in Section II-B and allows one eraseless rewrite (in addition to the initial write). Fixed sets of eight discretized rates are used for each write. As expected, there is some performance loss compared to ideal coding, but the improvement over single-write coding is still significant. At 0.1 spare factor, for example, the write amplification factor and memory wear improve over the compressed single-write case by a factor of about three. Alternatively, the system with two-write coding achieves the same write amplification and memory wear at 0.1 spare factor as the single-write system does at roughly 0.25 and 0.35 spare factors respectively.

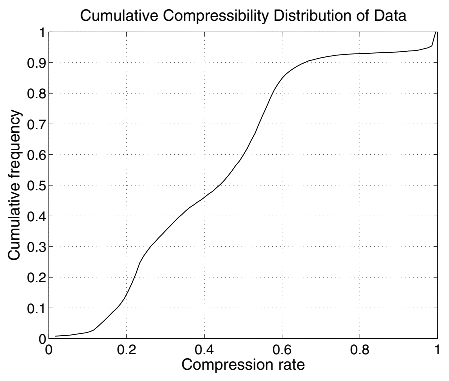


(a)

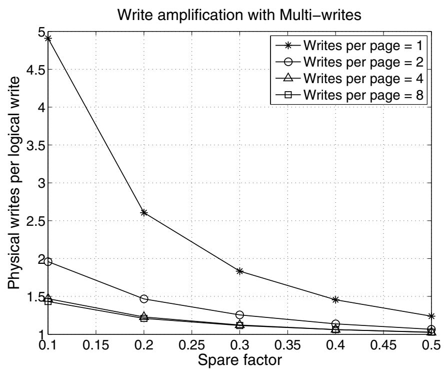


(b)

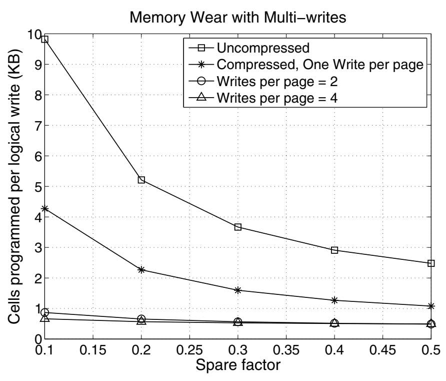
Fig. 6. Simulation results for two-write code (section II-B) and comparison to single-write and ideal codes. (a) Write amplification. (b) Memory wear.



(a)



(b)



(c)

Fig. 5. Simulation results for ideal multi-write codes. (a) Cumulative data compressibility distribution. (b) Write amplification. (c) Memory wear, measured as the number of cells programmed per logical page write.

V. Remarks

We have presented a multi-write coding system for NAND Flash devices which can substantially reduce write amplification and memory wear. Several issues merit further investigation. A key issue is Flash device characterization to evaluate the physical effect of multi-write coding. Newer technology Flash devices display a number of complicated coupling and disturb noise effects. It will be crucial to evaluate the effect of multi-writes on bit error rate, and on retention performance. A second issue is improving multi-write code performance, and designing efficient encoder and decoder implementations. A third issue is that of system optimization. Opportunities include algorithm design for reprogramming page selection and for garbage collection, and optimization of meta-data structures and algorithms.

References

[1] M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” *ACM TOCS*, vol. 10, pp. 1–15, 1991.

[2] D. Woodhouse, “Jffs: the journaling flash file system,” in *Proc. Ottawa Linux Symp.*, 2001.

[3] R. Konishi et al., “The linux implementation of a log-structured file system,” *ACM SIGOPS Oper. Sys. Rev.*, vol. 40, pp. 102–107, 2006.

[4] X. Hu et al., “Write amplification analysis in flash-based solid state drives,” in *SYSTOR*, 2009.

[5] T. Kgil et al., “Improving NAND flash based disk caches,” in *Proc. IEEE ISCA*, 2008, pp. 327–338.

[6] A. Jiang et al., “Floating codes for joint information storage in write asymmetric memories,” in *Proc. IEEE ISIT*, 2007, pp. 1166–1170.

[7] H. Mahdavifar et al., “A nearly optimal construction of flash codes,” in *Proc. IEEE ISIT*, 2009.

[8] C. Heegard, “On the capacity of permanent memory,” *IEEE Trans. Info. Th.*, vol. 31, no. 1, pp. 34–42, 1985.

[9] R. L. Rivest and A. Shamir, “How to reuse a ẅrite - once m̈emory,” in *Proc. ACM STOC*, 1982, pp. 105–113.

[10] A. Jiang, “On the generalization of error-correcting wom codes,” in *Proc. IEEE ISIT*, 2007, pp. 1391–1395.

[11] E. Yaakobi et al., “Error correction coding for flash memories,” Flash Memory Summit, 2009, www.flashmemorysummit.com.

[12] G. Keshet et al., “Channel coding in the presence of side information,” *Found. Trends Comm. Info. Th.*, vol. 4, no. 6, pp. 445–586, 2007.

[13] T. Cover, “Enumerative source encoding,” *IEEE Trans. Info. Th.*, vol. 19, no. 1, pp. 73–77, 1990.

[14] T.V. Ramabadran, “A coding scheme for m-out-of-n codes,” *IEEE Trans. Comm.*, vol. 38, no. 8, pp. 1156–1163, 1990.

[15] S.Lin and D.J.Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, NJ., 2004.

[16] N. Agrawal et al., “Design tradeoffs for ssd performance,” in *USENIX ATC*, 2008, pp. 57–70.

[17] L. Chang et al., “Real-time garbage collection for flash-memory storage systems . . . ,” *ACM TECS*, vol. 3, no. 4, pp. 837–863, 2004.

[18] S. Deorowicz, “Silesia corpus,” Silesian University of Technology, Poland, 2003. http:// data-compression.info/Corpora/SilesiaCorpus.