Retired-Page Utilization in Write-Once Memory – a Coding Perspective

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

Abstract—**In write-once memory (e.g., Flash), a cell's level can only be raised, and erasure is only in bulk. The total number of erasures (endurance) is limited, and drops sharply with technology shrinkage and with cell-capacity increase. The normalized write capacity (ratio of total amount of data that can be written to storage capacity) drops similarly. Various coding schemes enable overwrites at the expense of storage capacity. With all of them, whenever desired data cannot be written to its current page, the page is "retired" for subsequent erasure.**

Interestingly, however, a retired page can still be used for writing other data, and it has been proposed to try and use retired pages. Simulation results are promising. In this paper, after briefly presenting Retired Page Utilization, we cast it as a cross-page coding technique. We employ Markovian analysis to derive the expected number of random-data writes with RPU, and show how the required state space can sometimes be substantially reduced.

I. Introduction

A. *Background and the challenge*

NAND Flash is presently the most prominent non volatile solid-state memory technology, and the use of solid-state drives is expanding.

In Flash, the charge level of a cell can only be raised. As long as one makes changes to the data of a given page such that the level of any cell either remains unchanged or is raised, in-place updates are possible. Once the desired data cannot be written to its page, the page is ***retired*** for subsequent erasure and reuse, and the data is written elsewhere. Erasure can only be carried out in bulk (multi-page blocks). Moreover, the number of erasure cycles (*endurance)* is limited**.**

The Solid-State Drive (SSD) cost challenge is addressed mostly by aggressive device scaling and by multi-level cell (MLC) architectures [1, 2]. Unfortunately, these reduce the endurance 10-100 fold, thereby limiting product lifetime [3, 4].

Endurance itself is a purely physical property, as are the number of cells and the number of levels per cell. However, the truly important measures from a system perspective are storage capacity and the normalized write capacity.

Definition 1: visible (raw) storage capacity is the amount of information that may (can) be stored in a device at any given time. (The raw capacity of device with N L-level cells is $N\log_2 L$. With coding, the visible storage capacity can be much lower than this raw capacity.)

Definition 2: normalized write capacity is the total amount of information that can be written to a storage device during its useful lifetime, divided by the product of the device's raw storage capacity and its endurance. The challenge is to improve the trade-off between device cost and its performance.

B. *Coding for Write-Once Memory*

The realization of the "pain" in having to nearly always erase a page after each write to it has triggered extensive research into techniques that can mitigate this situation. Most notably, techniques that combine some intra-page over provisioning of storage space (redundancy) with one-to-many data mappings: the same information can be represented in more than one way, thereby offering a choice among several writing options in the hope that at least one is permissible in the current states of the target page's cells.

Write-once memory (WOM) codes, first suggested by Rivest and Shamir [5], paved the way for exploring efficient and high-rate WOM codes. Fig. 1 depicts an example of writing two bits twice into three two-level cells (SLC): the encoding of the 1st write is dictated by the corresponding column, whereas the 2nd one may use either encoding. One can easily verify that following any 1st-write data, any 2-bit data can be written without a need to change a 1 to a 0. Normalized guaranteed write capacity is increased from 1 to 4/3.

Data	1st Write	2nd Write (if data changes)
00	000	111
01	001	110
10	010	101
11	100	011

Fig. 1. Write-Once Memory (WOM) coding for writing two bits twice in three SLC cells [5]. Normalized write capacity =4/3.

Various constructions of WOM codes have been proposed, along with encoding and decoding algorithms. Rivest and Shamir proposed tabular and linear WOM codes [5]. Cohen et-al. described a coset-coding technique [6]. Jiang et-al. suggested generalizations for a multi-level memory cell [7]. Yaakobi et-al. proposed high-rate WOM codes [8]. These proposed WOM constructions focus on maximizing the code rate under the assumption that a codeword is exponentially larger than data size. It is shown [5] that in order to enable two guaranteed writes of a data block, at least 50% additional storage capacity is required.

With Floating WOM codes [9], the setting of only one bit is permissible in each write. Since data is likely to change in more than one bit, WOM codes are more suitable for Flash storage. Other codes include rank modulation codes, which encode data as a permutation of the relative values [10].

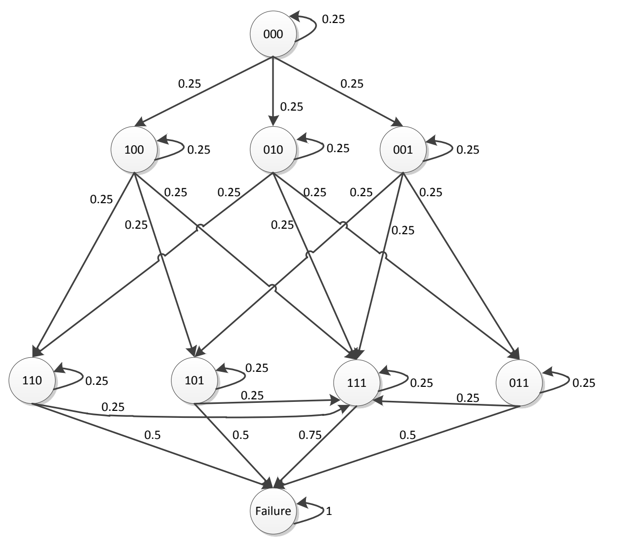


Fig. 2. Markov chain of Rivest and Shamir wom of 2 guranteed writes of two bits using three binary cells for random uniformly distributed input data.

Practical WOM codes are moreover often combined with error-correction mechanisms in Flash memory controllers [11].

Current WOM codes focus on improving the guaranteed (worst-case) number of writes in the same set of cells. In [12], it was argued that the mean (over data) number of writes is more important: given the large number of erasure cycles and the remapping of pages, the mean write capacity is achieved with very high probability. It was also shown there that writing random data using a WOM code can be represented by a Markov chain, and the CDF of the number of writes between erasures was derived for certain codes. Fig. 2 depicts the Markov chain for the 2-3 code [5]. The mean number of successful writes is 3.29 (vs. 2 guaranteed writes).

Write capacity (absolute, not normalized) can also be increased by "external" over-provisioning [13], namely additional pages that are not declared to the user. Data pages are written to blank physical pages, with the mapping chosen so as to roughly equate the number of writes to all physical pages (wear leveling). In fact, "cold" data can be moved from little-used physical pages to "aging" ones.

File system approaches utilize the flexibility to use MLC as SLC. FlexFS [14] dynamically allocates SLC and MLC memory regions according to application needs, possibly requiring data reorganization and large over-provisioning.

Common to all the above schemes is that overwrite attempts, if any, are made only to a page's current location. Upon failure, the physical page is retired for subsequent erasure and is not used until erased, and the logical page is remapped to a blank physical page. WOM coding and over-provisioning serve to improve the normalized write capacity at the cost of storage capacity [5, 8, 13, 14].

In [12], it was observed that a page was retired because certain data couldn't be written in it, but other data may be writable in it. Fig. 3 depicts an example of four 4-bit pages, two of them blank and two retired (not blank). Clearly, 0111 can be written into page 3, whose content is 0110, obviating the need to contaminate a blank page. This has led to *Retired-Page Utilization* (RPU), a family of schemes that complement the aforementioned techniques, all of which operate at the intra-page level. RPU has been shown by simulation to improve write capacity by anywhere from a few percents to several fold, depending on the WOM code being used, on the RPU policy, and on parameter values. Studies of RPU policies are ongoing.

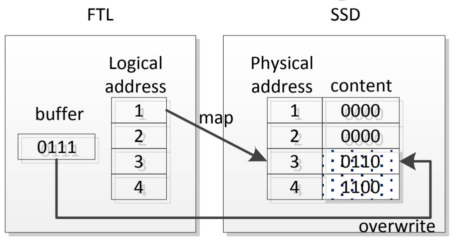


Fig. 3. Example of retired page utilization (overwrite) opportunities. (Retired pages are marked with dots).

The remainder of this paper is organized as follows. In Section II presents RPU's salient components and features. Section III presents the main contribution of this paper; we cast RPU as a cross-page coding technique, extend the Markovian analysis of [12] to RPU, and employ a two-step approach for the probabilistic analysis of the number of possible random-data writes, thereby reducing the state space. The approach is demonstrated on a simple example, but is adaptable to more complex situations. Section IV offers concluding remarks.

II. Retired-Page Utilization (RPU)

RPU entails the use of extra pages (beyond the declared storage capacity), as well as ones that are unused when the SSD is not full to capacity with data. These initially-erased pages form the *Retired Page Pool* (RPP). Whenever a page is retired, it joins the RPP. The data that couldn't be written into this page is written into an RPP page, preferably a non-erased one, and that page is withdrawn from the pool. RPP size is thus unaltered as long as the amount of stored data is unchanged.

There are always many extra pages in an SSD (to permit wear leveling, block erasure, etc.), so the RPP comes for free. Dynamic page mapping is also used anyhow for these and other reasons. Incorporation of RPU thus entails mostly a policy change and small additional metadata.

Whenever there are multiple RPP pages to which a given data page can be written, a page selection policy is used. Considerations may include the ease of finding a suitable page, minimization of "page consumption" (e.g., total number of level changes), the block to which a page belongs, etc.

RPU (probabilistically) increases the number of writes to a physical page between erasures, thereby increasing the write capacity for any given physical endurance. Since the RPP comes for free, the normalized write capacity also increases.

Obviously, if the number of retired pages considered for writing a given data page is kept moderate, the impact of RPU is significant if and only if the probability of any given retired page being writable is sufficiently high. This depends on the coding scheme used in the first place, comprising WOM coding and ECC, both of which permit some flexibility in data representation at the cost of reduced storage capacity. RPU complements intra-page WOM coding and ECC.

III. Markovian Analysis of RPU

As depicted in Fig. 4, RPU employs any desirable (intra-page) WOM encoder. The RPU policy decides where to write a data page, and the page mapping table is updated accordingly. Reading is unchanged.

We now consider a specific, simple RPU policy and uniform i.i.d. data, dubbed “random”. For simplicity, we consider a single data page and m physical pages. The data page is (over)written in place until it reaches a *near-failure* (NF) state, namely one in which there is at least one data value that cannot be written successfully. Then, writing continues similarly in a blank page. Once all pages are in NF states, they are considered for writing data in a round-robin fashion, starting with a with the data page’s current address and moving to the next page upon failure. The scheme reaches Failure state when the desired data cannot be written anywhere.

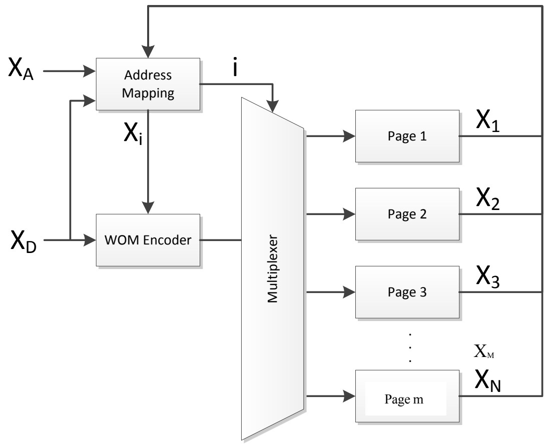


Fig. 4. RPU scheme. Pages 1 to m include the current physical target page and those pages of the retired-page pool being considered by the policy.

Analysis of this scheme with a q-state WOM code and m physical pages appears to require a Markov chain with q^m states. However, for deriving the probabilistic write capacity, we are able to reduce the state space.

Lemma 1: the CDF of the number of successful writes of random data with the aforementioned policy can be calculated from two Markov chains: the original q-state chain and an $\binom{m+l-1}{m}$-state chain, where l is the number of NF states in the (single-page) WOM code.

Proof: The 1st phase of the write policy is carried out independently m times, leaving each of the m physical pages in some NF state. The number of writes in this phase is thus the sum of m i.i.d. random variables, and the (NF) states of these m pages are also i.i.d. Therefore, and since page identity is unimportant, for the 2nd phase it suffices to know how many of the m pages are in each of the aforementioned l NF states. The number of states of the 2nd phase “NF” Markov chain therefore equals the number of ways of choosing m values out of l, with repetition and regardless of order, namely $\binom{m+l-1}{m}$. ■

Remark: writing to any given page in the 1st phase is stopped once reaching an NF state rather than upon failure. This is required for complete decoupling; specifically, for the data written in the first attempt to a new page as well as for the first data written in the 2nd phase to be independent of the final state of the previous page.

Lemma 2: the expected number of successful in-place random-data writes until reaching an NF state (inclusive) is:

$$E(X)=\sum_{t=1}^{\infty}\Pr(X\ge t)=\sum_{t=1}^{\infty}\left[1-\left(P_0M^t\right)_{Q'}\right],$$

where M is the single-page state-transition diagram modified such that an NF state is followed with probability 1 by a “Next” state (E.g., Fig. 5(a)), P_0 is the q-dimensional vector (1,0,…,0), Q' denotes the union of NF states, and $(P_0M^t)_{Q'}$ denotes the sum of the values of the vector P_0M^t at the indices corresponding to the NF states; i.e., the probability of first reaching an NF state after exactly t writes.

Proof: the above is a weighted summation (over the number of writes) of the probability of being in any of the NF states following that write. ■

Remark: although the summation is infinite, the terms diminish rapidly so it may be truncated.

Lemma 3: The probability of any given page being in NF state s at the end of the 1st phase is:

$$P(s)=\sum_{t=1}^{\infty}\left(P_0M^t\right)_s\Big/\sum_{s\in Q'}\sum_{t=1}^{\infty}\left(P_0M^t\right)_s$$

Proof: Summation of the probabilities of being in that state after t steps, along with the fact that M was modified so as to transition the state to “Next” right after reaching an NF state. The normalization accounts for being in non-NF states along the way. ■

Corollary 4: The initial state probabilities of the 2nd phase can readily be calculated from *P(s)*.

Proof: follows from the independence among the states of the pages at the end of the 1st phase. ■

Lemma 5: $E_{NF,}$ the expected possible number of writes in the 2nd phase, is given by the sum over the $\binom{m+l-1}{m}$ possible initial 2nd-phase states of the expected number of successful writes for each initial state, weighted by the probability of that state being the initial state.

Proof: The number of initial states is given in Lemma 1, and their probabilities are per Lemma 4. ■

Theorem 6: the expected number of successful writes of random data to m pages is

$$E=m\cdot\sum_{t=1}^{\infty}\left[1-\left(P_0M^t\right)_{Q'}\right]+E_{NF}$$

where M is the single-page Markov chain.

Proof: Follows directly from the lemmas and from the linearity of the mean, which renders the dependence of the 2nd phase duration on that of the 1st phase irrelevant. ■

Example 1. We now consider the aforementioned RPU policy in conjunction with the Rivest and Shamir WOM, writing of 2 bits twice in 3 cells, with m=2 distinct addresses ("pages"). Fig. 5(a) depicts the single-address Markov chain, highlighting the *l*=4 NF states (those that have an edge to the Failure state). The diagram has been modified, replacing Failure with Next, so that an NF state changes to the "Next" state with certainty in order to terminate the 1st phase for a page after reaching an NF state for the first time. "Next" means going to the next blank page, and in the case of the last page, it means going to the 2nd phase. The probabilities of the 1st phase ending in each of the NF states are all 0.25, and these determine the initial-state probabilities of the 2nd phase. The corresponding 2-page NF (2nd phase) Markov chain is depicted in Fig. 5(b). The expected number of writes is 2·3.29+0.76=7.34 writes, 11.5% greater than the number of writes to two addresses without RPU, namely 2·3.29=6.58. Note that the number of states for the 2nd phase is 11, much smaller than the 65 that would be required for two pages with a straightforward approach.

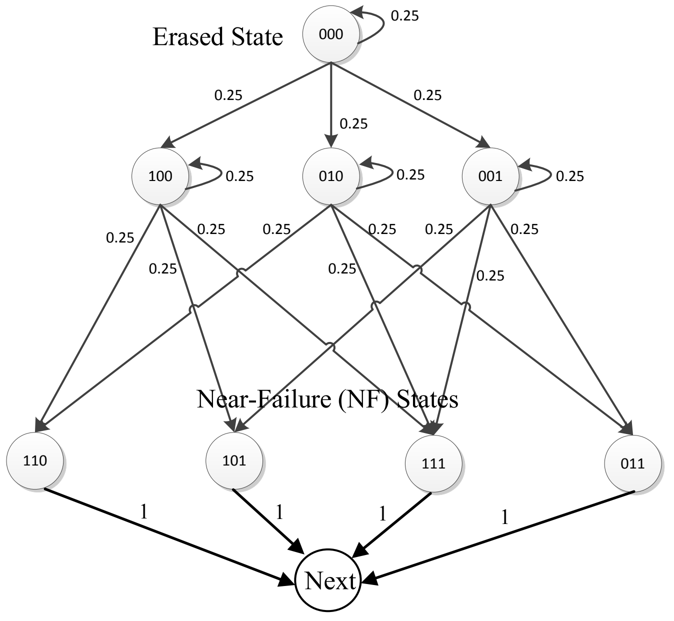
Example 2. Consider again the Rivest and Shamir scheme (Fig. 1), two pages (addresses), but for a different RPU policy. First, data is written the guaranteed number of times to the 1st address. Next, the same is done for the 2nd address. Next, write is attempted to the 1st address and, if fails, to the 2nd, The process ends when the data cannot be written in either one of the addresses. The full Markov chain of the described policy is shown in Fig. 6. The expected number of writes for this scheme is 9.833, as compared with 6.58 without RPU, a 50% increase. The CDF of this two-address scheme is shown in Fig. 7. Here, the analysis did not employ our 2-phase approach.

Remark: The above examples used a simple WOM code and tiny pages for illustration purposes. The numerical results are therefore not indicative of the achievable improvement for common page sizes using other WOM codes. In [15], simulation results of RPU with 2kB pages are provided for various WOM codes and RPP sizes. It is shown that RPU can more than double the write capacity in some cases.

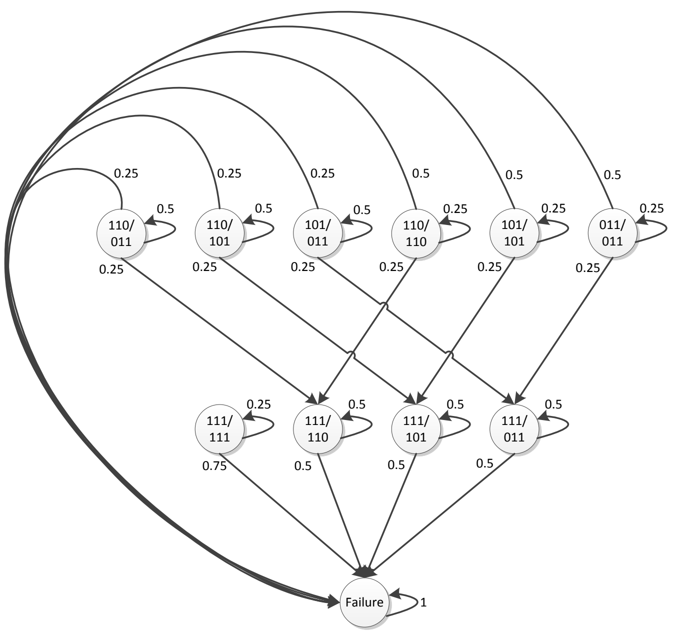
IV. Conclusions

Aggressive device scaling and switching to MLC architectures make write capacity a critical parameter in SSD design. Retired Page Utilization (RPU) offers significant benefits at virtually no storage cost when used in conjunction with codes that offer intra-page data representation flexibility. In this paper, we showed how to cast RPU as a cross-page coding scheme, and adapted Markovian analysis to construct a framework for such analysis of RPU. In so doing, we moreover showed how the RPU page-selection policy can sometimes simplify the analysis by reducing the required number of states. For example, in RPU with two addresses and [n=3, k=2, t=2] WOM, the number of states was reduced from 35 to 20. The evaluation was carried out for SLC, but both RPU and the analysis framework presented here are equally applicable to MLC.

Suggested topics for further investigation include a more comprehensive exploration of RPU parameters: RPP size, erase block size, WOM/ECC coding such as rank modulation and more, as well as retired-page selection policies. Efficient implementation architectures and performance studies for real benchmarks are also called for.



(a)



(b)

Fig. 5 2-page RPU in conjunction with Rivest and Shamir rewrite of 2 bits twice in 3 cells. (a) Phase-1 single-address Markov chain, showing state types; (b) 2-page phase-2 NF Markov chain.

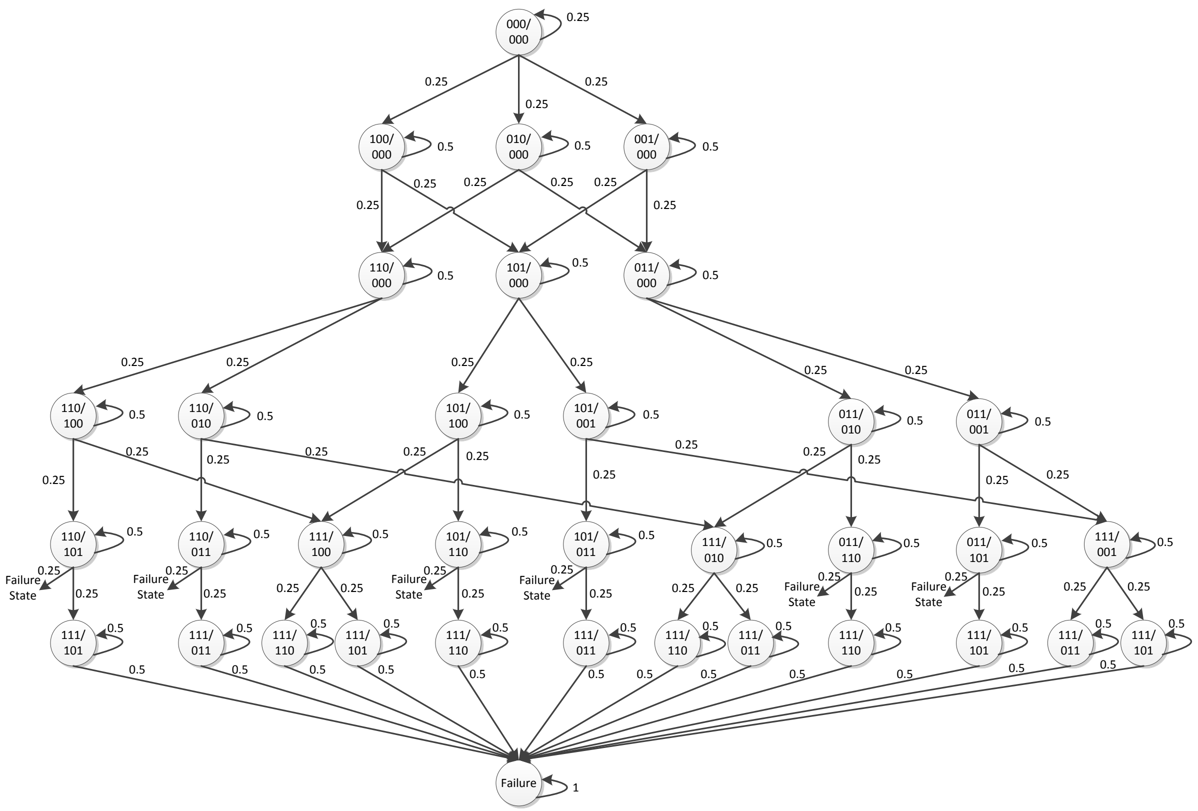


Fig. 6. Full Markov chain for RPU spreading on two addresses with Rivest and Shamir rewrite of 2 bits twice on 3 cells. Each state shows the cell contents of the two addresses. First, data is written the guaranteed number of times to the first address. Next, the same is done for the second address. Next, write is attempted to the first address, and if fails to the second, etc. Writing ceases when data cannot be written at any of the addresses.

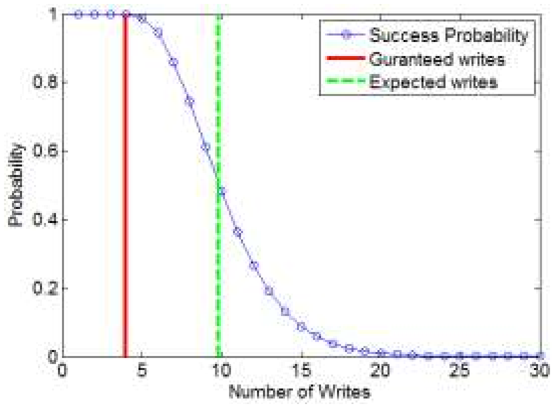


Fig. 7. CDF of two-address RPU in conjunction with Rivest and Shamir scheme (Fig. 1) for the RPU policy of Example 2.

Acknowledgment

This work was supported in part by HPI institute for scalable computing and by a grant from the Technion's Mitchell Entrepreneurship Fund.

References

[1] K. Prall, "Scaling non-volatile memory below 30nm", In Non-Volatile Semiconductor Memory Workshop (NVSMW), pages 5-10, 2007

[2] C. Trinh1et-al, "13.6 A 5.6MB/s 64Gb 4b/Cell NAND Flash Memory in 43nm CMOS", in IEEE Intl. Solid-State Circuits Conf. (ISSCC) 2009.

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

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

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

[6] G.D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," IEEE Trans. IT, vol. 32(5), pp. 697– 700, Sep. 1986.

[7] A. Jiang and J. Bruck, "Joint coding for flash memory storage," Proc. IEEE Intrnl Symp. on Info. Theo., pp. 1741–1745, Toronto, July 2008.

[8] S. Kayser, E.Yaakobi, P.H. Siegel, A.Vardy, and J.K.Wolf, "Multiplewrite WOM-codes," Proc. 48-th Allerton Conf. on Communication, Control and Computing, Monticello, IL, Sep. 2010.

[9] E. Yaakobi, S. Kayer, P.H. Siegel, A. Vardy and J.K. Wolf, "Codes for Write-Once Memories", IEEE Trans. Infor Theo (to appear).

[10] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank Modulation for Flash Memories", IEEE Trans. Information Theory, vol. 55, 2009.

[11] A. Jiang, "On The Generalization of Error-Correcting WOM Codes", IEEE Intl. Symp. Information Theory (ISIT), pp. 1391-1395, 2007.

[12] A. Berman, Y. Birk, "Probabilistic Performance of Write-Once Memory with Linear WOM codes – Analysis and Insights", IEEE Allerton, 2012.

[13] L.-P. Chang. "On efficient wear leveling for large-scale flash memory storage systems". In SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, 2007

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

[15] A. Berman, Y. Birk, "Utilizing Retired Pages for Improved Write Capacity of Solid-State Drives", Non-Volatile Memories Workshop (NVMW), 2013.