New rewriting schemes for MLC flash memory

Luoming Zhang, Amin Shokrollahi, *Fellow, IEEE*

Abstract

MLC flash memory is a non-volatile memory, in which the cells can be injected with different numbers of electrons. The unique characteristics of asymmetric writing/erasing operation and limit writing/erasing cycles of flash memory make the erasing operation expensive in terms of the lifetime of flash memory. We present a new rewriting scheme to increase the writing operations between two consecutive erasing operations by making better usage of the charge levels in the cells. The scheme can get better rewritten times without expensive modifications to the normal flash memory architecture.

I. Introduction

FLASH memory is a non-volatile memory that can be both electrically programmed and electrically erased. There are two major types, NOR and NAND, differentiated by the matrix structures of storage units [1]. In recent years, the NAND flash memory has been experiencing a revolutionary development in the storage area because of the continuing scaling in the cost per GByte, especially by using Multiple Level Cells (MLC) [2]. Our paper will mainly focus on NAND MLC flash memory. In the following, flash memory means NAND MLC flash memory if not mentioned otherwise.

The unique characteristics of flash memory, such as low-latency IOPS and low power consumption lead to the increasing potentials of the application in both consumer and enterprise systems. However, the limited rewriting endurance of the flash memory, which is typically 10^4 for MLC flash memory, greatly slows its adoption to all the storage architectures. Moreover, depending on different workloads, the lifetime of the flash memory is greatly reduced. According to Micron's datasheet, the newest product RealSSD C400 with a capacity of 128GB actually has only a capacity of 72TB that can be written under the assumed conditions [3]. This is almost a reduction by a factor of 15 given that the program/erase cycle for MLC cells is 10^4. Therefore, how to prolong the lifetime of flash memory is a hot topic nowadays.

There have been many works done on the storage system level to improve the endurance of flash memory based Solid State Device (SSD). One of the main directions is using hybrid storage architecture using HDD and SDD. Soundararajan *et al.* [4] used HDD as write cache for SDD which could save writes to SDD up to 50% in order to prolong the lifetime of SSD. The

The work is supported by ERC grant ECC SciEng of Amin Shokrollahi

The authors are with ALGO, EPFL

other direction is trying to balance the writes to SSD on pages or block levels, called wear leveling technique [5]. In flash memory, there is an intermediate layer called Flash Translation Layer (FTL), which functions as an indirect mapping between the Logical Block Address (LBA) and Physical Block Address (PBA). The generic structure of FTL is the Log-Structured File system (LSF), in which several parameters of the basic operation units such as write times, validity and so on [6]. Based on these parameters, the write to the device are pointed to the less frequent written units, so as to average the write frequencies of the whole operation units in flash memory. The lifetime of flash memory could be effectively extended, by jointly considering the workloads, wear leveling policies and garbage collecting policies [7]–[10]. However, these schemes have a drawback that the lifetime of flash memory is extended by sacrificing the actual capacities of the devices. As Hu *et al.* [11] analyzed, even considering the static and dynamic workloads, these schemes may suffer write amplification as high as 3 in some cases.

On the other hand, some researchers work on the data representation in flash memory to make the write-constraint flash memory to be "write-free" devices. These works were first inspired by Rivest and Shamir [12]. They modeled some memories as Write-Once Memory (WOM), which means the basic storage unit could only transit to "1" state from "0" state but not vice versa. The WOM codes use a group of these basic units to represent the data of several variables which could be freely transited from "1" state to "0" state and vice versa, though the times of changing (rewriting) the variables maybe limited. People are trying to maximize the number of variables and the rewriting times by using the limit number of the basic storage units [13]–[18]. Meanwhile, people try to design WOM codes with error-correcting capability [19]–[21]. Recently, Jiang *et al.* [22] developed floating codes. Floating codes not only could generalize the WOM codes, but more important, they could achieve very high rewritten times for two or three variables. Even though, floating codes are not perfect schemes, such as the complicated mapping scheme prevent practical encoding and decoding the variables. We are trying to solve these problems by using a so-called water-filling scheme.

In the following of this paper, some fundamentals of flash memory will be given in section II, besides a simple example of the new scheme. And in section III, we will formulate the writing and reading operation of the new scheme. The scratched implementation of our scheme will be given at the end of the section. At last, performance analysis will be evaluated by comparing our scheme with other schemes.

II. Fundamentals and a simple construction

In this section, we will first introduce some fundamentals of working principles in flash memory. Later, we will explain the motivation of our new scheme. Before ending the section, we will give a simple example to show the initial motivation and how it works for our new scheme.

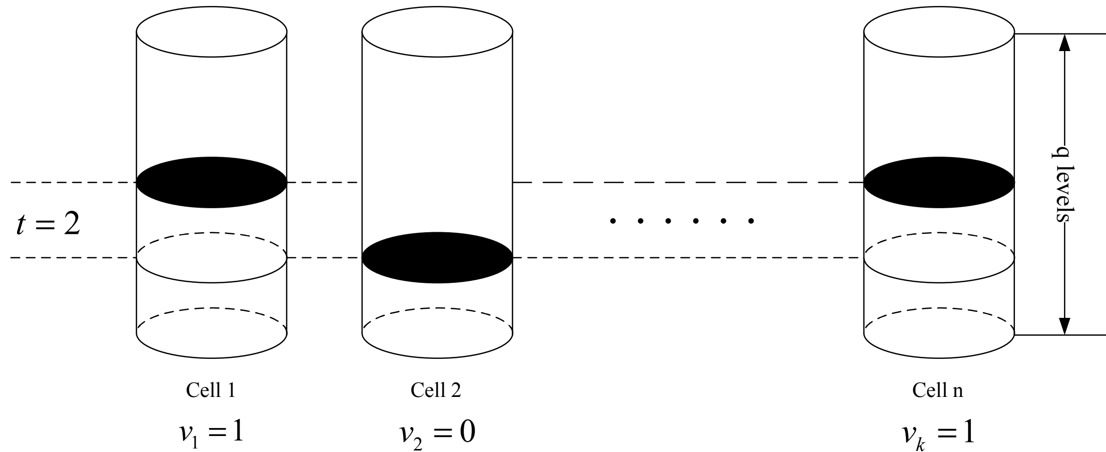

Fig. 1. A simple example of water-filling scheme. The write count is 2, the size of the variable is 2 , the charge level of the cell is q , and the n cells.

A. Fundamentals of Flash memory

In flash memory, the floating-gate transistor is the basic storage unit [1], [2]. The transistor is also called cell. By using hot-electron injection mechanism or the Fowler-Nordheim tunneling mechanism, the charge (i.e. electrons) are trapped in (programmed) or released from (erased) the floating gate of each cell. The charge could be measured by some discrete levels of voltage, called thresholds. The charges can be trapped in the floating gate without powering on, which makes flash memory a non-volatile memory. The combinations of various measurement results from several cells are used to represent the value of variables (data). The operation of trapping (or releasing) charge is called writing (or erasing), while the measurement of the charge levels is called reading. The flash cells are arranged in the matrix-like structures, which are called blocks. Inside each block, there are pages, which are the elementary writing/reading granularities in flash memory. Every page, usually contains 2kB cells to store data, and is extended with additional spare bytes, called metadata. In order to keep the low latency property and increase the cell density of flash memory, it has a unique characteristic that the erasing operation is based on blocks while the writing operation is based on pages. Meanwhile, the cells have limited program/erase cycles which typically is 10^5 times for single level cells (SLC) and 10^4 times for MLC. This brings great challenges when using flash memory to design SSD. For instance, in the primitive operation, it's necessary to erase the whole block and then rewrite it again except the target cell if there is just an update of the data stored in the cell. This operation will induce huge inefficiency not only on the writing/erasing operations, but also greatly reduce the lifetimes of other cells in the same block. There are various ways that could be and have been done to exploit the capacity of inefficient operations in order to prolong the lifetime of SSD. In this paper, we are going to exploit the capacity of the MLC cells from the angle of data representations so as to extend the lifetime of flash memory.

B. Motivation

In the traditional scheme, flash memory usually erases each MLC cell after each write and don't care which charge level of the cell it was written. This is not efficient in term of using the charge level to represent data (variable values), because the cell will get erased even if the cell is only charged to the lowest level. Jiang *et al.* designed floating codes by carefully changing the cell charge level when updating the variable values. Floating codes achieved great improvement on the rewritten times. Nothing is perfect. Floating codes have some drawbacks: First, the mapping between the cells' states and variable values is complicate, especially when the number of variables is more than three, which make the reading of data very hard; Second, it is necessary to have the knowledge of the exact charge levels of all the cells during reading the variable values even for small number of variables, which brings more routing complexity in the physical layer when parallel reading thousands of cells; Third, according the updating rules of floating codes, each write could only change the value of one variable.

C. A simple example

In order to solve the problems, we proposed a new data representation scheme. Here is the simple example of the scheme in the case of using one cell to represent data. As usual, the variable value is represented by the charge level of the cell in the example. If the value is binary, then one charge level is enough to differentiate the two states of the variable. Let's define the comparatively low charge level to denote 0 while the high one to 1. After every write, we count the write time starting from the latest erasing operation of the cell and store the counts in the metadata of the page. When there is a new writing operation, the charge level of the cell increase to the desired level which depends on the writing counts and the new value of the variable. For instance, if the previous write count is t , the charging target level will be $t+1$ if the new variable is 1, the level will be t if the new variable is 0. When there is a reading operation, the system first read the write counts stored on the page and the use the corresponding detecting threshold to detect the charge level of each cell whether it's high or low and interpret the corresponding variable value. Fig. 1 shows the example where the write count is two.

In the new scheme, the charge levels are keeping increasing after every write and the differences of charge levels between the neighboring cells are limited, like a shifting windows moving towards the highest charge level. We jointly use write counts and windowed charge levels of the cells to represent the variable value. The updating rule of cells' states in the new scheme is like filling the water into the containers and the average water level inside the container is increasing after every write so we name it the water-filling scheme.

III. Writing and reading of the water-filling scheme

In this section, the general writing and reading scheme will be given. At first, we will introduce some definitions to facilitate our description. Then we will describe how the water-filling scheme works for the case that the variable values are stored in one cell. And later, we will extend the one cell case to the multiple cells case. At last, we will discuss some practical implementation issues about the water-filling scheme.

A. Definitions

In order to reduce the ambiguity in description as much as possible, we follow the same definitions in Jiang's work [23] if there are overlaps.

Definition 1:

We use n cells of q charge levels to store k variables of alphabet size l. Meanwhile, we use T to denote the rewritten times in terms of how many writing operations between two consecutive erasing operations and t to denote the write counts beginning from the latest erasing operation on the cell, where $t \in \{1, 2, \cdots, T\}$. If c_i denotes the actual charge level, d_i denotes the relative charge level, and v_i denotes the variable value, then after t writes:

- $(c_1, c_2, \cdots, c_n)_t \in \{0, 1, \cdots, q-1\}^n$ denotes the state of flash memory with cells;
- $(d_1, d_2, \cdots, d_n)_t \in \{1, 2, \cdots, \Delta\}^n$ denotes the relative charge level to the lowest charge level in every write for n cells, where Δ is called the highest window height of n cells;
- $(v_1, v_2, \cdots, v_k)_t \in \{0, 1, \cdots, l-1\}^k$ denotes the values of k variables.

According the unique characteristic of flash memory, if $q \geq p$, and the state of the flash momory after the qth and pth write is $\left(c_1^{'}, c_2^{'}, \cdots, c_n^{'}\right)_q$ and $(c_1, c_2, \cdots, c_n)_p$respectively, then $c_i^{'} \geq c_i, i \in \{1, 2, \cdots, n\}$.

Definition 2:

We use f_u and f_d denote the writing function and the reading function of the water-filling scheme, and then they could be defined as following:

$$f_u : (c_1, c_2, \cdots, c_n)_{t-1} \rightarrow (c_1, c_2, \cdots, c_n)_t \tag{1}$$

$$f_d : (c_1, c_2, \cdots, c_n)_t \rightarrow (v_1, v_2, \cdots, v_k)_t \tag{2}$$

B. Single cell case

From the example given in section II, we could summarize principles of the water-filling scheme:

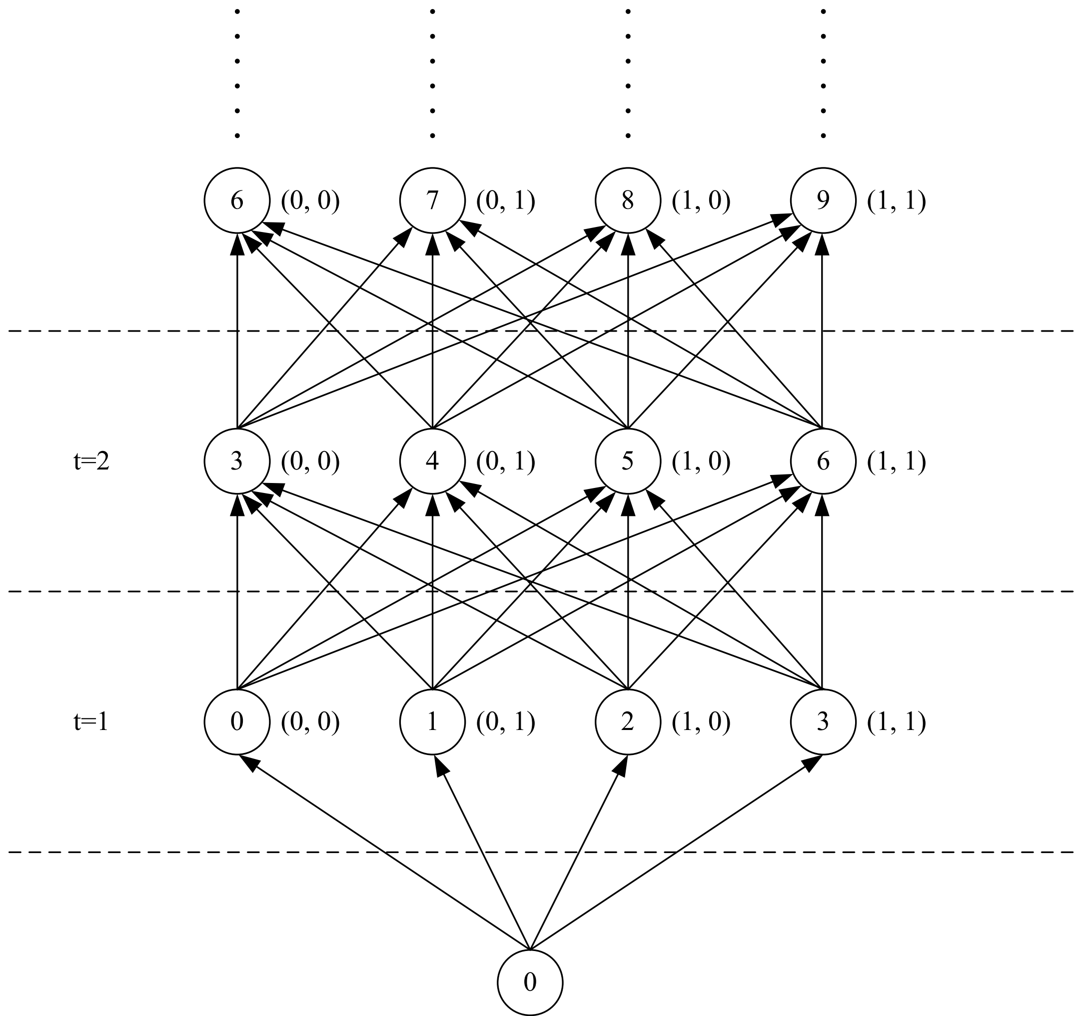

Fig. 2. The transition diagram for Water-filling scheme ($n = 1$, $k = 1$ and $l = 4$). The number inside the circle is the charge level; aside the circle is the corresponding binary value of the variable.

1) *The write counts are recorded and updated after every writing operation;*
2) *The consecutive windows share one frame (charge level);*
3) *Given* n,k *and* l*, the height of the shifting window in terms of charge levels are the same in every writing operation, i.e. equal to* $\Delta + 1$*;*
4) *The basis charge level of every writing operation, i.e the lowest charge level of all the possible states of the cell, is determined by write count, which is* $\Delta \times (t - 1)$.

Fig. 2 is the transition diagram for water-filling scheme when $n = 1$, $k = 1$ and $l = 4$. We can see the windows height is 3 and every variable is freely transited to all the possible variables in every write. The scheme could make full usage of the charge levels so as to increase the rewrite times. Moreover, the consecutive rewriting operation shares one charge level by jointly use the write counts. For the specified case in Fig. 2, every write of variables of alphabet size 4 only needs to increase

3 charge levels.

In the case of one cell data representation, the variable values are represented by the charge levels in one window of the cell. In order to include all the possible combinations of variables, the necessary number of unique charge levels will be l^k for each write. So the height of window should not be lower than l^k, i.e. $\Delta + 1 \geq l^k$. The writing function for one cell case is as following:

$$(c_i)_t = f_u\left((v_i)_t, t\right) = \Delta \times (t-1) + m\left((v_i)_t\right) \tag{3}$$

Where $m(\cdot)$ is a bijective function between the variable values and the relative charge levels. Reading operation is detecting the relative charge levels in the cell. Before detecting the relative charge level from the cell, the basis charge level is set up with the knowledge of the write count t. Then several thresholds of charge levels are added to the basis charge level to detect the relative charge value $(d_i)_t$.

Once the relative charge level of the cells is known, the variable values could be demapped. The reading function for one cell case is as following:

$$(v_i)_t = f_d\left((d_i)_t\right) = m^{-1}\left((d_i)_t\right) \tag{4}$$

Where $m^{-1}(\cdot)$ denotes the reverse function of $m(\cdot)$.

For the single cell case, it's obvious that the maximum rewritten times T is as below:

$$T = \left\lfloor \frac{q-1}{\Delta} \right\rfloor = \left\lfloor \frac{q-1}{l^k - 1} \right\rfloor \tag{5}$$

We found the rewritten times are exponentially decreasing when the number of variables increases. To solve this problem, we extend the water-filling scheme from data representation by one cell to jointly using multiple cells.

C. Multiple cells case

In the Multiple cells case, the variable values are jointly stored in a group of cells. The principles of writing and reading for the multiple cells case are similar to the single cell case, except that the mapping function between the relative charge level and variable value becomes more complex. We set height of the shifting windows in all cells to be equal to Δ considering the maximum rewritten times T is decided by the cell with the fastest rising of charge levels in the group, though the windows height of each cell in the group could be different. Let's use $M(\cdot)$ denote the bijective mapping from the relative charge level array to variable values and $M^{-1}(\cdot)$ denote the demapping function. Then the writing and reading function could be changed as below:

$$(c_i)_t = f_u\left((v_1, v_2 \cdots, v_k)_t, t\right) = \Delta \times (t-1) + M\left((v_1, v_2 \cdots, v_k)_t\right) \tag{6}$$

$$\left(v_i\right)_t = f_d\left(\left(d_1, d_2, \cdots, d_n\right)_t\right) = M^{-1}\left(\left(d_1, d_2, \cdots, d_n\right)_t\right) \tag{7}$$

Under such writing and reading mechanism, the write time T is as below:

$$T = \left\lfloor \frac{q-1}{\left\lceil l^{k/n} \right\rceil - 1} \right\rfloor \tag{8}$$

The detailed proof could be seen from the appendix.

D. Implementation issues

One of the advantages of the water-filling scheme is that the implementation of the scheme could be applied to the normal flash memory without lots of changes. Let's scratch the main changes as below:

1) First, we need reserve several bits on the page metadata area to store the write counts. According to equation (5) and (8), the necessary bits is less than $\log_2(q-1)$;
2) Then, we need some bytes on the system level to store the mapping tables of $m(\cdot)$ or $M(\cdot)$, and also store the demapping tables of $m^{-1}(\cdot)$ or $M^{-1}(\cdot)$.

- **Writing operation:**

  According to the writing function specified in equation (3) and (6), the write count is read out and used for calculating the basis charge level for next write, i.e $t \times \Delta$if the previous write cout is $t-1$ and the shifting windows height is Δ. The corresponding relative charging level of each cell are calculated by the mapping tables of $m(\cdot)$ or $M(\cdot)$. The target programming charge level of each cell is the addition of the relative charging level and the basis charge level. Once the programming of all the cells completes, the write count will be updated at the end of the writing operation.

- **Reading operation:**

  According to the reading function in equation (4) and (7), we need to detect the relative charge level of each cell by $\lceil \log_2(\Delta+1) \rceil$ detections and then use the demapping table to decode the variable values. Considering the writing disturb caused by over charging [24], the detection charge level should be one level higher than the tested threshold. By carefully designing the demapping tables, it is possible to read one variable value from each detection.

IV. Performance analysis

The water-filling scheme could increase the rewritten times of flash memory cells under comparatively low complexity which make the scheme very efficient in prolonging the lifetime of flash memory based SSD and practical to implement. We are going to discuss the performance of the scheme from the aspect of rewritten times, writing efficiency and reading complexity.

- **Rewritten times**

From (8), the Rewritten times are determined by k, l and n, given the constant charge levels q that the cells could be programmed. In the binary variable case, i.e $l = 2$, the write time is determined by k and n actually. It implies that there is a tradeoff between the capacity and the lifetime of the flash memory device: If ${}^{k}\!/_{n} > 1$, which means the capacity is larger than the number of cells, the lifetime becomes shorter; Othervise, vice versa.

Compare to floating codes, the water-filling scheme could get approximately equal or more rewritten times, especially for small k, l and n which is more practical for application. In [22], it is said the write time of changing one bit of the variable that could get by using floating codes is as following:

$$(n-1)(q-1) + \left\lfloor \frac{q-1}{2} \right\rfloor \tag{9}$$

When $n = 1$, $k = 1$ and $l = 4$, which is the case of single cell, floating codes could get $\left\lfloor \frac{q-1}{2} \right\rfloor$ rewritten times. Considering that it takes two write to change the whole value of variable, the rewritten times of the variable are $\left\lfloor \frac{q-1}{4} \right\rfloor$. According to (8), the water-filling scheme could get $\left\lfloor \frac{q-1}{3} \right\rfloor$ rewritten times of changing variable values, which actually is better in some q values.

When $n = 2$, $k = 3$, and $l = 2$, the water-filling scheme could get $\left\lfloor \frac{q-1}{2} \right\rfloor$ rewritten times of changing variable values, which is the same as floating codes.

- **Writing efficiency**

Flash memory is programmed by means of the Incremental Step Pulse Programming (ISPP) [25]. The target programming threshold of each cell is reached by small voltage steps which applied to the gate of the cell. The higher target programming threshold of the cell it is, the more voltage steps it needs to reach the target which leads to longer programming time. So in the range of a page programming, the programming time is determined by the cell with highest target threshold in the normal flash memory. In the water-filling scheme, the programming time is determined by the windows height, which is usually lower than the highest charge level that the cells could reach. So the programming of the page could be faster (more efficient) than the normal flash memory by using the water-filling scheme.

- **Reading complexity**

In the water-filling scheme, the reading operation only needs the relative charge level of the cell given the write counts is known on the system layer. Compared to the normal flash memory or floating codes, where the absolute charge level of the cell is needed, the bit width that needed to represent the charge level becomes shorter in the water-filling scheme. This brings more reduction on the routing complexity in the physical layer of flash memory. We could have faster reading speed by using the water-filling scheme.

Generally speaking, it's possible to get improvement on the lifetime, writing efficiency and reading speed through the water-filling scheme.

V. Conclusion

In the paper, we proposed a new rewriting scheme which could better use the charge levels of the cells to represent data. By jointly increasing the charge levels of a group of cells, flash memory could get approximately optimal rewritten times in order to get longer lifetime. The average charge levels of the cells could reach as high as possible before the cells get erased. So the new scheme is quite efficient in using the charge levels to represent data. Meanwhile, by using the write count mechanism, the writing and reading operation could become faster than normal flash memory schemes.

Appendix A
Proof of the Rewritten times for multiple cells case

As described in section III, we use the combinations of various charge levels in several cells to represent the variable value. So the number of the whole combinations (states) should be larger than the total size of the variables. Let's assume we use n cells of q charge levels to represent k variables of alphabet size l. Since the k variables are independent, so the total size of variables is l^k.

According to the writing principles, the basis charge level will increase the same level for all the cells after every write. If the windows height is Δ, then number of combinations that n cells could have is $(\Delta + 1)^n$, which should satisfy the following:

$$(\Delta + 1)^n \geq l^k \tag{10}$$

So the windows height should be $\Delta = \left\lceil l^{k/n} - 1 \right\rceil$.

Then the rewritten times T of the multiple cells case is:

$$T = \left\lfloor \frac{q-1}{\lceil l^{k/n} \rceil - 1} \right\rfloor \tag{11}$$

Acknowledgment

The authors would like to thank...

References

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, 2003.

[2] M. Sanvido, F. R. Chu, A. Kulkarni, and R. Selinger, "Nand flash memory and its role in storage architectures," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1864–1874, 2008.

[3] Micron, "C400 1.8-inch sata nand flash ssd." [Online]. Available: http://www.micron.com/get-document/?documentId=6419

[4] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX conference on File and storage technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 8–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855511.1855519

[5] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 138–163, 2005, 1089735.

[6] M. Rosenblum and J. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 1, pp. 26–52, 1992.

[7] L. Chang, T. Kuo, and S. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 4, pp. 837–863, 2004.

[8] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. [Online]. Available: http://portal.acm.org/citation.cfm?id=1404014.1404019

[9] F. Chen, T. Luo, and X. Zhang, "Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proceedings of the 9th USENIX conference on File and storage technologies*, 2011.

[10] Y. Pan, G. Dong, and T. Zhang, "Exploiting memory device wear-out dynamics to improve nand flash memory system performance," in *Proceedings of the 9th USENIX conference on File and storage technologies*, 2011.

[11] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 10:1–10:9. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534544

[12] R. Rivest and A. Shamir, "How to reuse a" write-once" memory," *INFO. CONTR.*, vol. 55, no. 1, pp. 1–19, 1982.

[13] J. Wolf, A. Wyner, J. Ziv, and J. Korner, "Coding for a write-once memory," *AT&T Bell Laboratories technical journal*, vol. 63, no. 6, pp. 1089–1112, 1984.

[14] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *Information Theory, IEEE Transactions on*, vol. 30, no. 3, pp. 470–480, 1984.

[15] F. Merkx, "Womcodes constructed with projective geometries," *TS Traitement du signal*, vol. 1, pp. 227–231, 1984.

[16] G. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories (corresp.)," *Information Theory, IEEE Transactions on*, vol. 32, no. 5, pp. 697–700, 1986.

[17] S. Kayser, E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf, "Multiple-write wom-codes," *Proc. 48th Annual Allerton Conferenece on Communications, Control and Computing*, 2010.

[18] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Efficient two-write wom-codes," in *Information Theory Workshop (ITW), IEEE*, 2010, pp. 1–5.

[19] G. Zemor and G. D. Cohen, "Error-correcting wom-codes," *Information Theory, IEEE Transactions on*, vol. 37, no. 3, pp. 730–734, 1991.

[20] E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "Multiple error-correcting wom-codes," in *Information Theory Proceedings (ISIT), IEEE International Symposium on*, 2010, pp. 1933–1937.

[21] A. Jiang, “On the generalization of error-correcting wom codes,” in *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, 2007, pp. 1391–1395.

[22] A. Jiang, V. Bohossian, and J. Bruck, “Rewriting codes for joint information storage in flash memories,” *Information Theory, IEEE Transactions on*, vol. 56, no. 10, pp. 5300–5313, 2010.

[23] A. Jiang and J. Bruck, “Data representation for flash memories,” in *Data Storage*. In-Tech Publisher, 2010.

[24] N. Mielke, T. Marquart, W. Ning, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, “Bit error rate in nand flash memories,” in *Reliability Physics Symposium, 2008. IRPS 2008. IEEE International*, 2008, pp. 9–19.

[25] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND flash memories*. Springer Verlag, 2010.