International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.9 (2014), pp.381-396
http://dx.doi.org/10.14257/ijmue.2014.9.9.38

The K-Partition Flash Code with BIFC-based Sharing and some Variants

Riz Rupert L. Ortiz and Proceso L. Fernandez, Jr.

Ateneo de Manila University
rizortiz@gmail.com, pfernandez@ateneo.edu

Abstract

Flash codes are used to handle decoding and encoding of digital information to flash memory devices. The performance of a flash code is usually evaluated using the write deficiency metric. This paper introduces the K-Partition Flash Code (KPFC) with BIFC-based sharing and explores some of its variants KPFC is a coding scheme that involves a sharing mechanism within partitions of a flash memory block. The technique was designed to allow more cell writes to flash devices in order to improve its performance by lowering its write deficiency. Computer simulations were conducted to estimate the average case performances of the flash codes. Simulation results showed that its performance is generally better than the flash codes in literature.

Keywords: *block erasure, block-write, flash code, flash memory*

1. Introduction

Flash memory is a non-volatile semi-conductor storage device. Flash memory has a wide variety of applications nowadays. It has become increasingly important that it is even utilized significantly in some embedded systems used by intelligent applications such as household appliances, telecommunication devices and other high technology machinery. It has become a dominant nonvolatile memory because it is cheap, fast, and reliable. Moreover, flash memory can be electrically programmed and erased with relative ease [1].

Flash memory has a hierarchical structure organized into blocks, where a block can contain thousands of memory cells. The memory cell is the smallest unit for performing read and write operations to flash memory. Each cell, which has multiple possible states, can store electric charges using some rules provided from a coding scheme that can also retrieve the data from the values stored in the flash memory. This coding mechanism is referred to as floating codes or commonly known now as *flash codes* [2, 9].

The process of adding a charge in a cell is called cell programming [4] which is typically done through *electron injection* [7]. Using the encoding function from some coding scheme, the level of charge of a cell can be increased. In multilevel flash memories, there can be two possible problems that may occur when performing cell writes: errors when too many electrons are added (overshoots) or errors when too few electrons are added (undershoot). There is difficulty in lowering a charge to flash memories. Thus, overshoots are more of a problem than undershoots [6].

Decreasing the charge of a cell is very costly and should be avoided as much as possible. While it is easy to raise the level of charge in a cell, to lower it is difficult and time consuming. This property of flash memory is referred to as *write asymmetry* [2]. In fact, lowering a charge in a single cell only is not allowed. Such lowering of charges can only be made through a *block erasure*, the process that erases the charges of all the cells within a

ISSN: 1975-0080 IJMUE

memory block. Unfortunately, flash memory is constrained by a limited number of erase cycles. Currently, typical flash memory can only allow up to about 10^5 block erasures [2], although there are already some flash devices that are capable of 10^6 block erasures [5]. The block erasure process is slow and, after some repetitions, causes the memory chip to wear out. Thus, block erasures significantly reduce the longevity, reliability and speed of flash memories [7]. Continued block erasures will make memory blocks unreliable for storing data and will eventually damage the flash memory when the block erasures exceed the allowed number of erase cycles.

One way of extending the lifespan of flash memory is to improve the hardware technology to allow for more block erasures. Another approach is to design flash codes that are efficient so that more cell writes can be made to flash memories before calling a block erasure. The focus of this study is to come up with efficient coding schemes that can increase the lifespan of flash memories. Such coding schemes can delay the occurrence of block erasures and, ultimately, prolong the lifespan of flash memory. The most salient point is to maximize the number of write operations or bit updates and delay the occurrence of a block erasure.

2. Preliminaries

Flash memory is composed of a number of blocks. A block in particular is comprised of thousand of cells, the exact number of which is denoted by n in this paper. The typical values of n are between 2^{18} and 2^{22} [8]. In multilevel flash memories, each cell can be in one of q levels from a finite set $A_q = \{0,1,\ldots, q\text{-}1\}$. Thus, a block can be abstractly represented as vector $C = (c_0, c_1,\ldots, c_{n-1})$, where $c_i \in \{0,1,\ldots, q\text{-}1\}$. The parameter q ranges from 2 to 256 [8]. Values of every cell can be increased or decreased by injecting a charge through the process called the Fowler-Nordheim tunneling mechanism or hot-electron injection mechanism [7], where electrons are trapped and in return determine the threshold voltage of the cell. The number of trapped electrons concentrates around q discrete levels that correspond to the q cell states [2]. A cell with a charge of 0 is *empty*, while a cell with a charge of q-1 is *full*. A cell that is neither empty nor full is said to be *active*.

The information vector encoded in a flash memory block, is a k-bit data $D = (d_0, d_1, \ldots, d_{k-1})$, where $d_i \in \{0,1\}$ and $k < n$. A block using binary cells normally stores 64, 128, or 256 kilobytes of data [2, 3, 10].

The mechanism to store the k-bit data D to the block state vector C is managed by some flash code F as implemented in the flash memory. Formally, the flash code $F = (\mathcal{E}, \mathcal{D})$ is a coding scheme consisting of two main functions. The encoding function $\mathcal{E}(i,C)$ provides the rules on writing a new state to the block given the index i of the data bit d_i that needs to be updated and the current state C of the block. The decoding function $\mathcal{D}(C)$ interprets the content C of the block into the corresponding k-bit data D [2].

Flash code operates on both the cell and the block levels. The metric normally used to evaluate the performance of the flash code is its *write deficiency.* It is defined as

$$\delta(F) = n(q - 1) - t \tag{1}$$

where $n(q-1)$ is the theoretical maximum number of cell writes allowed for a block of n cells with q levels each, while t is the actual number of write operations that the flash code F is able to perform before calling a block erasure [11]. Alternatively, the write deficiency ratio to compare flash codes can be expressed as

$$\hat{\delta}(F) = \frac{n(q-1)-t}{n(q-1)} \quad (2)$$

The range of possible values is [0,1], with $\hat{\delta} = 1$ as the worst performance and $\hat{\delta} = 0$ as the ideal one.

3. Related Literature

Flash codes are recently attracting the attention of researchers, especially in the area of computer science. More studies are coming out because of the possibilities that can be made in developing and improving new and efficient coding schemes. This section provides background on some flash codes in literature.

3.1 Index-less Indexed Flash Code and Layered Index-less Indexed Flash Code

The index-less Indexed Flash Code (ILIFC) and Layered Index-less Indexed Flash Code (LILIFC) are two of the popular flash codes in literature. Both coding schemes operate on sub-blocks called *slices*. A block of n cells is partitioned into equally sized slices with k cells each. ILIFC and LILIFC offer an elegant scheme of storing both the bit index and bit value from the active slices. As to performance, while both flash codes have the same worst case write deficiency, LILIFC performs better than ILIFC in the average case. The asymptotic worst case write deficiency of both flash codes is $O(k^2q)$. Refer to [8,11] for more detailed discussions on how these two flash codes manage the decoding and encoding of data to flash memory.

3.2 Bimodal Flash Code and 2-Split Bimodal Flash Code

The Bimodal Flash Code (BMFC) of Esling *et al.*, uses two modes of encoding [12, 13]. Instead of having k cells per slice, the flash code reduced the slice size to $k/2$, effectively lowering the write deficiency and improving its performance. Results of the study showed that BMFC returned a better average case performance than ILIFC and LILIFC. A variant called 2 Split BMFC (BMFC2) further reduced the size of slice to $k/4$, which also resulted to a better average case performance than BMFC. The worst case write deficiency of BMFC and BMFC2 is $O(k^2q + n/k)$. Both flash codes showed that performance can be improved by utilizing a reduced size of slices.

3.3 Binary Indexed Flash Code and BIFC-RCM

Tan and Kaji introduced the Binary Indexed Flash Code (BIFC) [14]. Similar to BMFC, the Binary Indexed Flash Code uses fewer cells for each slice. Specifically, the slice has a size $s = O(\log k)$, where s is the smallest even number satisfying $s \geq \lfloor 1+\log_2(k+1) \rfloor$. When k is sufficiently large, results of the study revealed that BIFC has a better write deficiency than ILIFC, LILIFC, BMFC and BMFC2 [12-14]. However, there is an overhead deficiency of s-2 for every slice, because of the implicit indexing. To mitigate this problem, another flash code was proposed by Tan and Kaji to address this overhead issue. It is called the BIFC with Resizable Cluster Method (BIFC-RCM) [15, 16]. BIFC-RCM uses the BIFC technique that capitalizes on smaller slice size and incorporates a growing slice to mitigate the effect of the overhead problem. The BIFC and BIFC-RCM have asymptotic worst case write deficiencies of $O(qk \log k + n)$ and $O(qk \log k + rk \log k \log((n/k) \log k))$, respectively.

3.4 Simple Segmentation Flash Code and the Phoenix Flash Code

In Simple Segmentation Flash Code (SSFC), a block is divided into equally-sized sub-blocks, called *segments*. The block $C = (c_0, c_1, ..., c_{n-1})$ has exactly $\lfloor n/k \rfloor$ segments. The ith segment ($0 \leq i < \lfloor n/k \rfloor$) corresponds to the vector $S_i = (c_{ik}, c_{ik+1}, ..., c_{ik+k-1})$. Within each segment, the jth entry ($0 \leq j < k$) is assigned to the jth bit. SSFC at its best case has a very good performance when each of the k bits has uniform or equal frequencies for write operations. On the other hand, Phoenix Flash Code (PFC) is a novel scheme that allows the process of absorption and revival operations. Similar to SSFC, the erase block $C = (c_0, c_1, ..., c_{n-1})$ has exactly $\lfloor n/k \rfloor$ segments, where the jth element of the segment corresponds to the jth bit. The idea of introducing absorption stems from the fact the SSFC has a very high write deficiency in the worst case. SSFC has an asymptotic worst case write deficiency of $O(nq)$ [17,18], while PFC has $O(k^2q + n)$ [19].

4. Computer Simulation for Average Case Write Deficiency

This section describes the various procedures involved in the completion of this study. Figure 1 illustrates how the study is organized and subsequently conducted. Basically, it involves three phases:

1. Benchmarking of some Flash Codes in Literature - The first phase refers to the implementation of some flash codes in literature to provide benchmark results relative to their performance. Write deficiency ratios were returned by the computer simulations and were later used for comparison. Flash codes with promising performance were taken into consideration to gain some insights for the design and development of the proposed flash codes.

2. Design and Development of the Proposed Flash Codes - After careful studies on the strengths and weaknesses of some flash codes in literature, some key concepts were integrated to the proposed flash codes. The developed flash codes were thoroughly tested to ascertain the correctness of the implementation.

3. Performance Analysis - In this study, computer simulation was used to estimate the average case performance of the flash codes. Computer simulations were implemented in java (see Table 1 for the parameters used in the simulations). Empirical results for the average case performances of the flash code were returned by the simulations, which are set to run in 30 experiments with fixed parameters of n=2048 and q=8. The experiments were performed for various k values from 4 to $n/2$, in increments of 4.

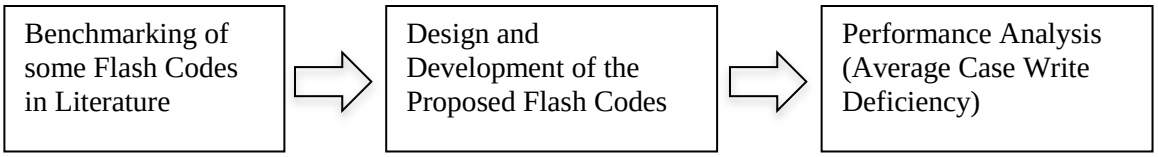

Figure 1. The Methodology of the Study

Table 1. Parameters used in the Computer Simulations

Variable	Values	Description
n	2048	Size of the Block
k	4,8,12, …, 1024	Information Vector Length
q	8	Number of States
E	30	Number of Experiments

The computer simulation used two types of distributions to examine the performance of the proposed flash codes. The following are the sample frequency distributions:

1. Uniform Frequency Distribution – Each of the *k*-bit data has an equal chance of being selected. For instance, if the simulation uses 4-bit data, then each bit has a 1/*k* (25%) probability of being selected for bit update. Consequently, all the bits have approximately equal total number of bit writes at the point of block erasure,

2. Steady Dominated Distribution – In this type of distribution, a certain bit is given a higher probability for bit update, i.e., that bit dominates the write operation in the computer simulation. For simplicity, this study examines the performance of several flash codes using 50% and 70% steady dominated distributions only. It simply means that a bit is set to 0.5 or 0.7 probability for bit update (see Table 2 for a 4-bit data illustration), while the remaining probability is distributed uniformly to the remaining bits.

Table 2. Steady Dominated Distribution

4-bit Data	50% Dominated	70% Dominated
Bit 0	0.500	0.7
Bit 1	0.167	0.1
Bit 2	0.167	0.1
Bit 2	0.167	0.1

5. K-Partition Flash Code with BIFC-based Sharing and its Variants

This section describes the K-Partition Flash Code with BIFC-based Sharing and its variants, namely KPFC-s, KPFC-r and KPFC-m. The performances of the flash code and its variants are shown for both uniform and steady dominated distributions.

The basic K-Partition Flash Code (KPFC) partitions the block of *n* cells into *k* groups of contiguous cells, with each partition having exactly ⌊*n/k*⌋ cells. The *k* partitions are mapped to the *k* bits of data. When a bit update is performed on the *i*th bit, the level of charge of a cell within the corresponding *i*th partition is increased.

For the purpose of illustration, a sample write sequence is shown in Figure 2. In this example, the data vector *D* is (1,1,1,1,) after the fourth bit update; each of the partitions has one charge added. Note that a block erasure only occurred when the partition for bit 0 is already full and the current bit index to be updated is also 0 (refer to the 11^{th} bit update). Using computer simulation, the average case performance of KPFC was estimated, and a summary of the write deficiency across different *k* values is shown in Figure 3. Regardless of the availability of other partitions to accommodate cell write, a block erasure is still inevitable using this coding scheme since each of the *k* bits is only assigned to one and distinct partition. With this scenario, an improvement to KPFC is proposed using some sharing mechanism.

n=12 k=4 q=3

	Index *i* of updated bit d_i	Data Vector ***D***	Block Vector ***C***
0]		(0,0,0,0)	(0,0,0) , (0,0,0) , (0,0,0) , (0,0,0)
1]	3	(0,0,0,**1**)	(0,0,0) , (0,0,0) , (0,0,0) , (**1**,0,0)
2]	2	(0,0,**1**,1)	(0,0,0) , (0,0,0) , (**1**,0,0) , (1,0,0)
3]	1	(0,**1**,1,1)	(0,0,0) , (**1**,0,0) , (1,0,0) , (1,0,0)
4]	0	(**1**,1,1,1)	(**1**,0,0) , (1,0,0) , (1,0,0) , (1,0,0)
5]	0	(**0**,1,1,1)	(**2**,0,0) , (1,0,0) , (1,0,0) , (1,0,0)
6]	0	(**1**,1,1,1)	(2,**1**,0) , (1,0,0) , (1,0,0) , (1,0,0)
7]	0	(**0**,1,1,1)	(2,**2**,0) , (1,0,0) , (1,0,0) , (1,0,0)
8]	0	(**1**,1,1,1)	(2,2,**1**) , (1,0,0) , (1,0,0) , (1,0,0)
9]	0	(**0**,1,1,1)	(2,2,**2**) , (1,0,0) , (1,0,0) , (1,0,0)
10]	1	(0,**0**,1,1)	(2,2,2) , (**2**,0,0) , (1,0,0) , (1,0,0)
11]	0		***block erasure***

Figure 2. A Sample Write Sequence for KPFC

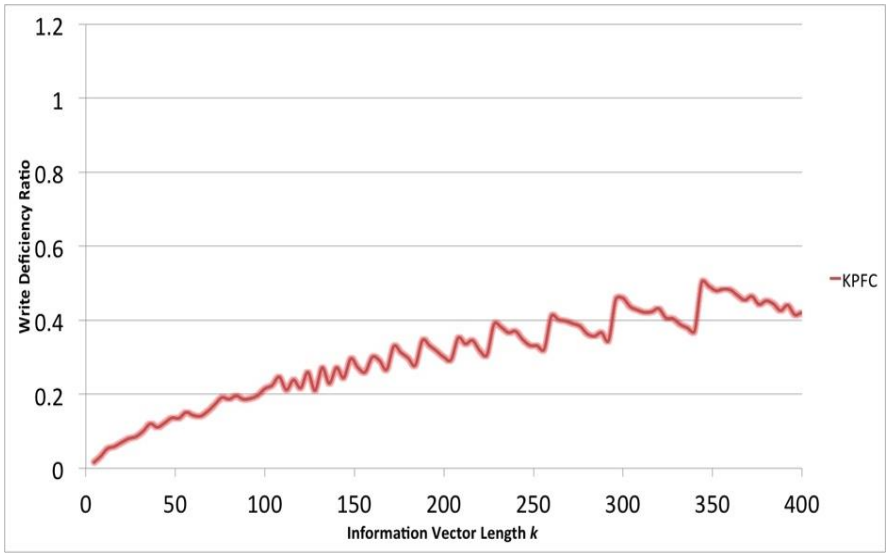

Figure 3. The Performance of KPFC using Computer Simulation Results. (Uniform Frequency Distribution, n=2048, q=8, $k \in$ {4, 8, … , 400})

5.1. KPFC-s

The first variant of KPFC with sharing mechanism is referred to as KPFC-s. In this flash code, the block is still divided into partitions to represent each of the k bits from the data vector. Technically, each partition also has $\lfloor n/k \rfloor$ cells. To avoid the scenario from KPFC that triggers the block erasure when a bit update is to be done on a full partition, a sharing mechanism is introduced. The technique utilizes a BIFC slice, which has a slice size of $s \geq \lfloor 1+\log_2(k+1) \rfloor$ (a detailed discussion on how each BIFC slice is initialized is available in [14]). Note that the BIFC slice must satisfy the constraint $s < \lfloor n/k \rfloor$ so that a BIFC slice can be integrated to a normal partition. Hence, instead of calling for a block erasure when a write operation is to be performed to an already full partition, the KPFC-s encoding looks for another partition where there is enough contiguous cells to accommodate a BIFC slice and perform the necessary bit update. See Fig. 4 for the illustration of a sample write sequence for KPFC-s.

To avoid ambiguity in decoding, a separator empty cell is maintained within a partition with a BIFC slice. A BIFC slice can be extended one cell at a time if there is a need for more cells, a scenario that occurs when there is a dominating bit index to be encoded. Extending the

BIFC slice is allowed if there are still available cells within the partition. Since a separator cell can also contribute to write deficiency, it is discarded when the BIFC slice needs to take over the whole partition assuming the original bit assigned to the partition has no bit update at all. Please refer to Figure 5 on how a BIFC slice is extended with KPFC-s. In effect, a block erasure occurs if an attempt to initialize a BIFC slice fails.

n=32 k=4 q=3

	Index *i* of updated bit d_i	Data Vector ***D***	Block Vector ***C***
0]		(0,0,0,0)	(0,0,0,0,0,0,0,0), (0,0,0,0,0,0,0,0), …)
1]	0	(0,0,0,0)	(**1**,0,0,0,0,0,0,0), (0,0,0,0,0,0,0,0), …)
2]	0	(1,0,0,0)	(**2**,0,0,0,0,0,0,0), (0,0,0,0,0,0,0,0), …)
	…		(2,**1**,0,0,0,0,0,0), (0,0,0,0,0,0,0,0), …)
16]	0	(0,0,0,0)	(2,2,2,2,2,2,2,**2**), (0,0,0,0,0,0,0,0), …)
17]	0	(1,0,0,0)	(2,2,2,2,2,2,2,2), (0,0,0,**0,0,0,1**), …)
18]	0	(0,0,0,0)	(2,2,2,2,2,2,2,2), (0,0,0,**0,0,0,2**), …)
19]	0	(1,0,0,0)	(2,2,2,2,2,2,2,2), (0,0,0,**0,0,1,2**), …)

BIFC slice initiliazed within the 2nd Partition

	Index *i* of updated bit d_i	Data Vector ***D***	Block Vector ***C***
20]	0	(0,0,0,0)	(2,2,2,2,2,2,2,2), (0,0,0,0,**0,0,2,2**), …)
21]	0	(1,0,0,0)	(2,2,2,2,2,2,2,2), (0,0,0,0,**0,1,2,2**), …)
22]	1	(1,1,0,0)	(2,2,2,2,2,2,2,2), (**1**,0,0,0,**0,1,2,2**), …)
23]	1	(1,0,0,0)	(2,2,2,2,2,2,2,2), (**2**,0,0,0,**0,1,2,2**), …)
24]	1	(1,1,0,0)	(2,2,2,2,2,2,2,2), (**2**,**1**,0,0,**0,1,2,2**), …)
25]	1	(1,0,0,0)	(2,2,2,2,2,2,2,2), (**2**,**2**,0,0,**0,1,2,2**), …)
26]	1	(1,1,0,0)	(2,2,2,2,2,2,2,2), (**2**,**2**,**1**,0,**0,1,2,2**), …)
27]	1	(1,0,0,0)	(2,2,2,2,2,2,2,2), (**2**,**2**,**2**,**0**,**0,1,2,2**), …)

2nd Partition is full with one empty cell as separator cell

Figure 4. A Sample Write Sequence for KPFC-s

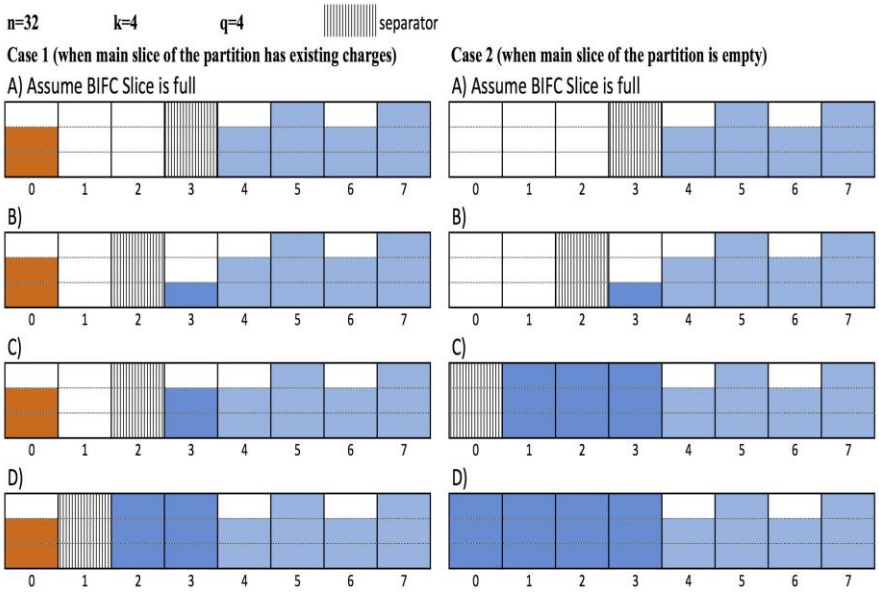

Figure 5. Extending a BIFC Slice in KPFC-s

The implementation of a sharing mechanism is only possible when $s < \lfloor n/k \rfloor$. However, if s exceeds the number of cells assigned to every partition, a BIFC slice cannot be made available within a normal partition. Thus, the flash code is reduced to the original KPFC. Using empirical results, the write deficiency of KPFC is slightly reduced using the sharing mechanism of KPFC-s. Note that depending on the number of cells of a block, denoted by n, and the length of the data vector, denoted by k, the expected improvement of KPFC-s is only

possible when $s < \lfloor n/k \rfloor$. When $s \geq \lfloor n/k \rfloor$, the performance of KPFC-s is exactly the same as that of KPFC (see Figure 6).

The present setup of KPFC-s assigns exactly $\lfloor n/k \rfloor$ for each partition and does not utilize the remainder (n mod k) cells. These remainder cells can be at most k-1. When k is large, the remainder cells can significantly affect the write deficiency of the flash code. Technically, KPFC-s just leaves these remainder cells untouched. It is with this context that KPFC-s can still be enhanced by applying some mechanism to utilize those unassigned cells to allow more cell writes and possibly reduce the write deficiency.

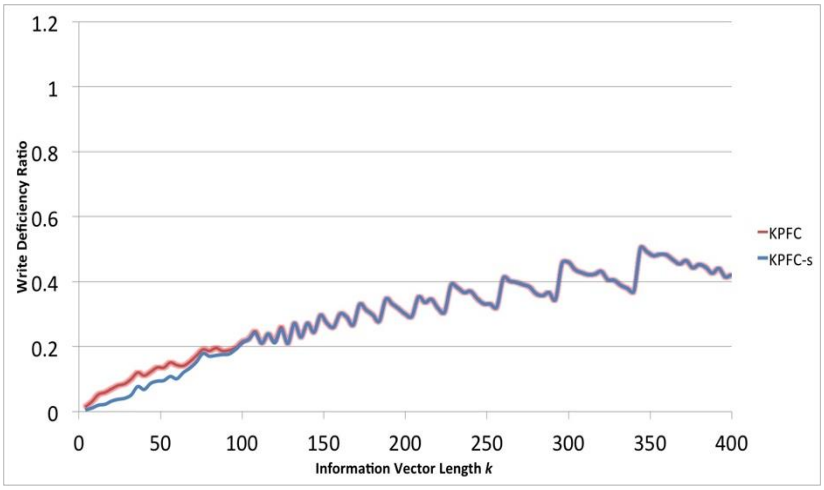

Figure 6. The Performance of KPFC and KPFC-s (Uniform Frequency Distribution, n=2048, q=8, $k \in$ {4, 8, …, 400})

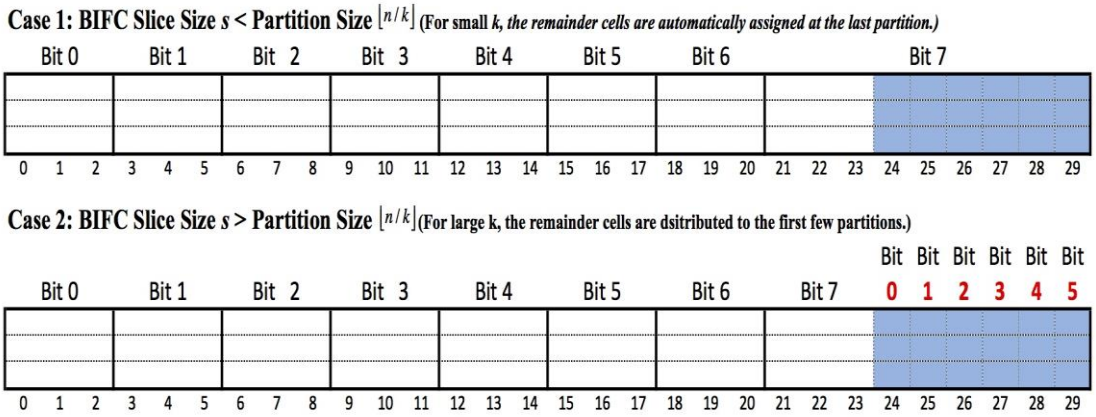

Figure 7. KPFC-r Slice Allocation

5.2. KPFC-r

KPFC-r is another variant of KPFC that introduces a method to utilize those unassigned cells not used by KPFC and KPFC-s. Instead of being left discarded, a new technique is introduced to KPFC so that these remainder cells can still be used. There are two scenarios

where KPFC-r allocates and utilizes the unassigned cells to the existing partitions of the block. When $s \leq \lfloor n/k \rfloor$, the flash code automatically assigns the cells not part of any partition to the last partition, i.e., partition k-1 has a bigger size compared to the other partitions. For simplicity of implementation, the last slice absorbs those extra cells so that a sharing BIFC slice can be initialized and extended if possible. If $r = n \bmod k$ refers to the number of unassigned cells, the size of slice k-1 in this scheme is $\lfloor n/k \rfloor + r$. For example, if n=30 and k=8, then r=6, therefore, slice k-1 will have a size of 9 cells while the rest will have a size of only 3 cells each (See Figure 7 Case 1). This strategy works well when k is not sufficiently large.

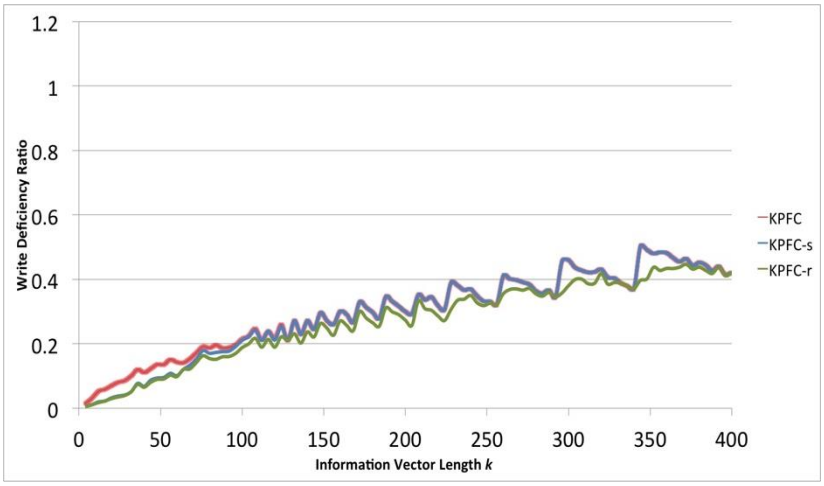

Figure 8. The Performance of KPFC, KPFC-s and KPFC-r (Uniform Frequency Distribution, n=2048, q=8, $k \in \{4, 8, \ldots, 400\}$)

The second scenario happens when $s > \lfloor n/k \rfloor$, in which case it is no longer possible to initialize a sharing BIFC slice. Instead of reverting to the original KPFC where a simple mapping is implemented, KPFC-r may still use up some of the unassigned cells. Because $r = n \bmod k$, then at most k-1 cells can be distributed and therefore at most k-1 partitions can have one added cell each (See Fig. 7 Case 2). On the average, half of the partitions can have an extra cell in this KPFC variant, thus potentially lowering the write deficiency further. The performance of the flash code is improved because of the technique where unassigned cells are still utilized as much as possible. Refer to Fig.8 for a comparison of simulation results for KPFC, KPFC-s and KPFC-r.

5.3. KPFC-m

The last variant of KPFC is called KPFC-m. The idea behind KPFC-m is to initialize as many BIFC slices as possible within a normal partition. Unlike KPFC-s that can only allocate a single BIFC slice within a partition, multiple BIFC slices can be accommodated in KPFC-m as long as there are available contiguous cells for sharing mechanism (see Fig.9). Similar to KPFC-r, the unassigned cells are also utilized in KPFC-m. However, the remainder cells are automatically assigned to the last partition. Hence, the last partition will have more cells compared to the rest of the partition. Contrary to KPFC-s and KPFC-r, extending the BIFC slice in KPFC-m is not possible. If a bit update is mapped to an already full BIFC slice, the

flash code looks for adjacent empty cells that can accommodate a BIFC slice. Block erasure is returned if search for vacant cells returns a null.

With the introduction of multiple BIFC slices within normal partition, it turns out that the flash code returned the best write deficiency among the variants of KPFC. With more BIFC slices made, more write operations were performed (see Table 3). Consequently, call for block erasures were delayed KPFC-m performs better when there is uniform frequency of distribution. As shown in Figure 10, KPFC-m outperforms KPFC, KPFC-s and KPFC-r. KPFC-m is even better than KPFC-s when $s \leq \lfloor n/k \rfloor$ (Note that KPFC-s has the ability to extend its BIFC slice by a single cell only). It can also be inferred from the graph that it has a superior mechanism to utilize the unassigned cells compared to KPFC-r, most especially for large values of k. KPFC-m significantly reduces the write deficiency of the flash code resulting to a better performance.

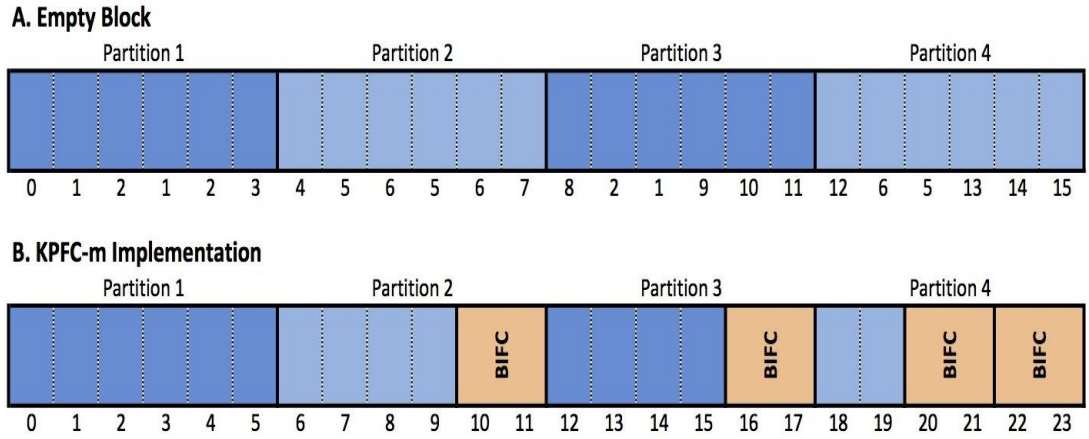

Figure 9. Initializing BIFC-slices in KPFC-m

Table 3. Comparison between KPFC-m and KPFC-r in terms of Average Write Operations

(n=2048, q=8, $k \in \{4, 8, \ldots, 1024\}$, E=30 experiments)

k	KPFC-r	KPFC-m	Additional Writes
4	14254	14287	33
204	10582	10649	68
404	8645	9505	860
604	6709	8753	2044
804	4991	7541	2550
1004	5581	6793	1212

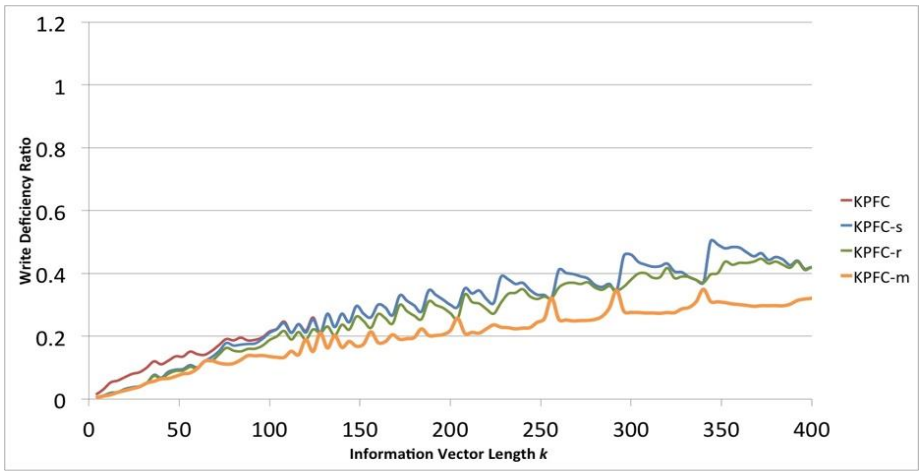

Figure 10. The performance of KPFC, KPFC-s, KPFC-r and KPFC-m (Uniform Frequency Distribution, n=2048, q=8, $k \in$ {4, 8, …, 400})

5.4. KPFC-r and KPFC-m in Steady Dominated Distributions

To examine further the performance of the proposed flash codes, the variants with better write deficiency in uniform distribution were tested on steady dominated distributions. As discussed in the previous section, a dominating bit index is designed in this type of distribution. The performances of the KPFC-r and KPFC-m in 50% and 70% steady dominated distributions were returned after running the simulation. Results are shown in Figure 11a and Figure 11b, respectively.

Given the graph of KPFC-r and KPFC-m, the results show that the latter is generally better than the former for both 50% and 70% steady dominated distributions. However, the write deficiency of both flash codes eventually increases as the value of k grows. Note that KPFC-r degenerates at k>200, because the write deficiency ratio shoots up close to 1 already. On the contrary, KPFC-m can still manage to lower it due to its inherent capability of utilizing the extra cells of the block. Initializing multiple BIFC slices potentially uses these unassigned cells.

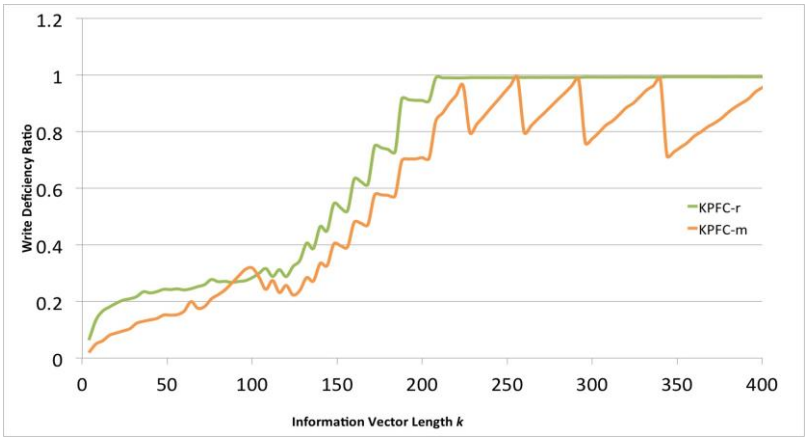

Figure 11.a KPFC-r and KPFC-m in 50% Steady Dominated Distribution (n=2048, q=8, $k \in$ {4, 8, … , 400})

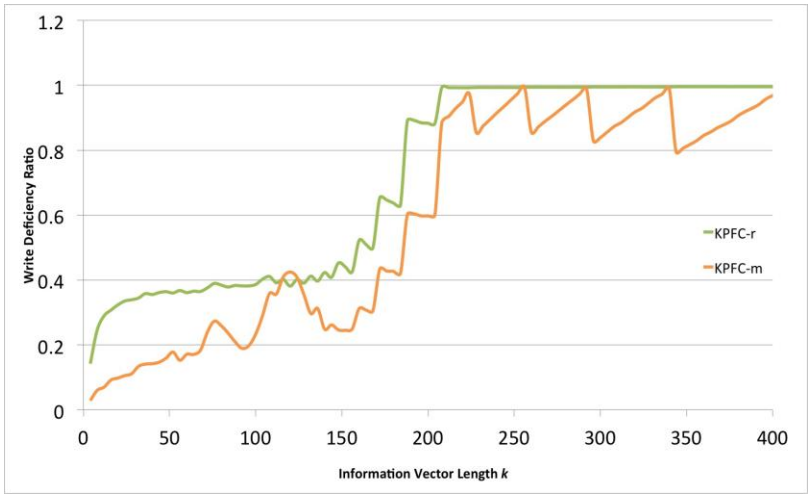

Figure 11. b KPFC-r and KPFC-m in 70%, Steady Dominated Distribution (n=2048, q=8, $k \in \{4, 8, \ldots, 400\}$)

6. Comparison of KPFC variants with existing Flash Codes

This section presents the derived average write deficiency of KPFC, KPFC-s, KPFC-r and KPFC-m. Moreover, the KPFC-r and KPFC-m were compared to the flash codes in literature.

Figure 12 shows the write deficiency ratios of KPFC, KPFC-s, KPFC-r and KPFC-m for larger range of values for k, unlike in the previous figures that were only up to k=400. As depicted in the graph, KPFC-s has a lower write deficiency for some lower values of k when compared to the original KPFC. Basically, this is due to the sharing mechanism employed in KPFC-s. Note that KPFC and KPFC-s converge starting at k when $s > \lfloor n/k \rfloor$. Nonetheless, another variant, KPFC-r, performs better than the two former flash codes for all instances of k. This is due to the utilization of the remainder cells in KPFC-r. In general, it is the KPFC-m that has the best performance among all the variants of KPFC. Apparently, KPFC-m returns the lowest write deficiency ratios for all values of k. In using multiple BIFC slices, KPFC-m allows more cell writes, and this results in a delay to the occurrence of block erasures.

KPFC-r and KPFC-m are then compared to the existing flash codes in literature. The performance of the flash codes as compared to slice based flash codes like ILIFC, LILIFC, BMFC and BMFC2 are shown in Figure 13, where both KPFC-r and KPFC-m showed a remarkable performance. When compared to more recent flash codes like BIFC and its improved variant BIFC-RCM, the proposed flash codes still returned the better average case performance (see Figure 14). KPFC-r and KPFC-m are then compared to high performing flash codes like SSFC and PFC. Simulation results reveal that the proposed flash codes still outperform the SSFC and PFC, as shown in Figure 15. With the introduction of sharing mechanisms, the write deficiency of the flash codes is significantly reduced, contributing to the better performance of the flash codes in the average case.

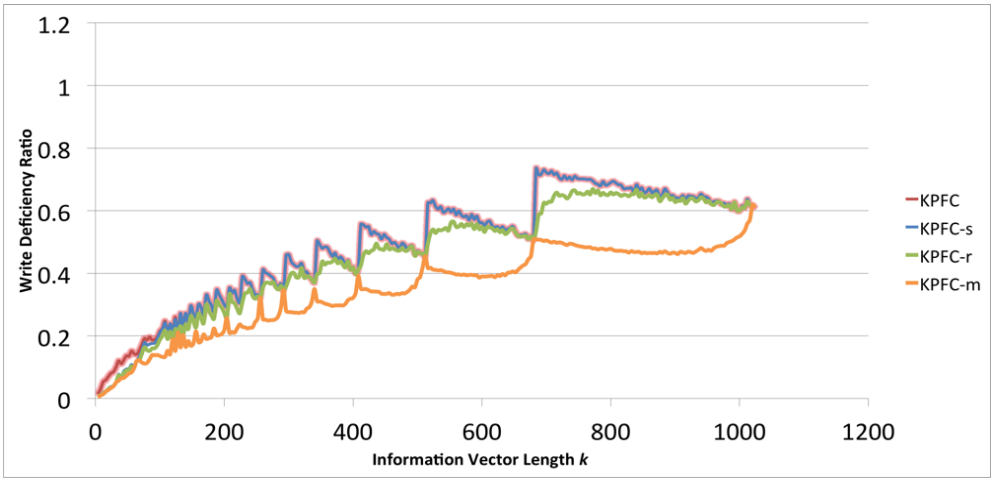

Figure 12. Comparing KPFC, KPFC-s, KPFC-r and KPFC-m (Uniform Frequency Distribution, n=2048, q=8, $k \in \{4, 8, \ldots, 1024\}$)

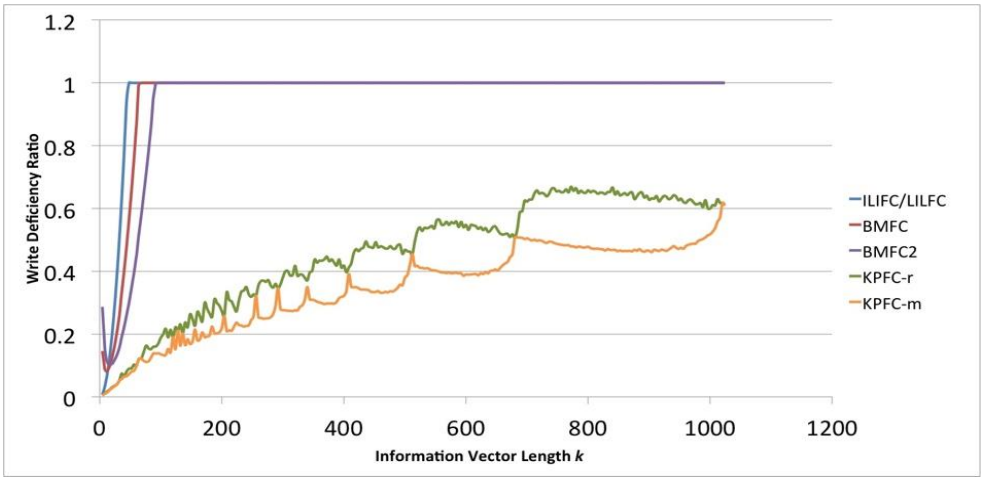

Figure 13. Comparing ILIFC/LILIFC, BMFC, BMFC2,KPFC-r and KPFC-m, (Uniform Frequency Distribution, n=2048, q=8, $k \in \{4, 8, \ldots, 1024\}$)

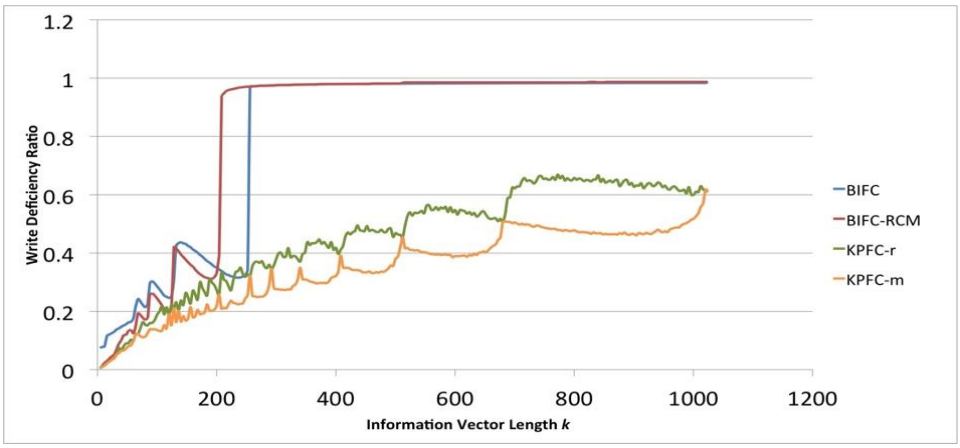

Figure 14. Comparing BIFC, BIFC-RCM, KPFC-r and KPFC-m (Uniform Frequency Distribution, n=2048, q=8, $k \in \{4, 8, \ldots, 1024\}$)

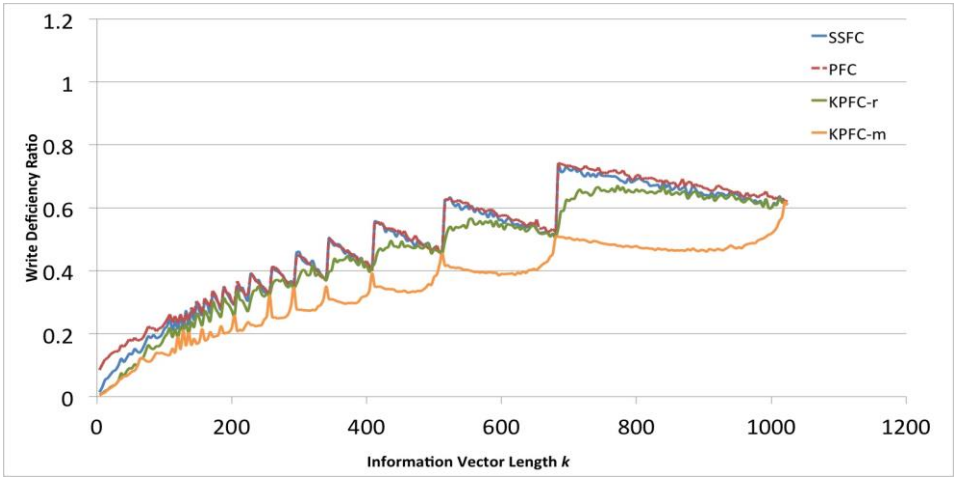

Figure 15. Comparing SSFC, PFC, KPFC-r and KPFC-m (Uniform Frequency Distribution, n=2048, q=8, $k \in \{4, 8, \ldots, 1024\}$)

7. Conclusion

The K-Partition Flash Code with sharing mechanism is presented in this study. There were three variants that were explored: KPFC-s, KPFC-r and KPFC-m. The average case performances of the flash codes were estimated through computer simulations. Overall, KPFC-r and KPFC-m returned the best performance. Empirical results showed that the aforementioned variants of KPFC are more efficient than the existing flash codes in literature. Further analysis shows that the write deficiency due to unassigned cells has a significant effect on the over-all performance of the flash codes. Hence, careful handling of these remainder cells is very valuable and essential in lowering the write deficiency of flash codes, as shown in KPFC-r and KPFC-m.

Further studies can be made to explore new methods of utilizing cells in a block to maximize cell writes. It is also interesting to conduct investigations on coming up with novel and promising ways of encoding such as new framework and hybrid flash codes, among others.

References

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory", in Proceedings of the IEEE, pages 489–502, 2003.
[2] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories", in Information Theory, 2007. ISIT 2007. IEEE International Symposium on, pages 1166–1170, June 2007.
[3] A. Jiang, V. Bohossian, and J. Bruck, "Rewriting codes for joint information storage in flash memories", Information Theory, IEEE Transactions on, 56(10):5300–5313, Oct 2010.
[4] A. Jiang and J. Bruck, "Joint coding for flash memory storage", in Information Theory, 2008. ISIT 2008. IEEE International Symposium on, pages 1741–1745, July 2008.
[5] Micron Technology Incorporated. Micron collaborates with sun microsystems to extend lifespan of flash-based storage, achieves one million write cycles, 2008. Accessed on 16 June 2014.
[6] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories", in Information Theory, 2008. ISIT 2008. IEEE International Symposium on, pages 1731–1735, July 2008.
[7] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, "Flash Memories", Kluwer Academic Pub., 1999.

[8] H. Mahdavifar, P.H. Siegel, A. Vardy, J.K. Wolf, and E. Yaakobi, "A nearly optimal construction of flash codes", in Information Theory, 2009. ISIT 2009. IEEE International Symposium on, pages 1239–1243, 2009.
[9] E. Yaakobi, A. Vardy, P.H. Siegel, and J.K. Wolf, "Multidimensional flash codes", in Communication, Control, and Computing, 2008 46th Annual Allerton Conference on, pages 392–399, Sept 2008.
[10] A. Jiang, "On the generalization of error-correcting wom codes", in Information Theory, 2007. ISIT 2007. IEEE International Symposium on, pages 1391–1395, June 2007.
[11] R. Suzuki and T. Wadayama. "Layered index-less indexed flash codes for improving average performance", in Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on, 2011, pp. 2138-2142
[12] H. Esling, R.R. Ortiz and P. Fernandez, "Bi-Modal Flash Code using Index-less Indexed Flash Code and Layered Index-less Indexed Flash Code", Advanced Science and Technology Letters (Software 2013), vol. 35, pp.19-22.
[13] R.R.L. Ortiz, H. Esling, and P. Fernandez, "Combining flash codes for write deficiency reduction", International Journal on Software Engineering and Its Applications, 8(4):47–60, April 2014.
[14] M.J. Tan and Y. Kaji, "Uniform compartment flash code and binary-indexed flash code", Technical Report of IEICE, pages 25–30, 2012.
[15] M.J. Tan and Y. Kaji, "Flash code utilizing resizable-clusters", in IEEE International Conference on Electro Information Technology, number 2013 IEEE International Conference on Electro/Information Technology, EIT 2013, Graduate School of Information Science, Nara Institute of Science and Technology, 2013.
[16] M.J. Tan and Y. Kaji, "Flash code utilizing binary-indexed slice encoding and resizable-clusters", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E96-A(12):2360–2367, 2013.
[17] M.J. Tan, P. Fernandez, N.A. Salazar, J. Ty, and Y. Kaji, "Flash code with dual modes of encoding", in Proceedings of the 2013 Workshop on Computation: Theory and Practice, 2013.
[18] M.J. Tan, P. Fernandez, N. Salazar, J. Ty, and Y. Kaji, "Multi-mode encoding with binary-index flash code", Technical Report of IEICE, 2012.
[19] G.N. Corneby, L.K. Sanchez, M.J. Tan, P. Fernandez, and Y. Kaji, "Phoenix flash code: Introducing the absorption and revival operations for reducing flash memory write deficiency", in Proceedings of the 11th National Conference on Information Technology Education, 2013.

Authors

Riz Rupert L. Ortz, is a Ph.D. in Computer Science student at the Ateneo de Manila University. He is currently connected to the College of Management and Information Technology (CMIT) at Northwest Samar State University (NwSSU), Calbayog City, Samar, Philippines.

Proceso L. Fernandez, Jr., Ph.D., is an Associate Professor at the Ateneo de Manila University.