Innovations

Implementing K-Partition Flash Code in Simultaneous Bit Update Framework

Riz Rupert L. Ortiz, Ph.D.
College of Computing and Information Sciences,
Northwest Samar State University, Calbayog City, Philippines
Corresponding Email: riz.rupert.ortiz@nwssu.edu.ph
Received: 28 May 2022 Accepted: 30 June 2022 Published: 30 June 2022

Abstract

A flash code is a mechanism basically used in encoding and decoding digital information to flash memory devices. Flash codes in literature operates in single bit update framework where a data update equates to a single cell write to the flash memory block. In this study, the K-Partition Flash Code (KPFC) and its variant KPFC-m is implemented in the new framework where multiple bit update is possible. Analytic investigation was conducted to derive the theoretical worst case write deficiency while computer simulations were used to estimate its average case performance. Results show that KPFC and its variant KPFC-m is still competitive with some flash codes in literature using the new framework. More importantly, the implementation of this coding scheme can help extend the lifespan of flash devices.

Keywords : *1.Flash Code, 2.Flash Memory, 3.Simultaneous Bit Update, 4.Write Deficiency*

I. Introduction

Flash memory is classified as a semi-conductor storage device that can be found in variety of applications. It is a type non-volatile memory (NVM) generally used in mass storage devices nowadays. Its application can be significantly found in embedded systems like in household appliances; it is also progressively used in telecommunication devices and other machineries. The technology in flash memories guarantees a high-density tolerance making it quick, compact, reliable and shock-resistant. Furthermore, it is relatively easy to electrically program and erase flash memory devices (Richter 2014; Bez et al. 2003).

The layered structure of flash memories is logically organized into blocks composed of thousands of cells that can store electric charges. These array of floating gate cells can be grouped as partitions or slices inside a block. A cell charge can transition to a higher level through *Channel hot electron injection* (Cappelletti &Modelli 1999; Rivest & Shamir 1982). The process is basically called *cell programming* (Jiang & Bruck, 2008), where the electric charges of each cell are determined by some encoding function defined by a coding mechanism. The coding technique is commonly denoted as *flash codes* (Jiang, Bohossian& Bruck 2007; Yaakobi et al. 2008).

Flash memory is modelled as a write-asymmetric memory (Jiang, Bohossian& Bruck 2007). Decreasing the level of charge in a cell is not possible unless *block erasure* is performed that physically erases all eletric charges of

cells in a block (Chee et al. 2017). *Fowler-Nordheim tunneling mechanism* is applied to the whole block (Canet et al. 2005), where all cells are set back to 0. The whole process is computationally costly and must be avoided because flash memory devices are constrained by some limited number of block erase cycles. Moreover, memory cells eventually wear out once a practical limit is operationally reached. Block erasure significantly affects the lifespan and reliability of flash memory devices. A typical memory block can only tolerate approximately 10^3 - 10^5 number of block erasures before a memory block becomes unreliable for storing data (Mahdavifar et al. 2009; Yaakobi et al. 2015).

To address the problem on limited number of block erasure, an efficient flash code can be carefully and intelligently designed so that more write operations can be accommodated before calling block erasure. Assuming no errors will occur, delaying the call for block erase will lengthen the lifecycle of flash memories. Hence, the underlying efficient design of coding mechanism of flash code can potentially affect the speed and lifespan of flash memories.

Most flash codes in literature only allow single bit update framework per cell write. More recently, there are already developed flash codes that accommodate multiple bit updates. The new platform which was advanced by Bautista and Fernandez (2014) is called Simultaneous Bit Update framework; it allows multiple bits of data updates using a single cell write. Some flash codes in literature in both frameworks will be described and explained in greater detail in the next section.

The K-Partition Flash Code (KPFC) is a recent flash code in literature that is shown to be efficient in maximizing cell updates before calling block erasure (Ortiz and Fernandez, 2014). Most flash codes in literature only allow single bit update framework per cell write. Nonetheless, this study will implement KPFC and its variant KPFC-m in a new framework advanced by Bautista and Fernandez (2014). This framework is called Simultaneous Bit Update framework; it allows multiple bit of data updates using a single cell write.

II. Objectives

This study focused on the performance of K-Partition Flash Code and its variant KPFC-m in the Simultaneous Bit Update Framework. The study sought answers to the following additional objectives:

1. To assess the average-case performance of the flash codes through computer simulation in terms of:
    1.1 write deficiency; and
    1.2 average number of data updates.
2. To compare the performance of the flash codes against other flash codes.

III. Context of the Study

The logical structure of flash memory is logically prearranged as blocks. A block is comprised of a large number of flash cells represented by n. Typical values of n range from 2^{18} to 2^{20} floating gate cells (Mahdavifar et al. 2009).

In multilevel flash memories, a block is abstractly denoted by a vector $C = (c_0, c_1,..., c_{n-1})$, where $c_i \in \{0,1,..., q-1\}$. Each cell transitions in one of q levels from a finite set $A_q = \{0,1,..., q-1\}$. The parameter q spans from 2 to 256 (Mahdavifar et al. 2009). The values indicate the amounts of electric charges present in the cell array. Accordingly, a cell is *empty* when it has a charge of 0, while a cell is *full* when it has a charge of q-1. An *active* cell is in effect when it is neither empty nor full.

On the other hand, the information stored in the memory block is a binary k-bit data $D = (d_0, d_1, ..., d_{k-1})$, where $d_i \in \{0,1\}$ and $k<n$. The size of a memory block can be 64, 128, or 256 kilobytes of data (Jiang, Bohossian and Bruck 2010). The mapping between the block state vector C and the contained digital data D is handled by some coding mechanism called flash code F.

Formally, the flash code $F = (E, D)$ is a coding mechanism composed of two central functions. The encoding function $E\ (i,C) : \{0,1, \ldots, k\} \times \{0,1,\ldots, q-1\}^n \rightarrow \{0,1,\ldots, q-1\}^n \cup \varepsilon$ specifies the procedures on creating a new state to the block with the index i of the data bit $d_i$which requires to be written and ultimately call for update on the current block state C. In contrast, the decoding function $D\ (C) : \{0,1, \ldots, q\text{-}1\}^n \rightarrow \{0,1\}^k$ translates the information state C stored in the block into its equivalent k-bit data D (Jiang, Bohossian& Bruck 2007).

There are only two operations to update a block's cell state vector. The first is cell programming, that increments the level of a cell by at least one until it reaches q - 1. Basically, the cell state vectors transition from a lower state to a higher state. Suppose there are two cell state vectors $C = (c_0, c_1,\ldots, c_{n-1})$, and $C' = (c'_0, c'_1,\ldots, c'_{n-1})$. If $c'_i \geq c_i$ holds true for all $i \in \{0,1,\ldots, n\text{-}1\}$, then C' is higher than C, and say that C' is strictly above C if $C' \neq C$ is further satisfied. The other block operation is the block erasure which resets the block's cell state vectors values to zero and reducess it to an empty state $\{0\}^n$.

For purposes of evaluating the performance of the flash code, metrics like write-deficiency, average number of data updates and number of auxiliary writes may be used.

IV. Literature Review

This section provides a short summary of the development of flash codes. Discussions on flash memory, its technology and architecture and flash codes are presented in this section. It is notable to mention that there are studies that stressed the significance of improving the worst case write deficiency of the flash codes. Further, there are also studies that focused on reducing the write deficiency ratio of the flash codes in the average case which is conceivably an approximate to real world applications.

Flash Memory

Flash memories are regarded as the most attractive kind of non-volatile memory nowadays. With their promising features and qualities like speed, high-density, robustness, and reliability, flash memories are now widely used in memory cards, mobile devices, embedded systems and other standard storage devices. The flash memory technology is based on floating-gate transistors having storage cells that allow a transition from a lower state to a higher state but not vice versa. This is similar to punch cards and digital optical discs technologies that cell transition is irreversible between states.

Technology and Architecture

From the technology perspective, there are two types of flash memories, to wit: NOR type of flash memory and NAND type of flash memory. Similarly, both technologies are based on floating-gate transistors. However, the NAND technology can store more data because of its denser layout compared to NOR. Hence, it is more commonly used in designing mass storage devices.

The NAND technology flash memory is further categorized into two: single-level cell (SLC) and multi-level cell (MLC). Normally, SLC flash is used for industrial products while MLC flash is intended for consumer products. MLC flash memory allows to store multiple bits; on the other hand, SLC flash memory stores a single bit of information per cell. Multi-level flash memory was developed recently to attain higher density (Lee et al. 2009).

Flash Codes

When multilevel flash devices came to be widely used, generalization of floating codes rose to be of equal importance. While there are several studies on floating codes, not much attention was given to the development of

coding mechanisms to optimize the number of rewrites prior to the calling block erasure. Delaying the call for block erase significantly enhances writing speed and potentially extends the lifespan of flash memory.

Coding scheme for write asymmetric memories (WAM) has roots to the older write once memory codes (WOM) codes, basically called floating codes where each cell can either be in two states. Flash code or floating code was coined in 2007 (Jiang, Bohossian and Bruck 2007). A floating code was introduced for $k = 2$, $n \geq 3$ and arbitrary q. In the subsequent study, Jiang and Bruck (2008) presented floating codes for limited values of k and n, like $n = k \geq 3$ and $3 \leq k \leq 6$. They also described an indexed code that stores bits into groups of cells using some indexing scheme. Similarly, the study of Yaakobi et al. (2008), introduced a code for arbitrary n, q, and k; the authors were able to improve the coding schemes of Jiang, Bohossian, and Bruck (2007).

The Index-less Indexed Flash Code (ILIFC) is among the most popular flash codes in literature. The ILIFC basically divides the block into logical units called *slices* with k cells. Consequently, there are approximately m slices a memory block where $m = \lfloor n / k \rfloor$. ILIFC offers a well-designed mechanism for storing the bit index i as well as the bit value v_i of the active slices. We denote the vector $C = (c_0,c_1,...,c_{n-1})$ loosely as $C = (s_0|s_1|...|s_{m-1})$ to illustrate the division of the block into equally sized slices. Activation of a slice is achieved by performing cell write in the i-th cell within a slice that matches the i-th bit index of the k-bit data (Yamawaki, Kamabe& Lu 2017).

As to its performance, its worst case write deficiency is $O(k^2q)$. The study of Mahdavifar et al. (2009) provides detailed information of ILIFC and its decoding and encoding functions. Block erasure only happens when a write operation is not accommodated from actives slices. This type of encoding with k cells in every slice performs poorly k increases. The remainder cells, ie unused cells which can be at most n mod k also contributes to the write deficiency.

The Layered Index-less Indexed Flash Code (LILIFC) is an adaptation of ILIFC. Technically, LILIFC operates similarly with ILIFC. This coding technique also operates on equally sized sub-blocks called *slices* with k cells each. However, the only difference is the layer-based encoding where the level of each cell in a slice is increased in a circular manner. The study of Suzuki and Wadayama (2011) presents a more detailed discussion on how LILIFC manages the encoding and decoding of k-bit data to the memory block. With regards to performance, LILIFC returns a better performance than ILIFC in the average case. Nevertheless, both coding techniques have the same worst case write deficiency

A hybrid flash coded using both techniques, ie ILIFC, and LILIFC was studied by Ortiz, Esling and Fernandez (2014). It was referred to as the Bi-Modal Flash Code (BMFC). As opposed to the former flash codes, BMFC reduced the size of every slice to $k/2$, with the first half indeces using ILIFC and the remaining half using LILIFC. It treated the block into two-sided storage growing towards opposite ends whenever slice is activated. To do this, an empty slice is maintained and preserved to distinguish left and right oriented slices. This slice consequently added to the write deficiency of the flash code. Its asymptotic write deficiency is $O(k^2q + n/k)$.

On the other hand, the study of Tan and Kaji (2012) introduced the Binary Indexed Flash Code (BIFC). This flash code likewise used fewer cells for each slice. Contrary to ILIFC and LILIFC, the slice in BIFC has typically a size of $s = O(\log k)$ and $s \geq \lfloor 1+\log_2(k+1) \rfloor$. BIFC performed better than the former flash codes when there is larger value of k. However, this flash code suffers an overhead deficiency of s-2 over each of slices in the block due to inherent indexing mode. The BIFC has a worst case write deficiency of $O(qk\log k + n)$.

The write deficiency is the metric used to describe their performances. Using computer simulations, the ILIFC and LILIFC have better performances in the lower values of k. As the value of k increases, its write deficiency shoots up. BMFC was able to improve the performance of ILIFC and LILIFC to some values of k. From the given flash codes, it is the BIFC that has the better performance. Its ability to use smaller size of slices translated to better performance. It significantly outperformed the other flash codes using their average case performances.

The study of Bautista and Fernandez (2014) first introduced the multi-bit update framework where simultaneous data bit updates are possible in a single data update. Essentially, the performance of the flash code is the sum of all unused levels and auxiliary writes. The authors also presented a novel coding mechanism referred to as

Sequential Cascading Flash Code (SCFC) that requires k active cells to represent the data vector of length k. Block erasure is returned if there are no available k cells to encode k-bit data. In SCFC, the value of cell $c_j$represents a single bit v_i. Thus no logical indexing is required. The bits are assigned from left to right ignoring full cells. When current cell is full, there is a cascade shift performed to assign the bit to the next cell. In this encoding, no residual cells are produced.

Another flash code that operates in multi-bit update framework is the Circular Pair Flash Code (CPFC) (Agustin & Fernandez 2015). In this scheme, the concept of pair slices is utilized where it has the capability to program a single cell for two updates. With this flash code, a slice can be one of two types: single slice or pair slice. The pairing of bits is done in circular fashion. However, applying all possible pairs to code is impractical because the slice size will definitely be affected. The process enables the flash code to handle more data updates before resorting to block erasure. The performance of the flash code has been described using three metrics: namely, write deficiency, write efficiency and average number of data updates. Results of the study showed that CPFC could handle more data updates before calling block erasure. This was attributed to the utilization of pair slices accommodating more bit updates against existing codes.

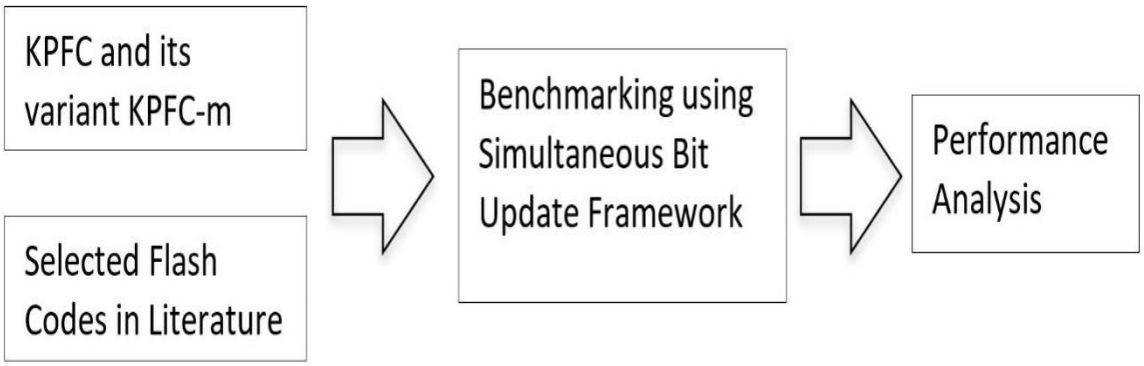
As to the write deficiency ratios of SCFC and CPFC in the uniform, the SCFC returned the better performance than CPFC in the multiple bit update framework. The sequential cascading technique where there is no indexing required was able to utilize most of the cells in the block. Hence, the flash code was able to maximize cell writes in the block and further delayed the call for block erase.

V. Methodology

The flash codes were tested to ascertain the accuracy of the implementation. Selected flash codes from literature were also included in the study. Using the new framework, the flash codes involved were implemented to produce benchmarking results as to its performance. Using computer simulations, the write efficiency ratios, write deficiency ratios, and number of data updates were returned for comparison.

In this study, the algorithms were theoretically analyzed as to its worst case performance. For the empirical results, computer simulations implemented in Java were used to describe the average case performance of the flash codes. The computer simulations were set to run in 30 experiments with parameters of n=2048 and q=8 for various k values from 4 to $n/2$, in increments of 4. The update probability of the information vector p has values of 10% to 90%.

Figure 1: The Methodology of the Study



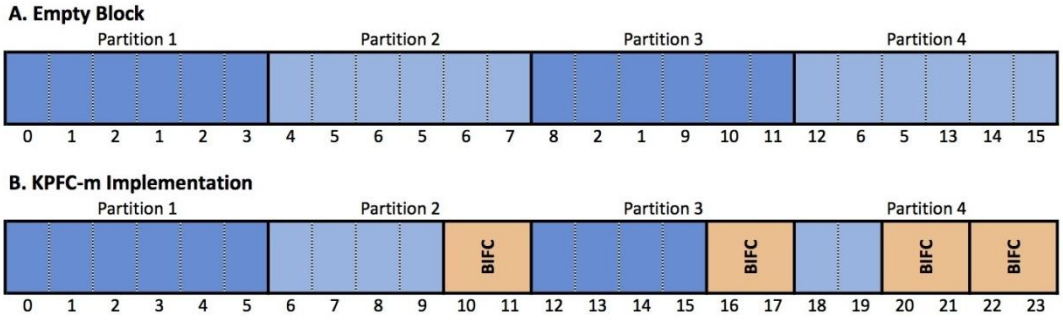
VI. KPFC and its variant KPFC-m in the Simultaneous Bit Update Framework

The K-Partition Flash Code and its variant, KPFC-m are discussed in detail in this section. Refer to the study of Ortiz and Fernandez (2014) on operation of the flash codes and their performances in single bit update framework.

The K-Partition Flash Code (KPFC) basically partitions the block of n cells into k groups of contiguous cells mapped bijectively to the k indices of the data. Each group is referred to as a partition having exactly $\lfloor n/k \rfloor$ cells. Updating a bit is relatively simple, Figure 2 illustrates how a cell write is implemented to a cell within the corresponding ith partition that corresponds to the index of the bit update.

n=12 **k=4** **q=3**

	Index i of updated bit d_i	Data Vector $\boldsymbol{D}$	Block Vector $\boldsymbol{C}$
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***



A variant of KPFC is called KPFC-m. It offers a sharing mechanism using a BIFC slice. The BIFC slice has a size of $s \geq \lfloor 1+\log_2(k+1) \rfloor$. The introduction of sharing mechanism delays the occurrence of block erasure. The idea behind KPFC-m is to initialize as many BIFC slices as possible within a normal partition (see Fig. 3 for the sharing mechanism). KPFC-m works well when there are many unassigned cells, these remainder cells are automatically assigned to the last partition where BIFC slices can be implemented.

The following figures shows the decoding and encoding algorithms of the K-Partition Flash Code (see Fig. 4 and 5). On the other hand, Figures 6 and 7 illustrate the decoding and encoding algorithms of KPFC-m, respectively.

Data: Block vector $C = (c_0, c_1, \ldots, c_{n-1})$
Result: Data Vector $D = (d_0, d_1, \ldots, d_{k-1})$

1 $h \leftarrow \lfloor n/k \rfloor$
2 **for** $i \leftarrow 0$ **to** $k-1$ **do**
3 $\quad d_i \leftarrow (\sum_{j=0}^{h-1} c_{ih+j}) \bmod 2$
4 **end**
5 **return** $D = (d_0, d_1, \ldots, d_{k-1})$

Figure 4: Decoding Algorithm of KPF

Data: Block vector $C = (c_0, c_1, \ldots, c_{n-1})$, partitioned into k groups of $\lfloor n/k \rfloor$ cells each, i.e. $C = (p_0|p_1|\ldots|p_{k-1})$ and the index i
Result: Block vector $C' = (c'_0, c'_1, \ldots, c'_{n-1})$ or the erase symbol $\mathcal{E}$

1 $Y = (y_0|y_1|\ldots|y_{k-1}) \leftarrow (p_0|p_1|\ldots|p_{k-1})$
2 **if** `active` (p_i) **then**
3 $\quad$ `write` (y_i)
4 $\quad$ **return** Y
5 **else**
6 $\quad$ **return** $\mathcal{E}$
7 **end**

Figure 5: Encoding Algorithm of KPFC

Data: Block Vector $C = (c_0, c_1, \ldots, c_{n-1})$
Result: Data Vector $D = (d_0, d_1, \ldots, d_{k-1})$

```
h ← ⌊n/k⌋
s ← sizeofBIFCslice()
for i ← 0 to k − 1 do
    t ← numofBIFCslices(i)
    d_i ← (Σ_{j=0}^{(h−1−st)} c_{ih+j}) mod 2
end
for i ← 0 to k − 1 do
    t ← numofBIFCslices(i)
    if t ≠ 0 then
        for u ← 0 to t − 1 do
            S_u ← u^th BIFC Slice of the ith partition
            if isActiveBIFC (S_u) then
                j ← indexofBIFC(S_u)
                d_j ← (d_j + weightofBIFC(S_u)) mod 2
            end
        end
    end
end
return D = (d_0, d_1, ..., d_{k−1})
```

Figure 6: Decoding Algorithm for KPFC-m

Data: Block vector $C = (c_0, c_1, \ldots, c_{n-1})$, partitioned into k groups of $\lfloor n/k \rfloor$ cells each, i.e. $C = (p_0|p_1|\ldots|p_{k-1})$ and the index i
Result: Block vector $C' = (c'_0, c'_1, \ldots, c'_{n-1})$ or the erase symbol $\mathcal{E}$

```
Y ← (y_0|y_1|...|y_{k−1}) ← (p_0|p_1|...|p_{k−1})
if active(p_i) then
    write(y_i)
else
    S_j ← active BIFC Slice of the ith bit
    if (S_j = null) then
        S_j ← new BIFC Slice of the ith bit
        if (S_j = null) then
            return E
        end
    end
    write2 (S_j)
end
return Y
```

Figure 7: Encoding Algorithm for KPFC-m

Average Case Performance

Computer simulations were performed to estimate the average case performance of the flash codes involved in the study. The implemented simulation had the following parameters: n=2048, q=8, $k \in \{4,8,...,1024\}$. The k-bit binary sequence D' was randomly generated, where each bit (d'_i) is given the same probability of being updated. The flash codes were tested using update probability values from p =0.10 to p=0.90 with increments of 0.20. There were 30 experiments for each value of $k \in \{4,8,...,1024\}$. Similarly, the same configurations were used in comparing the flash codes involved with other flash codes in literature. For each experiment, the write deficiency and the average number of data updates were returned by the simulation.

The write deficiency ratios of KPFC and KPFC-m are presented in Figures 8 and 9. It can be observed that as the update probability p increases, the write deficiency ratios of both flash codes decreases. This can be attributed to the equal probability of every bit in the conduct of simulation; there is uniform frequency distribution.

Parallel to single bit update framework, the performance of the flash codes decreases when the size of the data k increases. Note that between KPFC and KPFC-m, it is the latter that has the better write deficiency for all update probability values of p. This is due to the ability of KPFC-m to utilize the residual cells, which can be at most $n \bmod k$. These remainder cells cause the spikes in the trajectory of the write deficiency ratios of the flash codes. Yet, the KPFC-m variant can significantly use these cells at some lower values of update probability p.

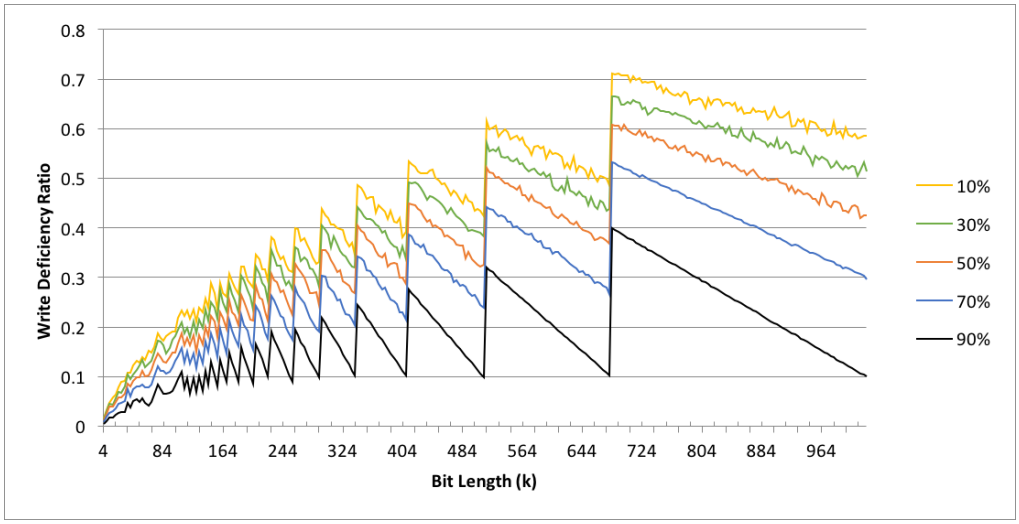


Fig. 8: Worst Case Write Deficiency of KPFC

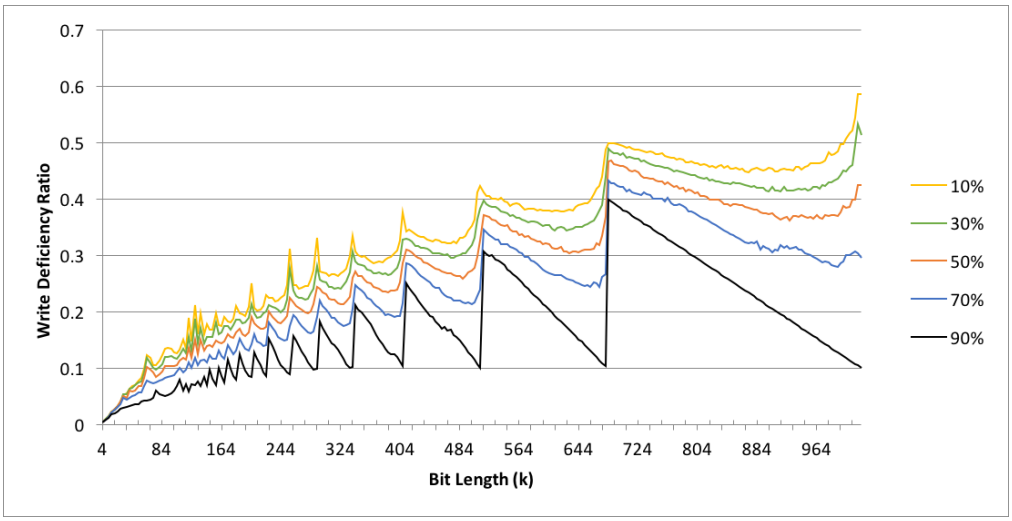


Fig. 9: Worst Case Write Deficiency of KPFC-m

Figures 10 and 11 present the average number of data updates for the flash codes in the simultaneous bit update framework. As shown in the graphs, while the size of the data k increases, the number of data update decreases. The larger the size of the data, the more difficult for the flash code to accommodate cell writes. Thus, this phenomenon results to the calling of block erasure. Moreover, the number of data updates gradually decreases as the update probability p increases. The same scenario can be observed in both flash codes, the KPFC and its variant KPFC-m. It can be inferred that in simultaneous bit update framework, multiple bits can be updated at a single time. Hence, the total number of data writes is expected to decrease considering that more cell writes are performed for every data update.

However, looking closely at the data, KPFC-m has more data updates compared to its predecessor, the KPFC. In all update probability, the flash code showed better performance than KPFC. This implies that the variant KPFC-m can still work well in the new framework.

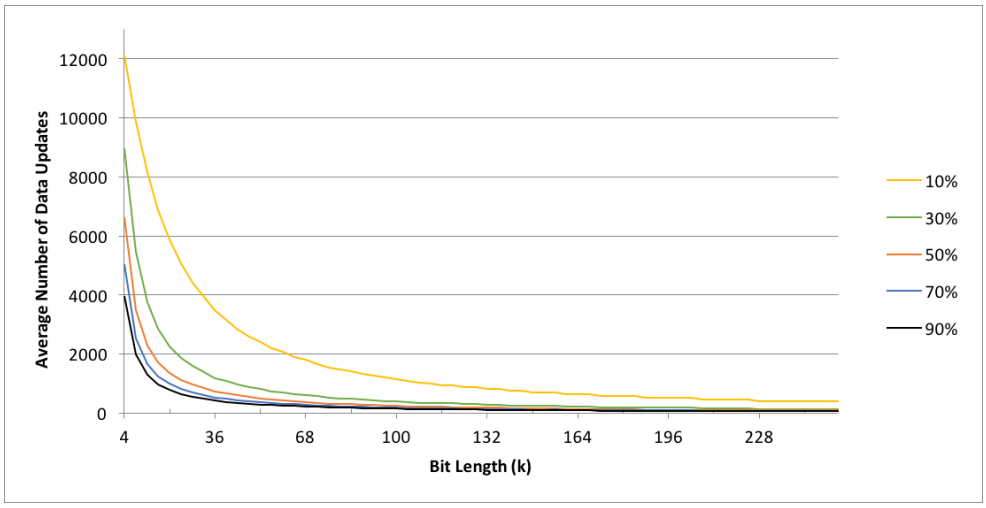


Fig. 10: Average Number of Data Updates of KPFC

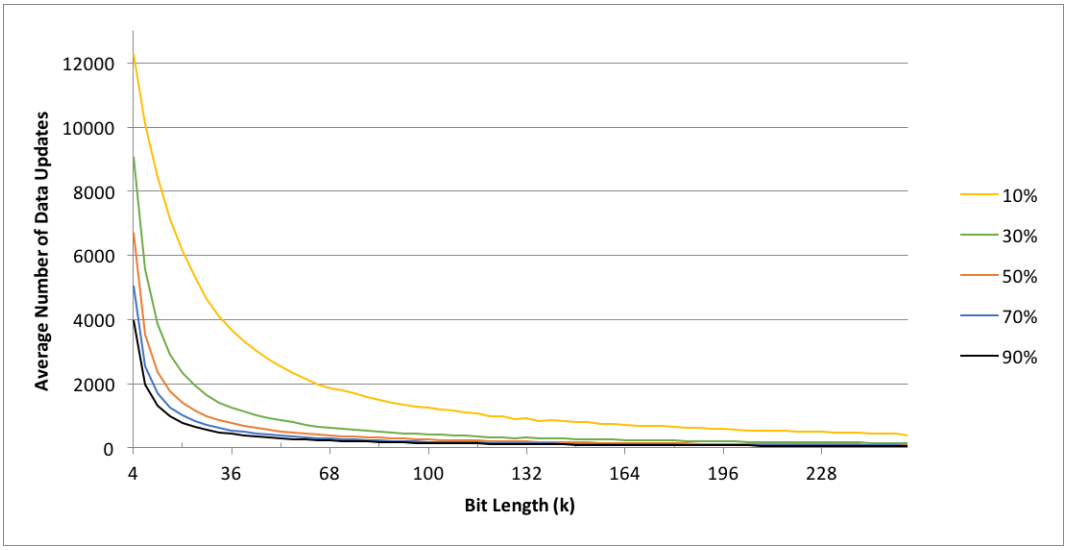


Fig. 11: Average Number of Data Updates of KPFC-m

Comparison to other Flash Codes in Literature

For the purpose of comparison, the variant KPFC-m is used for simplicity. The performance of KPFC-m is then compared to the slice based flash codes like ILIFC and LILIFC, as well as the high performing flash codes designed in the simultaneous update framework like SCFC and CPFC (see Figures 12 and 13for the comparison).

The write deficiency ratios of KPFC-m, ILIFC, LILIFC, CPFC and SCFC are shown in Figure 12. In general, the variant KPFC-m outperforms ILIFC, LILIFC and even CPFC in terms of write deficiency. Nevertheless, it is SCFC that performs better than KPFC-m. The ability of SCFC to avoid residual cells results to its better performance when dealing with more cell writes. As to number of data updates (see Fig. 13), It is the CPFC that returned more data updates. Its concept of pair slices where a single cell is capable for two updates is the key to handle more data updates. KPFC-m and SCFC are comparable in terms of data updates. From the results, it can be shown that KPFC and its variant KPFC-m is still competitive in the new framework.

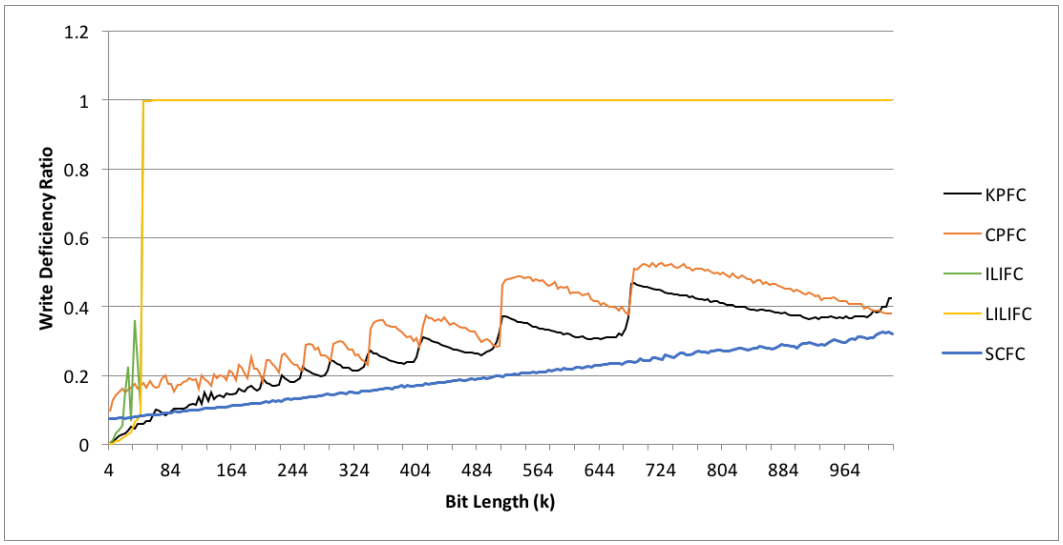


Fig. 12: Worst Case Write Deficiency of KPFC-m, CPFC, ILIFC, LILIFC, SCFC

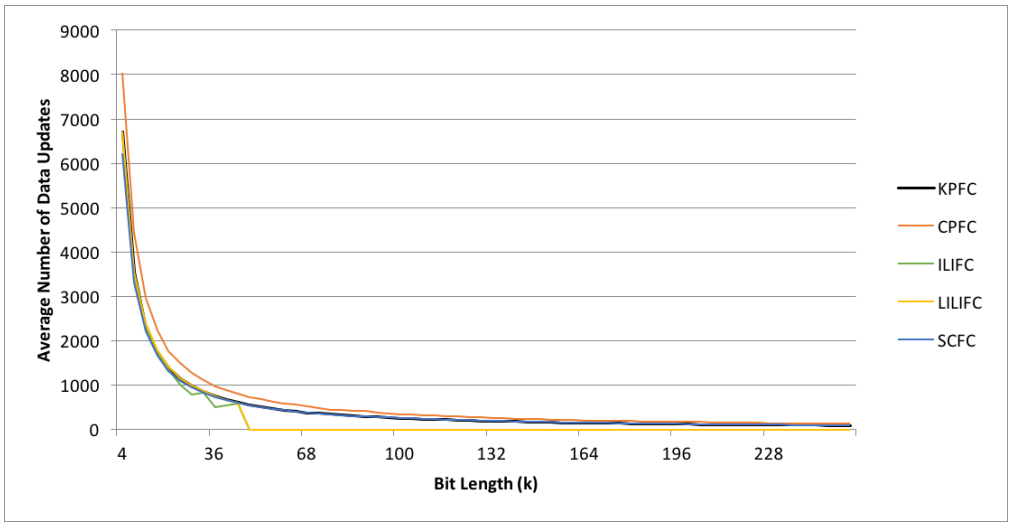


Fig. 13: Average Number of Data Updates of KPFC-m, CPFC, ILIFC, LILIFC, SCFC

VII. Conclusion

In this study, the KPFC and its variant KPFC-m is implemented in the new framework where multiple bit update is possible. The theoretical worst case of the flash codes was analyzed. The expected average case performances of flash codes were estimated using computer simulations. Performance was evaluated using two metrics, namely the write deficiency and the average number of data updates.

Simulation results showed the variant KPFC-m compares favorably to ILIFC, LILIFC and CPFC when using write deficiency. On the other hand, its performance as to number of data updates only comes second to a high performing flash code designed specifically in multiple update framework. Overall, the KPFC and its variant KPFC-m

is competitive even in the new framework. This indicates that KPFC is still efficient and suggests that using the flash code can extend the life of flash memory devices.

For future studies, another variant of KPFC can be developed that would really work well in the simultaneous bit update framework. It would be interesting to experiment on cells capable of more data updates like enhancing the pair slice concept.

VIII. References

1. *Agustin, J.S. & Fernandez, P.L. 2015, 'Circular pair flash code', Proceedings of the 15th Phililppine Computing Science Congress.*
2. *Bautista, M. & Fernandez, P. 2014, 'The sequential cascade flash code: A novel flash code with simultaneous bit update', Proceedings of the 11th National Conference on IT Education, pp. 202-10.*
3. *Bez, R., Camerlenghi, E., Modelli, A. & Visconti, A. 2003, 'Introduction to flash memory', Proceedings of the IEEE, vol. 91, no. 4,pp. 489-501.*
4. *Canet, P., Bouquet, V., Lalande, F., Devin, J. &Leconte, B. 2005, 'Fowler-nordheim erasing time prediction in flash memory', 2005 Non-Volatile Memory Technology Symposium, NVMTS05, pp. 15-8.*
5. *Cappelletti, P. &Modelli, A. 1999, 'Flash memory reliability', Flash Memories, Springer, pp. 399-441.*
6. *Chee, Y.M., Kiah, H.M., Vardy, A. &Yaakobi, E. 2017, 'Explicit constructions of finite-length wom codes', IEEE International Symposium on Information Theory - Proceedings, pp. 2860-4.*
7. *Esling, H., Ortiz, R.R. and Fernandez, P., 2013. Bi-modal flash code using index-less indexed flash code and layered index-less indexed flash code. Advanced Science and Technology Letters (Software 2013), 35, pp.19-22.*
8. *Jiang, A., Bohossian, V. & Bruck, J. 2007, 'Floating codes for joint information storage in write asymmetric memories', IEEE International Symposium on Information Theory - Proceedings, pp. 1166-70.*
9. *Jiang, A., Bohossian, V. & Bruck, J. 2010, 'Rewriting codes for joint information storage in flash memories', IEEE Transactions on Information Theory, vol. 56, no. 10,pp. 5300-13.*
10. *Jiang, A. & Bruck, J. 2008, 'Joint coding for flash memory storage', IEEE International Symposium on Information Theory - Proceedings, pp. 1741-5.*
11. *Lee, S., Ha, K., Zhang, K., Kim, J. & Kim, J. 2009, 'FlexFS: A Flexible Flash File System for MLC NAND Flash Memory', USENIX Annual Technical Conference, pp. 1-14.*
12. *Mahdavifar, H., Siegel, P.H., Vardy, A., Wolf, J.K. &Yaakobi, E. 2009, 'A nearly optimal construction of flash codes', IEEE International Symposium on Information Theory - Proceedings, pp. 1239-43.*
13. *Ortiz, R.R.L., Esling, H.R. & Fernandez, P.L. 2014, 'Combining flash codes for write deficiency Reduction', International Journal of Software Engineering and its Applications, vol. 8, no. 4,pp. 47-60.*
14. *Ortiz, R.R.L. and Fernandez Jr, P.L., 2014. The K-Partition Flash Code with BIFC-based Sharing and some Variants''. International Journal of Multimedia and Ubiquitous Engineering, 9(9), pp.381-396.*
15. *Richter, D. 2014, 'Flash Memories: Economic principles of performance, cost and reliability optimization', Springer Series in Advanced Microelectronics, Article, vol. 40.*
16. *Rivest, R.L. & Shamir, A. 1982, 'How to reuse a "write-once memory', Information and control, vol. 55, no. 1-3,pp. 1-19.*
17. *Suzuki, R. &Wadayama, T. 2011, 'Layered index-less indexed flash codes for improving average performance', IEEE International Symposium on Information Theory - Proceedings, pp. 2138-42.*
18. *Tan, M. &Kaji, Y. 2012, 'Uniform compartment flash code and binary-indexed flash code', Technical Report of IEICE, pp. 25-30.*

19. *Tan, M.J. &Kaji, Y. 2013, 'Flash code utilizing binary-indexed slice encoding and resizable-clusters', IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E96-A, no. 12,pp. 2360-7.*

20. *Yaakobi, E., Vardy, A., Siegel, P.H. & Wolf, J.K. 2008, 'Multidimensional flash codes', 46th Annual Allerton Conference on Communication, Control, and Computing, pp. 392-9.*

21. *Yaakobi, E., Yucovich, A., Maor, G. &Yadgar, G. 2015, 'When do wom codes improve the erasure factor in flash memories?', IEEE International Symposium on Information Theory - Proceedings, vol. 2015-June, pp. 2091-5.*

22. *Yamawaki, A., Kamabe, H. & Lu, S. 2017, 'Lower bounds on the number of write operations by index-less indexed flash code with inversion cells', IEEE International Symposium on Information Theory - Proceedings, pp. 2483-7.*