IPSJ Transactions on Mathematical Modeling and Its Applications Vol.8 No.1 1–6 (Mar. 2015)

Regular Paper

Performance Evaluation of Index-less Indexed Flash Codes for Non-uniform Write Operations

Yuichi Kaji[1,a)]

Received: May 19, 2014, Revised: July 14, 2014,
Accepted: September 5, 2014

Abstract: A random-walk model is investigated and utilized to analyze the performance of a coding scheme that aims to extend the lifetime of flash memory. Flash memory is widely used in various products today, but the cells that constitute flash memory wear out as they experience many operations. This issue can be mitigated by employing a clever coding scheme that is known as a flash code. The purpose of this study is to establish a well-defined random-walk model of a flash code that is known as an index-less indexed flash code (ILIFC), and clarify the expected performance of ILIFC. Preliminary study has been made by the author for a simplified model of data operation, and the contribution of this study is to extend the model of data operation to more general and practical one. Mathematical properties of the random-walk model is reconsidered, and useful properties are derived that help analyzing the performance of ILIFC both in non-asymptotic and asymptotic scenarios.

Keywords: flash codes, flash memory, random-walk model, index-less indexed flash codes, coding for storage

1. Introduction

Flash memory is widely used in a number of electronic products today, but data recording in flash memory is not a simple issue. Flash memory consists of many flash *cells* that can store electric charge in their *floating gates*. **Figure 1** illustrates the simplified structure of flash cells [10]. Each cell has its own *control gate*, and several cells (three cells in Fig. 1, but as many as 10^{18} to 10^{20} cells in practical products [8]) share a single *basement* and a *back gate*. The collection of cells that share a back gate is called an *erase block* because of the reason we mention later. There are *insulators* that separate floating gates from control gates and the basement, and therefore floating gates are electrically isolated. Basically, the amount of charge that is stored in the floating gate represents the *value* of the cell. In this study, we let the cell value be an integer in $A_q = \{0, \ldots, q-1\}$, where q is the resolution of the quantization of the charge. If we apply positive voltage to a control gate, then, thanks to the quantum tunnel effect, electrons in the basement "tunnel through" the insulator and they are added to the floating gate. This operation is called a *cell programming*, and is used to raise the value of a selected cell. The charge in a floating gate remains for a long period of time even if the voltage to the control gate is removed, which makes flash memory nonvolatile. An interesting aspect of flash memory is that the operation to remove charge from a floating gate is not symmetrical to the cell programming; the charge in a floating gate is removed by applying positive voltage to the back gate, but this operation affects all cells in the same erase block. With this *block erase* operation, values of all cells in the erase block are reset to 0. The block erasure is a quite strong electric operation, and it can deteriorate the insulators of the affected cells. The deteriorated insulator cannot prevent the charge in a floating gate from leaking out, and it is said that flash cells that have experienced thousands of block erasures are no more suitable for data recording [8]. In this sense, flash memory has finite lifetime in principle.

A number of efforts have been made to extend the lifetime span of flash memory products. For example, many flash memory products are equipped with the mechanism that is known as *wear-leveling* [9]. The wear-leveling is a kind of scheduling algorithm, and contributes to prevent excessive use of small number of specific cells. Recent operating systems inform flash memory controller of higher-level information of data operation, where the controller makes use of the information to avoid needless operations. The TRIM command of the Windows operating system [12] is an example of this device. We can also investigate a data structure that is suitable for flash memory. For example, the file-system architectures of JFFS and JFFS2 are designed to avoid "in-place" rewritings of journal files that are essential in hierarchical file-systems [14].

The use of *flash codes* can be regarded as one of such attempts to extend the lifetime span of flash memory products. The pur-

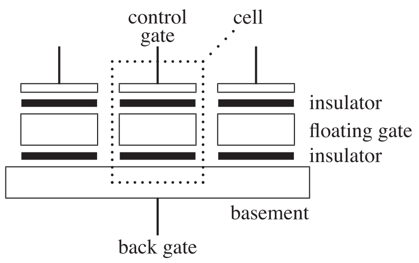


Fig. 1 Flash memory architecture.

[1] Nara Institute of Science and Technology, Ikoma, Nara 630–0101, Japan
a) kaji@is.naist.jp

pose of flash codes is to give a clever way of data representation and data operations in a flash memory. Historically saying, a flash code can be regarded as a descendant of *WOM* (write-once memory) codes that was proposed by Rivest in early 1980s [11]. However, today's framework of flash codes owes much to the studies of Jiang et al. [3], [4], in which K-bit *data* is stored in one erase block with N flash cells, and the K-bit data is updated by a *write operation* which randomly selects one of K-bits of the data and flips the binary value of the selected bit. A flash code is designed to accommodate as many write operations as possible between two consecutive block erasures. Jiang et al. proposed flash codes which allow more number of write operations than naive coding scheme [3], and extended the idea in Ref. [4]. Mahdavifar et al. improved the idea of Ref. [4], and proposed *index-less indexed flash codes* (*ILIFC*) [8].

The purpose of this study is to analyze the *expected* performance of ILIFC. In the traditional discussion of WOM codes and flash codes, the *worst-case* performance has been regarded as significant. The worst-case performance gives the guarantee of the lifetime of a memory product in the most unfortunate case, which is especially important when the memory is really "write-once." However, as stated previously, flash cells today endure thousands of block erasures. It is quite unlikely that the most unfortunate scenario is repeated for thousands times, and the expected performance should have strong and direct relation to the lifetime of mass-produced flash memory products [2], [7]. In Ref. [2], the expected performance is discussed in terms of "cost" of moves of a certain Markov chain model, and a flash code with good expected performance was proposed. The code construction is further improved in Ref. [7], but these two studies do not discuss ILIFC. Suzuki considered to improve the expected performance of ILIFC, and applied the Markov chain formalization of Ref. [2] in their analysis [13]. The formalization contributes to estimate the expected performance of ILIFC with small parameters, though, the approach seems not scalable because we need to construct and analyze a Markov model whose size is exponential in N. Kaji modeled the behavior of ILIFC as a multi-token cyclic random-walk model, and clarified the expected performance of ILIFC in a *uniform writing* scenario in which it is assumed that K data bits are selected by write operations with an equal probability [6].

The contribution of this study is to relax the assumption in Ref. [6], and to extend the discussion in Ref. [6] to a *non-uniform writing* scenario. In many applications of data processing, data to be handled is not homogeneous. In a practical file-systems, for example, some files are updated frequently, while many files remain unchanged for long time. When a user edits an ASCII encoded text file, the most significant bit hardly changes while the other seven bits have great probabilities to be modified. This kind of bias is common in computer system, and the performance of flash codes for such non-uniform writing scenario has practical importance. Fortunately, the random-walk model that was developed in Ref. [6] is still effective for this extended scenario. Unfortunately, however, the analysis techniques that was used in Ref. [6] is no more available because bits of the data have different statistical characteristics. In this study, we reorganize the mathematical discussion in Ref. [6], and refine the analysis technique so that it can be adopted to the more general class of the problem.

2. Preliminary

2.1 Flash Codes

Flash codes and related notions are briefly reviewed in this section. See Refs. [3], [5] for detailed discussion and background issues related to these preliminary. An *erase block* of a flash memory is an array of N *cells*, where a *cell* is an element which stores an integer value in $A_q = \{0, \ldots, q-1\}$ where q is an integer greater than one. A cell is said to be *empty* (resp. *full*) if its value is 0 (resp. $q-1$). Cells in an erase block are ordered, and the value of the i-th cell with $0 \le i < N$ is denoted by c_i. A Tuple $(c_0, \ldots, c_{N-1}) \in A_q^N$ is used to represent the contents of cells, and called a *state* of the erase block. For two states $\mathbf{c} = (c_0, \ldots, c_{N-1})$ and $\mathbf{c}' = (c'_0, \ldots, c'_{N-1})$, we write $\mathbf{c} \le \mathbf{c}'$ if $c_i \le c'_i$ for all $0 \le i < N$, and $\mathbf{c} \prec \mathbf{c}'$ if $\mathbf{c} \le \mathbf{c}'$ and $\mathbf{c} \ne \mathbf{c}'$. The notion of "states" and "$\prec$" are extended to subsets of cells in a natural manner. In an erase block with N cells, we store a value of a K-bit binary *data* $(b_0, \ldots, b_{K-1})$. The value of the data is changed through a *write operation*, which randomly selects one of bits of the data and flips the binary value of the selected bit. This is a simplified model of the data operation for a memory with the striping architecture. In the following discussion, we write p_i with $0 \le i < K$ for the probability of the i-th bit of the data be selected by a write operation.

The purpose of a *flash code* is to give correspondence between $\{0,1\}^K$ (the values of the K-bit data) and A_q^N (states of N cells), and to determine how to operate cell values for a requested write operation. Indeed, a flash code is defined as a pair of functions $C = (\mathcal{E}, \mathcal{D})$. The *decoding function* $\mathcal{D}$ is a mapping from A_q^N to $\{0,1\}^K$, and used to translate the state of the block to a K-bit data value. The *encoding function* $\mathcal{E}$ is a mapping from $\{0, \ldots, K-1\} \times A_q^N$ to $A_q^N \cup \{\mathrm{E}\}$, where E is a special symbol which is called *block erasure*, and determines how to operate cell values. It is required that, if $\mathbf{c}' = \mathcal{E}(i; \mathbf{c})$ and $\mathbf{c}' \ne \mathrm{E}$, then $\mathbf{c} \prec \mathbf{c}'$, and $\mathcal{D}(\mathbf{c})$ and $\mathcal{D}(\mathbf{c}')$ differ at the i-th bit position only. If there is no $\mathbf{c}'$ that satisfies the above conditions, then $\mathcal{E}$ must return E. It is assumed that, at the initial moment, all cells are empty and the K-bit data is $(0, \ldots, 0)$. Write operations are then performed repeatedly, and the encoding function $\mathcal{E}$ is executed for each write operation. Because the state of the block increases monotonically with respect to $\prec$, the encoding function eventually returns E. Let T denote the number of write operations that were accommodated before $\mathcal{E}$ returns E (i.e., E is returned at the $(T+1)$-th call of $\mathcal{E}$). In general, the value of T depends on the bit positions that were selected by the write operations, and hence T is regarded as a random variable. It is understood that $T \le N(q-1)$, and therefore

$$\Delta = N(q-1) - T,$$

which is called a *write deficiency*, indicates the "overhead" that was induced by using the flash code. Obviously, a smaller value of Δ is more favorable. The maximum Δ is called the worst-case write deficiency, and the expected value of Δ is called the expected write deficiency.

2.2 Index-less Indexed Flash Codes

An *index-less indexed flash code* (*ILIFC*) [8] has two different "stages" in encoding. Asymptotically saying, using both of the first and the second stages is good to reduce the worst-case write deficiency [8]. However, the asymptotic difference is so small that the use of the second stage gives very little contribution for practical choices of N. Indeed it often happens that the performance is improved by omitting the second stage of ILIFC [6]. For this reason, we consider ILIFC with the first stage only.

In ILIFC, N cells in an erase block are divided into *slices* with each slice consists of K cells. For simplicity, we assume that N is a multiple of K and there are N/K slices in one erase block. We also assume that $K(q-1)$ is an even number, which is essential for the consistency of ILIFC. A slice consists of K cells $\mathbf{s}_m = (c_{mK}, \ldots, c_{mK+K-1})$ with $0 \le m < N/K$. A slice is *empty* (resp. *full*) if its state is $(0, \ldots, 0)$ (resp. $(q-1, \ldots, q-1)$). A slice which is neither of empty nor full is said to be *active*. The *weight* of a slice is defined as the sum of values of cells in the slice. The key idea of ILIFC is to devise a coding scheme which allows a slice to simultaneously represent the *value* and the *index* of one of bits of the data. For integers i and w with $0 \le i < K$ and $0 \le w \le K(q-1)$, let define $\mathbf{c}_w^{[i]}$ as follows;

- $\mathbf{c}_0^{[0]} = (0, \ldots, 0)$,
- $\mathbf{c}_{w+1}^{[0]}$ is obtained from $\mathbf{c}_w^{[0]}$ by increasing the value of the left-most non-full cell in $\mathbf{c}_w^{[0]}$ by one, and
- $\mathbf{c}_w^{[i+1]}$ is obtained from $\mathbf{c}_w^{[i]}$ by cyclically shifting $\mathbf{c}_w^{[i]}$ to the right direction by one position.

For example, if $q = 3$ and $K = 4$, then $\mathbf{c}_0^{[0]}, \ldots, \mathbf{c}_{K(q-1)}^{[0]}$ are

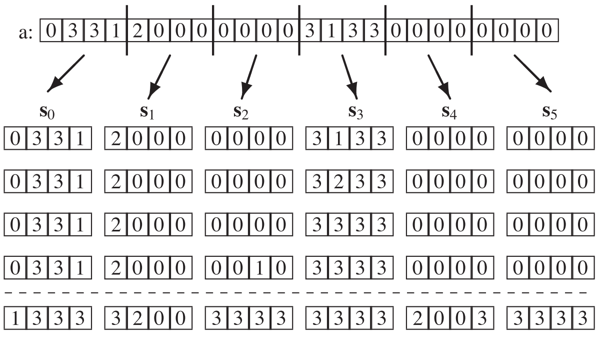
$$0000, 1000, 2000, 2100, 2200, 2210, 2220, 2221, 2222,$$

respectively. States $\mathbf{c}_0^{[2]}, \ldots, \mathbf{c}_{K(q-1)}^{[2]}$ are obtained by cyclically shifting the above states by two-bits each;

$$0000, 0010, 0020, 0021, 0022, 1022, 2022, 2122, 2222.$$

If the state of a slice equals to $\mathbf{c}_w^{[i]}$, then we say that the slice has an *index* i and a *binary value* $b = w \bmod 2$. An *update* is an operation to modify the state of a slice from $\mathbf{c}_w^{[i]}$ to $\mathbf{c}_{w+1}^{[i]}$ where $0 \le i < K$ and $0 \le w < K(q-1)-1$. Note that an update operation increases the weight of a slice by one, and flips the binary-value of the slice. The index of the slice stays unchanged by update operations.

ILIFC manages cell values in such a way that the i-th bit of the K-bit data is 1 if and only if there is an active slice whose index is i and whose binary value is 1. If there is no active slice with index i, or, if there is an active slice with index i but its binary value is 0, then the i-th bit of the data is interpreted as 0. Consider for example that a value (b_0, b_1, b_2, b_3) of the K-bit data is recorded as the state in **Fig. 2** (a), where we assume that $N = 24$, $K = 4$ and $q = 4$. In this case $N = 24$ cells are divided into six slices with $K = 4$ cells each as in Fig. 2 (b). We can determine that the slice $\mathbf{s}_0$ has an index 1 and a binary value of 1, which means that $b_1 = 1$. The slices $\mathbf{s}_1$ and $\mathbf{s}_3$ have indexes 0 and 2, respectively, and their binary values are both 0. Consequently we have $b_0 = b_2 = 0$. There is no active slice with index 3 in Fig. 2 (b), and b_3 is interpreted as 0. Summarizing, the state in Fig. 2 (b) is decoded to the data value $(b_0, b_1, b_2, b_3) = (0, 1, 1, 0)$.


a: 0 3 3 1 2 0 0 0 0 0 0 0 3 1 3 3 0 0 0 0 0 0 0 0
s0 s1 s2 s3 s4 s5
b: 0331 2000 0000 3133 0000 0000
c: 0331 2000 0000 3233 0000 0000
d: 0331 2000 0000 3333 0000 0000
e: 0331 2000 0010 3333 0000 0000
f: 1333 3200 3333 3333 2003 3333
Fig. 2 Illustration of ILIFC.

The encoding function $\mathcal{E}$ operates cell values so that the state is decoded to the current value of the K-bit data. Consider that a write operation requests to flip the i-th bit of the data. The first attempt the encoding function tries is to look for an active slice with index i, and to update the found slice. If there is no active slice with index i, then the function chooses one of empty slices, and *activates* the slice to become $\mathbf{c}_1^{[i]}$. In case there is no empty slice available, then $\mathcal{E}$ returns E. Again consider that the state of the block is given by Fig. 2 (b). If a write operation requests to flip b_2, then the encoding function spots and updates $\mathbf{s}_3$ because its index is 2. The encoding results in Fig. 2 (c). With one more write operation of b_2, the state becomes as in Fig. 2 (d). Note that there is no active slice with index 2 in Fig. 2 (d), but this is not a problem because $b_2 = 0$ at this moment. If another write operation is performed for b_2, then the encoding function activates an empty slice, for example $\mathbf{s}_2$, and the encoding continues. Consider that we have reached Fig. 2 (f), and a write operation of b_2 is requested. In this case, the erase block has no room to accommodate the request, and a block erasure E is returned.

3. Cyclic Random-walk Model

The purpose of this section is to define a mathematical model that contributes to analyze the performance of ILIFC. The model characterizes the distribution of weights of slices that are associated with the bits of the data. A fundamental property of the model is discussed in this section, which will be utilized in the next section for the performance analysis of ILIFC.

3.1 Definition of the Model

In the following discussion, we let $Z = K(q-1)$ which equals to the weight of a full slice. Consider a structure that consists of Z *places* $Q_0, \ldots, Q_{Z-1}$ and K *tokens* $\tau_0, \ldots, \tau_{K-1}$. **Figure 3** illustrates a simple example of such a structure with six states and four tokens, where tokens are represented by numbered small circles. The places are cyclically connected, and we say that $Q_{w+1 \bmod Z}$ is the *next* place of Q_w. The tokens are initially put in the place Q_0, and moved to the next places according to the execution of the encoding function $\mathcal{E}$; if $\mathcal{E}$ is invoked for a bit position i with $0 \le i < K$, then the i-th token τ_i is moved to the next place. From the characteristic of the encoding function, we have the relationship that a token τ_i is in the place Q_w with $1 \le w < Z$ (note that $w = 0$ is not included here) if and only if there is an active slice whose index is i and whose weight is w. The token τ_i is in Q_0 if and only if there is no active slice whose index is i.

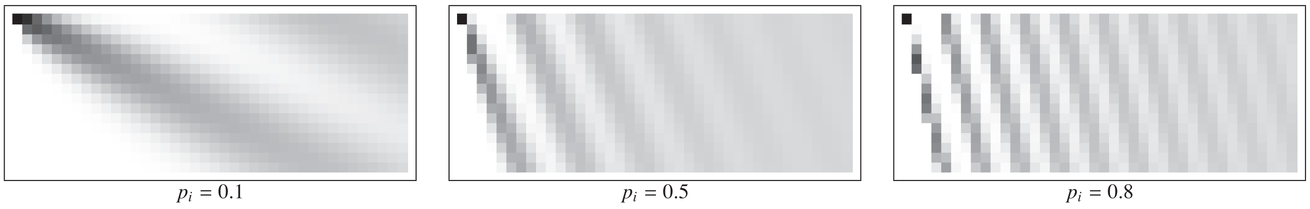


Fig. 4 Convergence of $\mathbf{Y}_i^{(t)}$ to the stationary vector.

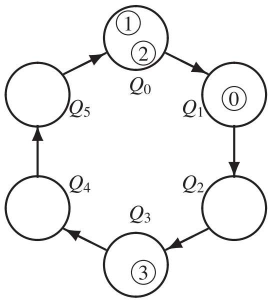


Fig. 3 Cyclic random-walk model with multiple tokens.

This place-and-token structure can be regarded as a *cyclic random-walk model* with *multiple* tokens [1]. At each *move* of the model, one of tokens is selected according to the probabilities $p_0, \dots, p_{K-1}$ (remind that p_i is the probability that a write operation selects the i-th bit of the data), and moved to the next place. We are interested in the distribution of the tokens after t moves of this model, where t is an arbitrary positive integer. For $0 \le w < Z$ and $t \ge 0$, let $X_w^{(t)}$ be a random variable of the number of tokens in the place Q_w after t moves. The expected value of $X_w^{(t)}$, which we denote by $E\left[X_w^{(t)}\right]$, plays a significant role in the analysis of ILIFC. Because it is not easy to derive a closed-form formula of $E\left[X_w^{(t)}\right]$, we consider to express $E\left[X_w^{(t)}\right]$ as a linear combination of simpler and more manageable quantities.

3.2 Investigation on the Expected Value

For $0 \le w < Z$, $0 \le i < K$ and $t \ge 0$, let $Y_{w,i}^{(t)}$ be a random variable which is 1 if the i-th token τ_i is in the place Q_w after t moves of the random-walk model, and 0 otherwise. Obviously

$$X_w^{(t)} = \sum_{i=0}^{K-1} Y_{w,i}^{(t)},$$

and we have the following lemma due to the linearity of the expected values.

Lemma 3.1

$$E\left[X_w^{(t)}\right] = \sum_{i=0}^{K-1} E\left[Y_{w,i}^{(t)}\right].$$

□

The realized value of $Y_{w,i}^{(t)}$ is either of 0 or 1, and therefore $E\left[Y_{w,i}^{(t)}\right]$ equals to the probability of $Y_{w,i}^{(t)} = 1$. For $t \ge 0$, let define a vector of expected values (probabilities) as $\mathbf{Y}_i^{(t)} = \left(E\left[Y_{0,i}^{(t)}\right], \dots, E\left[Y_{Z-1,i}^{(t)}\right]\right)$. Because all tokens are initially in Q_0, we have $\mathbf{Y}_i^{(0)} = (1, 0, \dots, 0)$ for any $0 \le i < K$. A token τ_i stays at the same place with probability $1 - p_i$, and moves to the next place with probability p_i. Consequently

$$\left(\mathbf{Y}_i^{(t)}\right)^{\mathrm{T}} = W_i \left(\mathbf{Y}_i^{(t-1)}\right)^{\mathrm{T}} = W_i^t \left(\mathbf{Y}_i^{(0)}\right)^{\mathrm{T}},$$

where T denotes the transposition of a vector and

$$W_i = \begin{pmatrix} 1-p_i & 0 & \dots & 0 & p_i \\ p_i & 1-p_i & & & \\ & \ddots & \ddots & & \mathbf{O} \\ \mathbf{O} & & \ddots & \ddots & \\ & & & p_i & 1-p_i \end{pmatrix}. \tag{1}$$

Lemma 3.2 $\lim_{t\to\infty} E\left[Y_{w,i}^{(t)}\right] = 1/Z$.

Proof: We can regard that the token τ_i and places $Q_0, \dots, Q_{Z-1}$ define a Markov model. The vector $\mathbf{Y}_i^{(t)}$ is a state probability vector at time t, and Eq. (1) is the state transition matrix of the defined Markov model. It is easily understood that this Markov model is irreducible, and has a unique stationary vector which is given as $\mathbf{V} = (1/Z, \dots, 1/Z)$ (confirm that $\mathbf{V}^{\mathrm{T}} = W_i \mathbf{V}^{\mathrm{T}}$). Because an arbitrary state probability vector converges to the stationary vector $\mathbf{V}$, we have

$$\lim_{t\to\infty} \mathbf{Y}_i^{(t)} = \mathbf{V} \tag{2}$$

for any $0 \le i < K$, and the lemma holds. □

Corollary 3.3 $\lim_{t\to\infty} E\left[X_w^{(t)}\right] = K/Z = K/K(q-1)$. □

We note that the speed of the convergence Eq. (2) depends on the probability p_i that the token τ_i moves. **Figure 4** illustrates how $\mathbf{Y}_i^{(t)}$ changes as the value of t increases, where $Z = 16$ and $p_i = 0.1$ (left), 0.5 (center) and 0.8 (right). Each illustration consists of arrays of square tiles, where the color of the (w, t') component (the top-left tile is the (0,0) component) represents the value of $E\left[Y_{w,i}^{(5t')}\right]$, and therefore one column shows the component values of $\mathbf{Y}_i^{(5t')}$. A tile with denser color means greater probability (expected value); black is 1, white is 0, and gray color represents an intermediate probability. We can see from the figure that $\mathbf{Y}_i^{(t)}$ shows different characteristics for different values of p_i, but it eventually converges to the stationary vector $\mathbf{V} = (1/Z, \dots, 1/Z)$.

4. Expected Write Deficiency of ILIFC

Lemma 3.1 gives the baseline of the performance analysis of ILIFC, but we need to employ different techniques to utilize Lemma 3.1 according to our target. We first consider a *non-asymptotic* scenario in which N is rather small, and the block erasure is returned while the random-walk model is still in the transient behavior. In the second *asymptotic* scenario, we deal with the case that N is quite large and sufficiently many write operations are performed before the block erasure is returned by the encoding function.

4.1 Non-asymptotic Discussion

Remind that the encoding function of ILIFC performs either

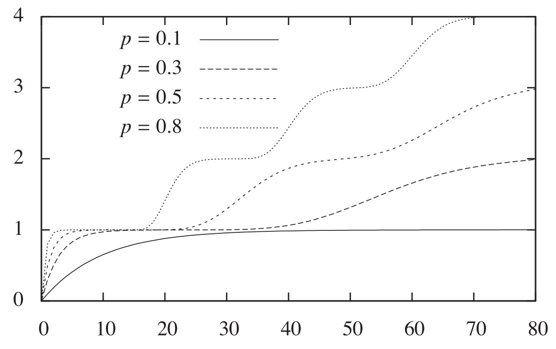


Fig. 5 The accumulation of activated slices (single bit).

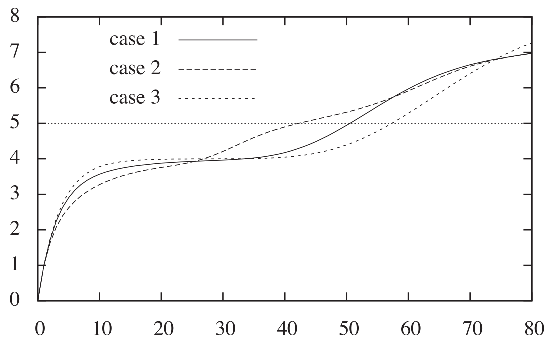


Fig. 6 The accumulation of activated slices (multiple bits).

one of two actions; increase the weight of an active slice, or "activate" an empty slice (i.e., choose an empty slice and raise its weight by one). The block erasure is requested when the encoding function tries to perform the $N/K+1$-th activation, because the number of slices in the erase block is only N/K. Note that the activation of a slice is made when a write operation tries to flip a data bit, say i-th bit, that does not have a corresponding active slice. In this case, the token τ_i must be in the place Q_0 due to the correspondence between the behavior of the encoding function and the random-walk model. Consequently, the probability that an activation of a slice takes place at the t-th write operation is given by $\sum_{i=0}^{K-1} p_i E\left[Y_{0,i}^{(t-1)}\right]$. This probability can be also regarded as the expected number of slices that are newly activated at the t-th write operation, and the accumulation of this value

$$S_t = \sum_{l=1}^{t}\sum_{i=0}^{K-1} p_i E\left[Y_{0,i}^{(l-1)}\right] = \sum_{i=0}^{K-1}\sum_{l=1}^{t} p_i E\left[Y_{0,i}^{(l-1)}\right] \tag{3}$$

gives the expected number of slices which have been activated by t write operations performed so far. Since the block erasure is needed when $(N/K+1)$-th activation is tried, the smallest t with $S_t > N/K+1$ gives the estimation of the number of write operations that causes the block erasure. The expected write deficiency is therefore estimated as $\Delta = N(q-1) - t$ where t is the smallest integer with $S_t > N/K+1$.

Figure 5 shows how the inner summation $\sum_{l=1}^{t} p_i E\left[Y_{0,i}^{(l-1)}\right]$ in Eq. (3) changes where the value of t is varied from 0 to 80. Four different values $p_i = 0.1$, 0.3, 0.5 and 0.8 are sketched. Intuitively, this graph shows the expected number of slices that have been used to represent a certain bit of the data. For example, we can read that the values of the curves for $p = 0.1$ and 0.3 at $t = 40$ are 0.99 and 1.06, respectively. This suggests that a data bit with probability 0.1 (resp. 0.3) should consume approximately 0.99 (resp. 1.06) slices after $t = 40$ write operations. The accumulation of these values gives the total number of slices that are used by either bit of the K-bit data. For example, if $K = 4$ and four bits are selected by write operations with probabilities 0.1, 0.3, 0.3 and 0.3, then, after 40 write operations, $0.99 + 1.06 \times 3 = 4.17$ slices are expected to be in use. We let this example as "case 1." In the "case 2," we consider that four bits are selected by write operations with probabilities 0.1, 0.1, 0.3 and 0.5. In this case, the expected number of slices in use will be $0.99\times2+1.06+1.87 = 4.91$ after 40 write operations. The expected number of slices for these two cases, and the expected number of slices for "uniform" write operations (case 3, in which all four bits are selected with the same probability 0.25), are illustrated in **Fig. 6**. If an erase block contains only four slices (and hence $4 \times 4 = 16$ cells), then block erasure occurs when the encoding function tries to activate the fifth slice. From Fig. 6, we can see that the expected number of slices exceeds 5 at $t = 51$ for the case 1, $t = 43$ for the case 2, and $t = 58$ for the case 3. This means that ILIFC shows different performance for different probability distribution; among these three cases, the case 3 is the most favorable, and we can have $58 - 43 = 15$ more write operations compared to the case 2. We can see that the non-uniform nature of write operations affects the performance of ILIFC for relatively small N.

4.2 Asymptotic Discussion

If there are so many cells in the erase block, then it is expected that many write operations are performed before block erasure occurs. In this case, the asymptotic discussion is feasible and we can take purely analytic approach.

Assume that t write operations have been successfully performed, and a block erasure occurs when the $(t+1)$-th write operation is tried. In ILIFC, the encoding function selects one slice and increases its weight by one. This means that the sum of weights of slices equals to the number of write operations performed so far. The write deficiency Δ is consequently determined as $\Delta = N(q-1)-t = N(q-1)-W$ where W is the sum of weights of all slices at the time just after the t-th write operation.

Lemma 4.1 For very large N, the expected write deficiency $E[\Delta]$ of ILIFC is $(K-1)(K(q-1)-1)/2$.

Proof: In the above discussion, assume without loss of generality that the $(t+1)$-th write operation tries to flip the 0th-bit of the data. Then, there are three types of slices in the erase block after the t-th write operation.

type 1 one full slice that was most-recently used to record the 0th-bit of the data.

type 2 $K-1$ active or full slices that are (were most-recently) used to record the remaining $K-1$ bits of the data.

type 3 $N/K - K$ full slices that are neither of type 1 nor type 2.

Similar to the random variable $X_w^{(t)}$ that was considered in a previous section, let $Z_w^{(t)}$ with $0 \le w < K(q-1)$ be a random variable of the number of slices that are type-2 and with weight w. To simplify the notation, we let $Z_{K(q-1)}^{(t)} = Z_0^{(t)}$. By using these random variables, the sum W of weights of slices can be written as

$$W = K(q-1) + \sum_{w=1}^{K(q-1)} wZ_w^{(t)} + \left(\frac{N}{K} - K\right)K(q-1) \tag{4}$$

where the first, second and the third terms in Eq. (4) are contributions of the types 1, 2 and 3 slices, respectively (note that one full slice contributes $K(q-1)$ to the weight). Remind the definition of $Z_w^{(t)}$, and it is understood that

$$\lim_{t\to\infty} E\left[Z_w^{(t)}\right] = \lim_{t\to\infty} \sum_{i=1}^{K-1} E\left[Y_{w,i}^{(t)}\right] = \frac{K-1}{K(q-1)}.$$

We have

$$\begin{aligned}\lim_{t\to\infty} E[W] &= K(q-1) + \sum_{w=1}^{K(q-1)} w\left(\lim_{t\to\infty} E\left[Z_w^{(t)}\right]\right)\\ &\quad + \left(\frac{N}{K} - K\right)K(q-1)\\ &= K(q-1) + \frac{K(q-1)(K(q-1)+1)}{2}\frac{K-1}{K(q-1)}\\ &\quad + N(q-1) - K^2(q-1)\\ &= N(q-1) - \frac{1}{2}(K-1)(K(q-1)-1),\end{aligned}$$

and

$$\lim_{t\to\infty} E[\Delta] = \frac{1}{2}(K-1)(K(q-1)-1).$$

This completes the proof because $t \to \infty$ if N is very large. □

We remark that the probabilities $p_0, \ldots, p_{K-1}$ do not affect the expected write deficiency in this asymptotic scenario, which is an interesting contrast to the non-asymptotic case.

5. Conclusion

The expected write deficiency of ILIFC is studied for non-uniform write operation. The write deficiency of ILIFC has strong relation to the weight distribution of active slices. The transition of weight distribution can be modeled as a cyclic random-walk with multiple tokens, and the analysis of the model can be decomposed to a simpler problem over a irreducible Markov model. Based on this decomposition, the expected write deficiency is determined for non-asymptotic scenario with the aid of computation. For asymptotic case with very large N, the weight distribution converges to a certain value, and the write deficiency is not affected by the non-uniform nature of write operations. This is an interesting contrast to the non-asymptotic discussion in which the non-uniform nature affects the write deficiency in general. From the practical viewpoint, it is important to clarify which types of discussion is appropriate for a given parameter. It is not easy to answer to this question, but the author conjectures that the asymptotic discussion seems more significant in practical parameters. As stated in Ref. [8], N can be a quite large number such as 10^{18} to 10^{20}. On the other hand, the performance of ILIFC fatally degrades if $K > \sqrt{N}$, and K is expected to be chosen as a number that is much smaller than N. In this parameter setting, there should exist large number of slices, and the non-uniform nature of the write operations will not affect to the performance of ILIFC.

References

[1] Feller, W., *An Introduction to Probability Theory and Its Applications, Third Edition*, Wiley (1968).

[2] Finucane, H., Liu, Z. and Mitzenmacher, M.: Designing Floating Codes for Expected Performance, *Proc. 46th Allerton Conf. Communication, Control and Computing*, pp.1389–1396 (2008).

[3] Jiang, A., Bohossian, V. and Bruck, J.: Floating Codes for Joint Information Storage in Write Asymmetric Memories, *Proc. 2007 Intl. Symp. Inf. Theory*, pp.1166–1170 (2007).

[4] Jiang, A. and Bruck, J.: Joint Coding for Flash Memory Storage, *Proc. 2008 Intl. Symp. Inf. Theory*, pp.1741–1745 (2008).

[5] Jiang, A., Mateesch, R., Schwartz, M. and Bruck, J.: Rank Modulation for Flash Memories, *IEEE Trans. Inf. Theory*, Vol.55, No.6, pp.2659–2673 (2009).

[6] Kaji, Y.: The Expected Write Deficiency of Index-Less Indexed Flash Codes, *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E95-A, No.12, pp.2130–2138 (2012).

[7] Kamabe, H.: Floating Codes with Good Average Performance, *Proc. 32nd Symp. Inf. Theory and Its Applications*, pp.856–861 (2009).

[8] Mahdavifar, H., Siegel, P.H., Vardy, A., Wolf, J.K. and Yaakobi, E.: A Nearly Optimal Construction of Flash Codes, arXiv:0905.1512 (2009).

[9] MicronTechnology, Inc.: Wear Leveling in Micron NAND Flash Memory, Technical Note, TN-29-61 (2010) (retrieved on May 12, 2014), available from ⟨http://www.micron.com/-/media/Documents/Products/Technical Note/NAND Flash/tn2961_wear_leveling_in_nand.pdf⟩.

[10] Olson, A.R. and Langlois, D.J.: Solid State Drives Data Reliability and Lifetime, Imation Corporation White Paper (2008) (retrieved on May 12, 2014), available from ⟨http://www.csee.umbc.edu/~squire/images/ssd1.pdf⟩.

[11] Rivest, R.L. and Shamir, A.: How to Reuse a 'Write-Once' Memory, *Information and Control*, Vol.55, p.1.19 (1982).

[12] Shu, R.: Windows 7 Enhancements for Solid-State Drives, *2008 Windows Hardware Engineering Conference* (2008) (retrieved on May 14, 2014), available from ⟨http://download.microsoft.com/download/F/A/7/FA70E919-8F82-4C4E-8D02-97DB3CF79AD5/COR-T558_Shu_Taiwan.pdf⟩.

[13] Suzuki, R. and Wadayama, T.: Modified Index-less Indexed Flash Codes for Improving Average Performance, *IEICE Trans. Fundamentals* (*Japanese Edition*), Vol.J94-A, No.12, pp.991–1000 (2011) (in Japanese).

[14] Woodhouse, D.: JFFS2: The Journalling Flash File System, version 2 (2003) (retrieved on May 14, 2014), available from ⟨https://www.sourceware.org/jffs2/⟩.

Yuichi Kaji was born in Osaka, Japan, on December 23, 1968. He received his B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1991, 1992 and 1994, respectively. In 1994, he joined Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. In 2003 and 2004, he visited the University of California Davis and the University of Hawaii at Manoa as a visiting researcher. His current research interests include the theory of error correcting codes, fundamental techniques for information security, and the theory of automata and rewriting systems. He is a member of IPSJ and IEEE.