

# Storage Coding for Wear Leveling in Flash Memories

Anxiao (Andrew) Jiang\*  
Jehoshua Bruck<sup>†</sup>

Robert Mateescu<sup>†</sup>  
Alexander Vardy<sup>‡</sup>

Eitan Yaakobi<sup>‡</sup>  
Jack K. Wolf<sup>‡</sup>

\*Department of Computer Science  
Texas A&M University  
College Station, TX 77843, U.S.A.  
ajiang@cs.tamu.edu

<sup>†</sup>California Institute of Technology  
1200 E California Blvd., Mail Code 136-93  
Pasadena, CA 91125, U.S.A.  
{mateescu,bruck}@paradise.caltech.edu

<sup>‡</sup>Electrical and Computer Engineering  
University of California, San Diego  
La Jolla, CA 92093, U.S.A.  
{eyaakobi,psiegel,avardy,jwolf}@ucsd.edu

**Abstract**—NAND flash memories are currently the most widely used flash memories. In a NAND flash memory, although a cell block consists of many pages, to rewrite one page, the whole block needs to be erased and reprogrammed. Block erasures determine the longevity and efficiency of flash memories. So when data is frequently reorganized, which can be characterized as a data movement process, how to minimize block erasures becomes an important challenge. In this paper, we show that coding can significantly reduce block erasures for data movement, and present several optimal or nearly optimal algorithms. While the sorting-based non-coding schemes require  $O(n \log n)$  erasures to move data among  $n$  blocks, coding-based schemes use only  $O(n)$  erasures and also optimize the utilization of storage space.

## I. INTRODUCTION

Flash memories have become the most widely used non-volatile electronic memories. They have two basic types: NAND and NOR flash memories [6]. Between them, NAND flash is currently used much more often due to its higher data density. In a NAND flash, floating-gate cells are organized as *blocks*. Each block is further partitioned into multiple *pages*, and every read or write operation accesses a page as a unit. Typically, a page has 2 to 4KB of data, and 64 pages form a block [6]. The flash memory has a unique *block erasure* property: although every page can be written individually, to rewrite a page (namely, to change its content), the whole block must be erased and then reprogrammed. Every block can endure  $10^4 \sim 10^5$  erasures, after which the flash memory may break down. Block erasures also reduce the quality of cells and the general efficiency. So it is critical to minimize block erasures. For this reason, numerous *wear leveling* techniques have been used to balance the erasures of blocks [6].

In a flash memory, data often needs to be moved. For example, files can have their segments scattered due to modifications, and need to be reassembled later. Files of similar statistics may also need to be grouped for easier information access. To facilitate data movement, a flash translation layer (FTL) is usually used in flash file systems to map logical data pages to physical pages [6]. How to minimize block erasures during the data movement process remains a main challenge.

In this paper, we show that coding techniques can significantly reduce block erasures for data movement. Besides erasures, we also consider coding complexity and the extra storage space needed for data movement. We show that without coding, at least two empty blocks are needed to facilitate data movement, and present a sorting-based solution

that uses  $O(n \log n)$  block erasures for moving data among  $n$  blocks. With coding, only one empty auxiliary block is needed, and we present a very efficient algorithm based on coding over  $GF(2)$  that uses only  $2n$  erasures. We further present a coding-based algorithm using at most  $2n - 1$  erasures, which is worst-case optimal. Although minimizing erasures for every instance is NP hard, both algorithms that use coding achieve an approximate ratio of two with respect to an optimal solution that minimizes the number of block erasures.

There have been multiple recent works on coding for flash memories, including codes for efficient rewriting [5] [7] [11], error-correcting codes [4], and rank modulation for reliable cell programming [8] [10]. This paper is the first work on storage coding at the page level instead of the cell level, and the topic itself is also distinct from all previous works.

Due to limited space, we skip some details in this paper. Interested readers are referred to [9] for the full analysis.

## II. TERMS AND CONCEPTS

**Definition 1** (DATA MOVEMENT PROBLEM) *There are  $n$  blocks storing data in the flash memory, where every block has  $m$  pages. The blocks are denoted by  $B_1, \dots, B_n$ , and the  $m$  pages in block  $B_i$  are denoted by  $p_{i,1}, \dots, p_{i,m}$  for  $i = 1, \dots, n$ . Let  $\alpha(i, j)$  and  $\beta(i, j)$  be two functions:*

$$\begin{aligned} \alpha(i, j) &: \{1, \dots, n\} \times \{1, \dots, m\} \rightarrow \{1, \dots, n\}; \\ \beta(i, j) &: \{1, \dots, n\} \times \{1, \dots, m\} \rightarrow \{1, \dots, m\}. \end{aligned}$$

*The data in page  $p_{i,j}$  is denoted by  $D_{i,j}$  and needs to be moved into page  $p_{\alpha(i,j),\beta(i,j)}$ , for  $(i, j) \in \{1, \dots, n\} \times \{1, \dots, m\}$ . (Clearly, the functions  $\alpha(i, j)$  and  $\beta(i, j)$  together have to form a permutation for the  $mn$  pages. To avoid trivial cases, we assume that every block has at least one page whose data needs to be moved to another block.)*

*A number of empty blocks, called auxiliary blocks, can be used in the data movement process, and they need to be erased in the end. The objective is to minimize the total number of block erasures in the data movement process.*

The challenge is that a block must be erased before any of its pages is modified. Let us first define some terms. There are two useful graph representations for the data movement problem: the *transition graph* and a *bipartite graph*. In the *transition graph*  $G = (V, E)$ ,  $|V| = n$  vertices represent the  $n$  data blocks  $B_1, \dots, B_n$ . If  $y$  pages of data need to be moved

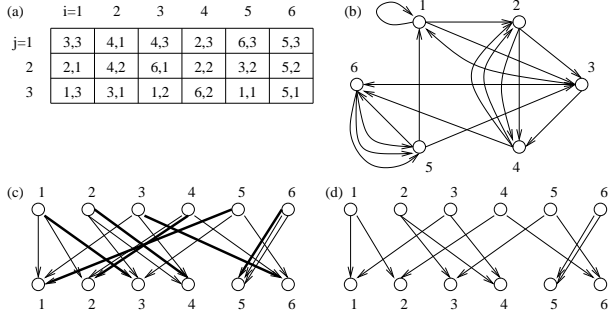


Fig. 1. Data movement with  $n = 6, m = 3$ . (a) The permutation table. The numbers with coordinates  $(i, j)$  are  $\alpha(i, j), \beta(i, j)$ . For example,  $(\alpha(1, 1), \beta(1, 1)) = (3, 3)$ , and  $(\alpha(1, 2), \beta(1, 2)) = (2, 1)$ . (b) Transition graph. (c) The bipartite graph representation. The  $n$  thick edges are a perfect matching (a block-permutation set). (d) After removing a perfect matching from the bipartite graph. Here for  $i = 1, \dots, n$ , vertex  $i$  represents block  $B_i$ .

from  $B_i$  to  $B_j$ , then there are  $y$  directed edges from  $B_i$  to  $B_j$  in  $G$ .  $G$  is a regular directed graph with  $m$  outgoing edges and  $m$  incoming edges for every vertex. In the *bipartite graph*  $H = (V_1 \cup V_2, E')$ ,  $V_1$  and  $V_2$  each has  $n$  vertices that represent the  $n$  blocks. If  $y$  pages of data are moved from  $B_i$  to  $B_j$ , there are  $y$  directed edges from vertex  $B_i \in V_1$  to vertex  $B_j \in V_2$ . The two graphs are equivalent but are used in different proofs.

**Definition 2** (BLOCK-PERMUTATION SET AND SEMI-CYCLE) A set of  $n$  pages  $\{p_{1,j_1}, p_{2,j_2}, \dots, p_{n,j_n}\}$  is a *block-permutation set* if  $\{\alpha(1, j_1), \alpha(2, j_2), \dots, \alpha(n, j_n)\} = \{1, 2, \dots, n\}$ . If  $\{p_{1,j_1}, p_{2,j_2}, \dots, p_{n,j_n}\}$  is a *block-permutation set*, then the data they originally store –  $\{D_{1,j_1}, D_{2,j_2}, \dots, D_{n,j_n}\}$  – is called a *block-permutation data set*.

Let  $z \in \{1, 2, \dots, n\}$ . An ordered set of pages  $(p_{i_0, j_0}, p_{i_1, j_1}, \dots, p_{i_{z-1}, j_{z-1}})$  is a *semi-cycle* if for  $k = 0, 1, \dots, z-1$ ,  $\alpha(i_k, j_k) = i_{k+1 \bmod z}$ .

**Example 3** The data movement problem in Fig. 1 exemplifies the construction of the transition and bipartite graphs. The  $nm = 18$  pages can be partitioned into three block-permutation sets:  $\{p_{1,1}, p_{2,2}, p_{3,2}, p_{4,2}, p_{5,3}, p_{6,1}\}$ ,  $\{p_{1,2}, p_{2,1}, p_{3,3}, p_{4,3}, p_{5,2}, p_{6,2}\}$ ,  $\{p_{1,3}, p_{2,3}, p_{3,1}, p_{4,1}, p_{5,1}, p_{6,3}\}$ . The block permutation sets can be further decomposed into six semi-cycles:  $(p_{5,3}, p_{1,1}, p_{3,2}, p_{6,1})$ ,  $(p_{2,2}, p_{4,2})$ ;  $(p_{5,2}, p_{3,3}, p_{1,2}, p_{2,1}, p_{4,3}, p_{6,2})$ ;  $(p_{1,3})$ ,  $(p_{2,3}, p_{3,1}, p_{4,1})$ ,  $(p_{5,1}, p_{6,3})$ .

**Theorem 4** The  $nm$  pages can be partitioned into  $m$  block-permutation sets. Therefore, the  $nm$  pages of data can be partitioned into  $m$  block-permutation data sets.

*Proof:* The data movement problem can be represented by the *bipartite graph*, where every edge represents a page whose data needs to be moved into another block. (See Fig. 1 (c) for an example.) For  $i = 1, \dots, n$ , any  $i$  vertices in the top layer have  $im$  outgoing edges and therefore are connected to at least  $i$  vertices in the bottom layer. So by Hall's theorem for matching in bipartite graphs [3], the bipartite graph has a perfect matching. The edges of the perfect matching correspond

to a block-permutation set. If we remove those edges, we get a bipartite graph of degree  $m - 1$  for every vertex. (See Fig. 1 (c), (d).) Similarly, we can find another perfect matching and further reduce the graph to regular degree  $m - 2$ . In this way, we partition the  $nm$  edges into  $m$  block-permutation sets. ■

A perfect matching can be found using the Ford-Fulkerson Algorithm [3] for computing maximum flow in time  $O(n^2m)$ . So we can partition the  $nm$  pages into  $m$  block-permutation sets in time  $O(n^2m^2)$ .

### III. CODING FOR MINIMIZING AUXILIARY BLOCKS

In this paper, we focus on the scenario where as few auxiliary blocks as possible are used in the data movement process. In this section, we show that coding techniques can minimize the number of auxiliary blocks. Afterwards, we will study how to use coding to minimize block erasures.

#### A. Data Movement without Coding

When coding is not used, data is directly copied from page to page. It can be shown that in the worst case, more than one auxiliary block is needed for data movement. (Please see [9] for a detailed analysis.) We now show that two auxiliary blocks are sufficient. The next algorithm operates in a way similar to bubble sort. And it sorts the data of the  $m$  block-permutation data sets in parallel. The two auxiliary blocks are denoted by  $B_0$  and  $B'_0$ .

#### Algorithm 5 (BUBBLE-SORT-BASED DATA MOVEMENT)

For  $i = 1, \dots, n - 1$

For  $j = i + 1, \dots, n$

Copy  $B_i$  into  $B_0$  and  $B_j$  into  $B'_0$ ; Erase  $B_i$  and  $B_j$ ;

For  $k = 1, \dots, m$

Let  $D_{i_1, j_1}$  and  $D_{i_2, j_2}$  be the two pages of data in  $B_0$  and  $B'_0$ , respectively, that belong to the  $k$ -th block-permutation data set. Let  $p_{i, j_3}$  be the unique page in  $B_j$  such that some data of the  $k$ -th block-permutation data set needs to be moved into it.

If  $\alpha(i_2, j_2) = i$  (which implies  $\beta(i_2, j_2) = j_3$  and  $\alpha(i_1, j_1) \neq i$ ), copy  $D_{i_2, j_2}$  into  $p_{i, j_3}$ ; otherwise, copy  $D_{i_1, j_1}$  into  $p_{i, j_3}$ .

Write into  $B_j$  the  $m$  pages of data in  $B_0$  and  $B'_0$  but not in  $B_i$ . Erase  $B_0$  and  $B'_0$ .

In the above algorithm, for every block-permutation data set, its data is not only sorted in parallel with other block-permutation data sets, but is also always dispersed in  $n$  blocks (with every block holding one page of its data). The algorithm uses  $O(n^2)$  erasures. If instead of bubble sorting, we use more efficient sorting networks such as the Batcher sorting network [2] or the AKS network [1], the number of erasures can be further reduced to  $O(n \log^2 n)$  and  $O(n \log n)$ , respectively. For simplicity we skip the details.

#### B. Storage Coding with One Auxiliary Block

In Algorithm 5, the only function of the auxiliary blocks  $B_0$  and  $B'_0$  is to store the data in the data blocks  $B_i, B_j$  when the data in  $B_i, B_j$  is being swapped. We now show how coding





