Storage Coding for Wear Leveling in Flash Memories

Anxiao (Andrew) Jiang[*] Robert Mateescu[†] Eitan Yaakobi[‡]
Jehoshua Bruck[†] Paul H. Siegel[‡] Alexander Vardy[‡] Jack K. Wolf[‡]

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

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

[‡]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 $B_1, \ldots, B_n$, and the m pages in block B_i are denoted by $p_{i,1}, \ldots, p_{i,m}$ for $i = 1, \ldots, n$. Let $\alpha(i,j)$ and $\beta(i,j)$ be two functions:*

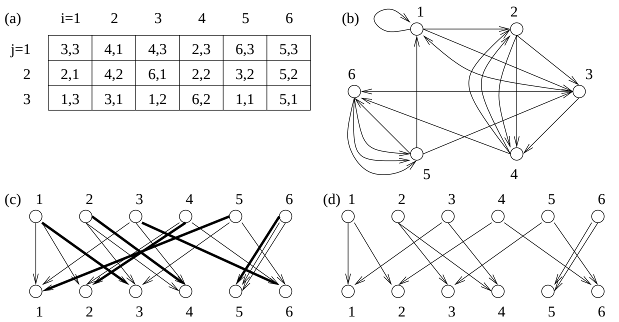
$$\alpha(i,j) : \{1,\ldots,n\} \times \{1,\ldots,m\} \to \{1,\ldots,n\};$$
$$\beta(i,j) : \{1,\ldots,n\} \times \{1,\ldots,m\} \to \{1,\ldots,m\}.$$

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 $(i,j) \in \{1,\ldots,n\} \times \{1,\ldots,m\}$. (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 $B_1, \ldots, B_n$. If y pages of data need to be moved

978-1-4244-4313-0/09/$25.00 ©2009 IEEE



	i=1	2	3	4	5	6
j=1	3,3	4,1	4,3	2,3	6,3	5,3
2	2,1	4,2	6,1	2,2	3,2	5,2
3	1,3	3,1	1,2	6,2	1,1	5,1

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 $i = 1, \ldots, n$, 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 $\{p_{1,j_1}, p_{2,j_2}, \ldots, p_{n,j_n}\}$ is a block-permutation set if $\{\alpha(1, j_1), \alpha(2, j_2), \ldots, \alpha(n, j_n)\} = \{1, 2, \ldots, n\}$. If $\{p_{1,j_1}, p_{2,j_2}, \ldots, p_{n,j_n}\}$ is a block-permutation set, then the data they originally store – $\{D_{1,j_1}, D_{2,j_2}, \ldots, D_{n,j_n}\}$ – is called a* block-permutation data set.

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

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 $i = 1, \ldots, n$, 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 $O(n^2 m)$. So we can partition the nm pages into m block-permutation sets in time $O(n^2 m^2)$.

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 $i = 1, \ldots, n-1$
  For $j = i+1, \ldots, n$
    Copy $B_i$ into $B_0$ and $B_j$ into $B_0'$; Erase $B_i$ and $B_j$;
    For $k = 1, \ldots, m$
      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 $B_i$ 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 can help reduce the number of auxiliary blocks to one, which is clearly optimal. Let B_0 denote the only auxiliary block, and let $p_{0,1}, p_{0,2}, \ldots, p_{0,m}$ denote its pages. For $k = 1, \ldots, m$, statically store in page $p_{0,k}$ the bit-wise exclusive-OR of the n pages of data in the k-th block-permutation data set. We make one change in Algorithm 5: when the data in B_i, B_j is being swapped, instead of erasing them together, we first erase B_i and write data into B_i, then erase B_j and write data into B_j. This is feasible because B_0 always provides enough redundant data. The number of block erasures is of the same order as before.

IV. Efficient Storage Coding over $GF(2)$

In this section, we present a data movement algorithm that uses only one auxiliary block and $2n$ erasures. The algorithm uses coding over $GF(2)$ and is very efficient.

For convenience, let us assume for now that every block has only one page. The results will be naturally extended to the general case. Let B_0 denote the auxiliary block, and let p_0 denote its page. For $i = 1, \ldots, n$, let p_i denote the page in B_i, and let D_i denote the data originally in p_i. Let $\alpha : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be the permutation such that D_i needs to be moved into $p_{\alpha(i)}$. Let α^{-1} be the inverse permutation of α. Say that the n pages can be partitioned into t semi-cycles, denoted by $C_1, \ldots, C_t$. Every semi-cycle C_i $(1 \leq i \leq t)$ has a special page called *tail*, defined as follows: if p_j is the *tail* of C_i, then for every other page $p_k \in C_i$, $j > k$.

We use "$\oplus$" to represent the bit-wise exclusive-OR of data. The following algorithm consists of two passes: the *forward pass* and the *backward pass*. It uses $2n$ erasures. Note that in the algorithm below, whenever some data is to be written into a page, that data can be efficiently computed from the existing data in the flash memory blocks. The detail will be clear later. Also note that $\forall\ 1 \leq i \leq n$, $D_{\alpha^{-1}(i)}$ is the data that needs to be moved into the block that originally contains D_i.

Algorithm 6 *($GF(2)$-CODING-BASED DATA MOVEMENT)*

```
FORWARD PASS:
 For $i = 1, 2, \ldots, n$ do:
   If $p_i$ is not the tail of its semi-cycle, write $D_i \oplus D_{\alpha^{-1}(i)}$
   into $p_{i-1}$; otherwise, write $D_i$ into $p_{i-1}$. Then, erase $B_i$;
BACKWARD PASS:
 For $i = n, n-1, \ldots, 1$ do:
   Write $D_{\alpha^{-1}(i)}$ into $p_i$. Erase $B_{i-1}$.
```

Example 7 *Figure 2 gives an example of the execution of Algorithm 6 with $n = 8$ and $t = 2$. Here $(\alpha(1), \alpha(2), \ldots, \alpha(8)) = (3, 6, 8, 1, 2, 5, 4, 7)$. (Consequently, $(\alpha^{-1}(1), \alpha^{-1}(2), \ldots, \alpha^{-1}(8)) = (4, 5, 1, 7, 6, 2, 8, 3)$.) The two semi-cycles are $(p_1, p_3, p_8, p_7, p_4)$ and (p_2, p_6, p_5). In Figure 2, each row is a step of Algorithm 6. The numbers are the data in the blocks. (For convenience, we use i to denote data D_i in the figure.) The rightmost column describes the computation performed for this step, where δ_i denotes the data in p_i then.*

B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	Operation
forward pass									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	$\delta_1 \oplus \delta_4$
$1 \oplus 4$		2	3	4	5	6	7	8	$\delta_2 \oplus \delta_5$
$1 \oplus 4$	$2 \oplus 5$		3	4	5	6	7	8	$\delta_3 \oplus \delta_0 \oplus \delta_4$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$		4	5	6	7	8	$\delta_4 \oplus \delta_7$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$		5	6	7	8	$\delta_5 \oplus \delta_6$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$		6	7	8	copy δ_6
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6		7	8	$\delta_7 \oplus \delta_8$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6	$7 \oplus 8$		8	copy δ_8
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6	$7 \oplus 8$	8		
backward pass									
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6	$7 \oplus 8$	8		$\delta_7 \oplus \delta_6 \oplus \delta_3 \oplus \delta_0 \oplus \delta_2$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6	$7 \oplus 8$		3	$\delta_6 \oplus \delta_3 \oplus \delta_0 \oplus \delta_2 \oplus \delta_8$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$	6		8	3	$\delta_5 \oplus \delta_4 \oplus \delta_1$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$	$5 \oplus 6$		2	8	3	$\delta_4 \oplus \delta_1 \oplus \delta_6$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$	$4 \oplus 7$		6	2	8	3	$\delta_3 \oplus \delta_0 \oplus \delta_2 \oplus \delta_8$
$1 \oplus 4$	$2 \oplus 5$	$3 \oplus 1$		7	6	2	8	3	$\delta_2 \oplus \delta_8$
$1 \oplus 4$	$2 \oplus 5$		1	7	6	2	8	3	$\delta_1 \oplus \delta_6$
$1 \oplus 4$		5	1	7	6	2	8	3	$\delta_0 \oplus \delta_3$
	4	**5**	**1**	**7**	**6**	**2**	**8**	**3**	

Fig. 2. Example execution of Algorithm 6.

The correctness of Algorithm 6 depends on whether the data written into a page can always be derived from the existing data in the flash memory blocks. Theorem 8 shows this is true.

Theorem 8. *When Algorithm 6 is running, at any moment, $\forall\ 1 \leq i \leq n$, if the data D_i is not in the $n+1$ blocks $B_0, B_1, \ldots, B_n$, then there must exist a set of data $\{D_i \oplus D_{j_1}, D_{j_1} \oplus D_{j_2}, D_{j_2} \oplus D_{j_3}, \ldots, D_{j_{k-1}} \oplus D_k, D_k\}$ that all exist in the $n+1$ blocks. Therefore, D_i can be easily obtained by computing the bit-wise exclusive-OR of the data in the set.*

Proof: Consider a semi-cycle C_i $(1 \leq i \leq t)$. Denote its pages by $p_{i_1}, p_{i_2}, \ldots, p_{i_x}$. Without loss of generality (WLOG), assume $\alpha(i_j) = i_{j+1}$ for $j = 1, 2, \ldots, x-1$, and $\alpha(i_x) = i_1$. Now imagine a directed cycle S as follows: "S has x vertices, representing the data $D_{i_1}, D_{i_2}, \ldots, D_{i_x}$; there is a directed edge from D_{i_j} to $D_{i_{j+1}}$ for $j = 1, \ldots, x-1$, and a directed edge from D_{i_x} to D_{i_1}." Let every directed edge in S represent the bit-wise exclusive-OR of the data represented by its two endpoint vertices.

Consider the *forward pass* in the algorithm. In this pass, every time some data represented by a vertex in S is erased, the data represented by the directed edge entering that vertex already exists. So for every vertex in S whose data has been erased, there is a directed path in S entering it with this property: "the data represented by the edges in this path, as well as the data represented by the starting vertex of the path, all exist in the blocks." This is the same condition stated in the theorem. The *backward pass* can be analyzed similarly. (Please see [9] for details). So the conclusion holds. ■

Algorithm 6 can be easily extended to the case where a block has $m \geq 1$ pages. Use the algorithm to process the m block-permutation data sets in parallel, in the same way as Algorithm 5. Specifically, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, let $p_{i,k(i,j)}$ denote the unique page in B_i such that some data in the j-th block-permutation data set needs to be moved into $p_{i,k(i,j)}$. In the algorithm, every time B_i is erased, write the data related to the j-th block-permutation data set into $p_{i,k(i,j)}$. Since every block-permutation set occupies exactly one page in each block, there will be no conflict in writing.

V. Storage Coding with Minimized Erasures

In this section, we present an algorithm that uses at most $2n-1$ erasures, which is worst-case optimal. We further show that minimizing erasures for every instance is NP hard, but our algorithm provides a 2-approximation.

A. Optimal Solution with Canonical Labelling

The n blocks can be labelled by $B_1, \dots, B_n$ in $n!$ different ways. Let y be an integer in $\{0, 1, \dots, n-2\}$. We call a labelling of blocks that satisfies the following constraint a *canonical labelling with parameter* y: "$\forall\ i \in \{y+1, y+2, \dots, n-2\}$ and $j \in \{i+2, i+3, \dots, n\}$, no data in B_j needs to be moved into B_i." Trivially, any labelling is a canonical labelling with parameter $n-2$. However, it is difficult to find a canonical labelling that minimizes y.

We now present a data-movement algorithm for blocks that have a canonical labelling with parameter y. It uses one auxiliary block B_0, and uses $n+y+1 \leq 2n-1$ erasures. For convenience, let us again assume that every block contains only one page, and let $p_i, D_i, \alpha, \alpha^{-1}$ be as defined in the previous section. Let r denote the number of bits in a page. [1] The algorithm can be naturally generalized for the general case, where every block has $m \geq 1$ pages, in the same way introduced in the previous section.

Algorithm 9 *(*Data Movement with Linear Coding*) This algorithm is for blocks that have a canonical labelling with parameter* $y \in \{0, 1, \dots, n-2\}$*. Let* $\gamma_1, \gamma_2, \dots, \gamma_n$ *be distinct non-zero elements in the field* $GF(2^r)$*.*

Step *1: For* $i = 0, 1, \dots, y$ *do: Erase* B_i *(for* $i = 0$ *there is no need to erase* B_0*), and write into* p_i *the data* $\sum_{k=1}^{n} \gamma_k^i D_k$*.*

Step *2: For* $i = y+1, y+2, \dots, n$ *do: Erase* B_i*, and write into* p_i *the data* $D_{\alpha^{-1}(i)}$*.*

Step *3: For* $i = y, y-1, \dots, 1$ *do: Erase* B_i*, and write into the page* p_i *the data* $D_{\alpha^{-1}(i)}$*.*

Theorem 10 *Algorithm* 9 *is correct and uses* $n+y+1 \leq 2n-1$ *erasures. (Note that the algorithm assumes that the blocks have a canonical labelling with parameter* y*.)*

Proof: We show that each time a block B_i is erased it is possible to generate all n data pages using the current data written in the other n pages. Denote by δ_i, $0 \leq i \leq n$, the current data written in each page, which is a linear combination of the n data pages. The linear combination written in each page can be represented by a matrix multiplication

$$H \cdot (D_1, D_2, \dots, D_n)^T = (\delta_0, \dots, \delta_{i-1}, \delta_{i+1}, \dots, \delta_n)^T.$$

The matrix H defines the linear combination of data pages written in each page. Consider the first step when the block B_i is erased. The data written in p_h, for $0 \leq h \leq i-1$, is $\delta_h = \sum_{k=1}^{n} \gamma_k^h D_k$, and the data written in p_h, for $i+1 \leq h \leq n$, is $\delta_h = D_h$. The matrix representation of this problem is

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \gamma_1 & \gamma_2 & \cdots & \gamma_n \\ \gamma_1^2 & \gamma_2^2 & \cdots & \gamma_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_1^{i-1} & \gamma_2^{i-1} & \cdots & \gamma_n^{i-1} \\ \multicolumn{2}{c}{0_{(n-i)\times i}} & \multicolumn{2}{c}{I_{n-i}} \end{pmatrix} \cdot \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \vdots \\ D_{n-1} \\ D_n \end{pmatrix} = \begin{pmatrix} \delta_0 \\ \vdots \\ \delta_{i-1} \\ \delta_{i+1} \\ \vdots \\ \delta_n \end{pmatrix}$$

where $0_{(n-i)\times i}$ is the zero matrix of size $(n-i) \times i$, and I_{n-i} is the unit matrix of size $(n-i) \times (n-i)$. Since this matrix is invertible it is possible to generate all data pages and in particular the required data that has to be written in p_i. Similarly, it is possible to show for the two other steps that the matrix, representing the linear combination of pages after erasing each block, is invertible. For the sake of space we omit the full proof and it appears in [9]. ∎

Theorem 11 *Assume* r *is sufficiently large. Let* $y \in \{0, 1, \dots, n-2\}$*. There is a data-movement solution using* $n+y+1$ *erasures if and only if there is a canonical block labelling with parameter* y*.*

Proof: First, assume that there is a data-movement solution using $n+y+1$ erasures. Since every block (including the auxiliary block) is erased at least once, there are at least $n-y$ blocks that are erased only once in the solution. Pick $n-y$ blocks erased only once and label them as $B_{y+1}, B_{y+2}, \dots, B_n$ this way: "in the solution, when $y+1 \leq i < j \leq n$, B_i is erased before B_j." Label the other y blocks as $B_1, \dots, B_y$ arbitrarily. Let us use contradiction to prove that no data in B_j needs to be moved into B_i, where $i \geq y+1$, $j \geq i+2$.

Assume some data in B_j needs to be moved into B_i. After B_i is erased, that data must be written into B_i because B_i is erased only once. When the solution erases B_{i+1} (which is before erasing B_j), the data mentioned above exists in both B_i and B_j. However, note that at the end of the solution all nm pages are located in their designated location. But, it is impossible to generate them using only $nm-1$ data pages, so there is a contradiction. Therefore, we have found a canonical labelling with parameter y. The other direction of the proof comes from the existence of Algorithm 9. ∎

We can easily make Algorithm 9 use $2n-1$ erasures by using $y = n-2$ and an arbitrary block labelling. $2n-1$ erasures are also necessary in the worst case. To see that, consider the case where every block has some data that needs to be moved into every other block, where a canonical labelling must have $y = n-2$. So Algorithm 9 is worst-case optimal.

B. Optimization for All Instances

A specific instance of the data movement problem may require less than $2n-1$ erasures. So it is interesting to find an algorithm that minimizes the number of erasures for every instance. The following theorem shows that this is NP hard.

[1] When r is greater than what is needed by Algorithm 9 (which is nearly always true in practice), we can partition each page into bit strings of an appropriate length, and apply the algorithm to the strings in parallel.

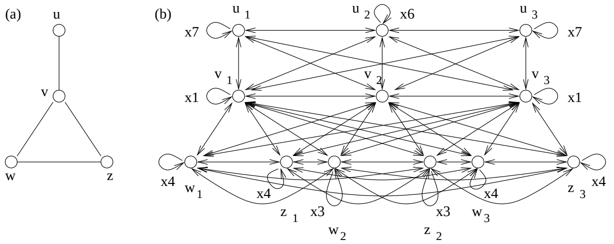


Fig. 3. NP hardness of the data movement problem. (a) A simple undirected graph G_0. (b) The corresponding regular directed graph G'. Here every edge between two different vertices has arrows on both sides, representing the two directed edges of opposite directions between those two vertices. There is a symbol $\times i$ beside every directed loop, representing i parallel loops of that vertex.

Theorem 12. *For the data movement problem, it is NP hard to minimize the number of erasures for every given instance.*

Proof: It has been shown in Theorem 11 and its proof that minimizing the number of erasures is as hard as finding a canonical block labelling with a minimized parameter y. So we just need to show that finding a canonical labelling with minimized y is NP hard. We prove it by a reduction from the NP hard MAXIMUM INDEPENDENT SET problem.

Let $G_0 = (V_0, E_0)$ be any simple undirected graph. Let $d(v)$ denote the degree of vertex $v \in V_0$ and let $\Delta = \max_{v \in V_0} d(v)$ denote the maximum degree of G_0. We build a regular directed graph $G' = (V_1 \cup V_2 \cup V_3, E')$ as follows. Let $|V_0| = |V_1| = |V_2| = |V_3|$. For all $v \in V_0$, there are three corresponding vertices $v_1 \in V_1, v_2 \in V_2, v_3 \in V_3$. If there is an undirected edge between $u, v \in V_0$ in G_0, then there are two directed edges of opposite directions between u_i and v_j for $i = 1, 2, 3$ and $j = 1, 2, 3$. For all $v \in V_0$, there are also two directed edges of opposite directions between v_1, v_2 and between v_2, v_3. Add some loops to the vertices in G' to make all vertices have the same out-degree and in-degree $3\Delta + 2$. See Fig. 3 for an example.

The graph G' naturally corresponds to a data movement problem with $n = 3|V_0|$ and $m = 3\Delta + 2$, where G' is its *transition graph*. (The transition graph is defined in Section II.) Finding a canonical block labelling with minimized parameter y for this data movement problem is equivalent to finding $t = n - y$ vertices – with the value of t maximized – in G',

$$a_1, a_2, \ldots, a_t,$$

such that for $i = 1, 2, \ldots, t-2$ and $j = i+2, i+3, \ldots, t$, there is no directed edge from a_j to a_i. We call such a set of t vertices – with t maximized – the MAXIMUM SEMI-INDEPENDENT SET of G'. For all $v \in V_0$, let $N(v)$ denote the neighbors of v in G_0.

CLAIM 1: "There is a maximum semi-independent set of G' where $\forall\ v \in V_0$, either all three corresponding vertices $v_1 \in V_1, v_2 \in V_2, v_3 \in V_3$ are in the set, or none of them is in the set. What is more, if v_1, v_2, v_3 are in the set, then no vertex in $\{w_1, w_2, w_3 | w \in N(v)\}$ is in the set."

To prove CLAIM 1, let $(a_1, a_2, \ldots, a_t)$ denote a maximum semi-independent set (*MSS*) of G'. (Note that the order of the vertices in the set matters.) Consider two cases:

Case 1: One of $\{v_1, v_2, v_3\}$ is in the MSS of G'. WLOG, say it is v_1. At most two vertices – say b and c – in $\{w_1, w_2, w_3 | w \in N(v)\}$ can be in the MSS, because otherwise due to the bi-directional edges between them and v_1, there would be no way to place them in the MSS. Let us remove b, c from the MSS and add v_2, v_3 right after v_1 in the MSS. It is simple to see that we get another MSS.

Case 2: Two of $\{v_1, v_2, v_3\}$ are in the MSS of G'. WLOG, say they are v_1 and v_2. At most one vertex – say b – in $\{w_1, w_2, w_3 | w \in N(v)\}$ can be in the MSS, for a similar reason as Case 1. In the MSS, let us remove b, move v_2 right behind v_1, and add v_3 right behind v_2. Again, we get an MSS.

So in this way, we can easily convert any MSS into an MSS satisfying the conditions in CLAIM 1. So CLAIM 1 is true.

CLAIM 2: "A set of vertices $\{w(1), w(2), \ldots, w(k)\}$ is a maximum independent set of G_0 if and only if the set of vertices $(w(1)_1, w(1)_2, w(1)_3, w(2)_1, w(2)_2, w(2)_3, \ldots, w(k)_1, w(k)_2, w(k)_3)$ is an MSS of G'." It is simple to see that this is a consequence of CLAIM 1.

So given a canonical labelling with minimized parameter y for the data movement problem with G' as the transition graph, in polynomial time we can convert it into an MSS of G', from that into an MSS of G' satisfying the conditions of CLAIM 1, and finally into a maximum independent set of G. So it is NP hard to find a canonical labelling with minimized parameter y. So minimizing the number of erasures is NP hard. ■

Since every algorithm uses at least $n + 1$ erasures, and Algorithm 9 can easily achieve $2n - 1$ erasures (by setting $y = n - 2$), the algorithm is a 2-approximation.

ACKNOWLEDGMENT

This work was supported in part by the NSF CAREER Award CCF-0747415, NSF grant ECCS-0802107, Caltech Lee Center for Advanced Networking, and the Center for Magnetic Recording Research at University of California, San Diego.

REFERENCES

[1] M. Ajtai, J. Komlós and E. Szemerédi, "An O(n log n) sorting network," in *Proc. ACM Symposium on Theory of Computing*, pp. 1–9, 1983.

[2] K.E. Batcher, "Sorting networks and their applications," in *Proceedings of the AFIPS Spring Joint Computer Conference*, pp. 307–314, 1968.

[3] B. Bollobas, *Modern Graph Theory*, Chapter 3, Springer, 2002.

[4] Y. Cassuto, M. Schwartz, V. Bohossian and J. Bruck, "Codes for multilevel flash memories: Correcting asymmetric limited-magnitude errors," *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2007, p. 1176-1180.

[5] H. Finucane, Z. Liu and M. Mitzenmacher, "Designing floating codes for expected performance," *Proc. 46th Annual Allerton Conference*, 2008.

[6] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," in *ACM Computing Surveys*, vol. 37, no. 2, pp. 138-163, June 2005.

[7] A. Jiang, V. Bohossian and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2007, pp. 1166-1170.

[8] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2008, pp. 1731-1735.

[9] A. Jiang, R. Mateescu, E. Yaakobi, J. Bruck, P. H. Siegel, A. Vardy and J. K. Wolf, "Storage coding for wear leveling in flash memories," Caltech Tech. Rep., online: $http://www.paradise.caltech.edu/etr.html$.

[10] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2008, pp. 1736-1740.

[11] E. Yaakobi, A. Vardy, P. H. Siegel and J. K. Wolf, "Multidimensional flash codes," *Proc. 46th Annual Allerton Conference*, 2008.