A Low-Complexity High-Performance Wear-Leveling Algorithm for Flash Memory System Design

Ching-Che Chung and Ning-Mi Hsueh
Department of Computer Science and Information Engineering
National Chung Cheng University
No. 168 University Rd., Min-Hsiung, Chia-Yi, Taiwan, R.O.C.
Email: wildwolf@cs.ccu.edu.tw

Abstract— **In this paper, a low-complexity high-performance wear-leveling algorithm which named sequential garbage collection (SGC) for flash memory system design is presented. The proposed SGC outperforms existing designs in terms of wear evenness and low design complexity. The lifetime of the entire flash memory can be greatly lengthened by the proposed SGC. In addition, the proposed SGC doesn't require any tuning threshold parameter, and thus it can be applied to various systems without prior knowledge of the system environment for threshold tuning. The low-complexity low-cost SGC makes it is easy to be implemented by firmware-based or hardware-based approaches. The simulation results show that the maximum block erase count and the standard deviation of the block erase count are decreased by up to 75% and 94%, respectively, as compared to the greedy algorithm.**

I. INTRODUCTION

In recent years, flash memory becomes an important storage system for many embedded system applications, such as smart phones, digital cameras, and solid-state disks (SSDs). Flash memory has lower power consumption, faster random access performance, and better shock resistance than the conventional hard disks, thus it can be applied as the cache of the hard disks or the fast booting devices for the operation system. However, it has some disadvantages such as the price, writing speed, and the most important is the endurance of the flash memory.

A flash memory chip contains many blocks, and each block is consisted of a fixed number of pages. The smallest unit for read and write operation is a page, but the smallest unit for erase operation is a block. The content of the flash memory cannot be overwritten; it must be erased before new data can be stored. However, erase operation usually takes several milliseconds. For performance consideration, updates to data are usually handled in an out-of-place fashion, rather than directly overwriting old data. Data that are to be updated should be written to another page of the flash memory, and the original page of the data is marked as an invalid page. The flash translation layer (FTL) is the core engine in the flash memory which maintains a mapping table between logically addresses and physical addresses on the flash memory chip. The FTL implementation can be carried out by either software on a host system or hardware/firmware on a flash memory system [1],[2],[4]-[7].

When data are written to the flash memory, the amount of free pages becomes low, and thus the garbage collection (GC) operation is performed to recycle the space occupied by invalid pages. The GC operation collects valid pages in the selected block, and copies these pages to another block, and then it erases the selected block. Unfortunately, each block typically has a lifetime of 10,000 times (multi-level cell, MLC) to 100,000 times (single-level cell, SLC) erase operations [10]. In addition, GC operation with greedy algorithm will select blocks that have the highest number of invalid pages [1]. Therefore, GC operation may wear out some blocks of the flash memory chip due to the localities of data access. For example, if some blocks stored hot data, they will be invoked by GC operation more frequently. These blocks may retire early, and this causes the reliability problem of the flash memory.

Therefore, there are many wear-leveling algorithms proposed in prior studies [3]-[9] to solve the problem of GC operation and extend the lifetime of the flash memory. The purpose of wear-leveling algorithm is to evenly distribute block erases over the flash memory, and thus the endurance of the flash memory can be enhanced. Wear-leveling algorithms can be classified into dynamic wear-leveling [3],[5],[9] and static wear-leveling [4],[6]-[8]. The performance of dynamic wear-leveling depends on hot data identification, and the cold data always stay at their blocks regardless of whether updating of hot data wear out the other blocks. As a result, the endurance improvement is strongly dependent on how many blocks are already occupied by cold data.

In conditional threshold wear-leveling algorithm [8], when a block is erased, the cold data in the block with minimum erase count is moved to the newly erased block to perform hot data and cold data swap. However, the newly erased block

This work was supported in part by the National Science Council of Taiwan, R.O.C., under Grant NSC100-2221-E-194-051.

doesn't guarantee to contain hot data, and thus this algorithm may perform unnecessary data movement. In addition, the threshold tuning will also influence the performance of wear-leveling.

In static wear-leveling [4],[6], when the difference between the maximum block erase count and minimum block erase count is larger than the specified threshold, it will moves the static data to the hot block to balance the erase count in each block. Therefore, a block erasing table (BET) is created to identify which block has been erased during a given time period. Then, the GC operation will pick up a block that has not been erased so far. This algorithm prevent cold data from staying at any block for a long time, and then the maximum block erase count difference between any two blocks can be minimized. However, the wear-leveling threshold for various system environments can be very different. Using inadequately tuned parameters can cause unexpectedly high wear-leveling overhead and worse performance of wear-leveling.

The implementation complexity of the wear-leveling algorithms determines the applicability of the algorithm. Existing wear-leveling algorithms require prior knowledge of the system environment for threshold tuning, and thus increase their design complexity. Therefore, in Lazy wear-leveling [7], an automatically on-line threshold tuning algorithm is proposed. The proposed algorithm monitors dynamic disk workloads to adjust the duty-cycle of wear-leveling, and thus the wear-leveling overhead and wear evenness can be trade-off.

All of the above wear-leveling algorithms [3]-[9] require a fixed or a dynamic threshold to control whether the wear-leveling algorithm is triggered. Thus unless the unbalance block erase count is larger than the specified threshold, the flash memory system still performs the normal GC operation. The wear-leveling can extend the lifetime of the flash memory chip, but it also has living pages copy overhead and read/write performance degradation.

In this paper, a low-complexity high-performance wear-leveling algorithm which named sequential garbage collection (SGC) for flash memory system design is presented. The proposed algorithm combines GC operation and wear-leveling into a single process. In addition, the proposed SGC doesn't require any tuning threshold parameter, and thus it can be applied to various systems without prior knowledge of the system environment. The low-complexity low-cost SGC algorithm makes it is easy to be implemented by firmware-based or hardware-based approaches.

The rest of the paper is organized as follows: the proposed SGC algorithm is presented in Section II. Section III shows the experimental results. Finally, Section IV concludes with a summary.

II. THE PROPOSED SGC ALGORITHM

In a flash memory system, the embedded micro-controller has very limited RAM space, and the hardware and computing resources are constricted. Thus the proposed SGC algorithm requires a low memory space and demands limited hardware resources. The basic idea to combine GC operation and wear-leveling into a single process is very simple that the blocks should be erased in a sequential manner.

Algorithm 1 shows the pseudo code of the proposed SGC1 algorithm. In SGC1 algorithm, *free_block* means the number of free blocks in the flash memory, and *index* is the index number of the block which erased last time. If the number of free blocks becomes one, the controller executes SGC1 algorithm and performs GC operation on the selected block. GC operation copies valid pages to the free block and erases the selected block.

Algorithm 1: SGC1 (Sequential Garbage Collection V1)

Input: *free_block, index*
Output: *null*

```
1: if free_block = 1 then
2:     index ←(index+1) MOD BlockSize;
       /* BlockSize is number of blocks in the flash */
3:     GarbageCollection(index);
       /*Request the garbage collection to erase the selected block and
         copy valid pages to the free block*/
4: end
```

The proposed SGC1 algorithm evenly distributes block erases over the entire flash memory. Thus it has greatest wear-leveling performance. However, the GC operation doesn't consider the number of invalid pages in the selected block, and therefore, the SGC1 algorithm may have many extra living pages copy in worse case conditions.

To reduce extra living page copy overhead of the SGC1 algorithm, a bit vector which named invalid page flag (IPF) is added. The IPF records whether a block contains more than 75% invalid pages. For example, if IPF[*index*] is equal to 1, then it means the block with index number: *index* contains more than 75% invalid pages. Oppositely, if IPF[*index*] is equal to 0, then it means the block has less than 75% invalid pages. The IPF vector can be updated during the write operation, and each block requires only one-bit flag. The GC operation will erase the blocks with IPF=1 with a higher priority.

Algorithm 2 shows the pseudo code of the proposed SGC2 algorithm. In SGC2 algorithm, *fcnt* is the number of "1" in the IPF vector. If *fcnt* is not zero, there are blocks having more than 75% invalid pages. In addition, *seq* and *index* are the index numbers for searching for the target block for current GC operation. If the number of free blocks becomes one, the controller executes SGC2 algorithm and performs GC operation on the selected block. In SGC2 algorithm, if *fcnt* is zero, there has no block with a higher priority. Then the blocks are erased in a sequential manner (step 2-6). Otherwise, if *fcnt* is not zero, there are blocks having more than 75% invalid pages. The SGC2 algorithm searches for the block with IPF=1 and then performs GC operation on the selected block (step 8-11).

The SGC1 algorithm erases blocks in a sequential manner but it may have many extra living pages copy overhead. The SGC2 algorithm uses the IPF vector to reduce the overhead in

the SGC1 algorithm, but it requires some clock cycles to search for high priority blocks. The proposed SGC1 and SGC2 algorithms evenly distributes block erases over the entire flash memory, and thus the lifetime of the entire flash memory can be greatly lengthened. The proposed SGC algorithm requires a low memory space and demands limited hardware resources, and thus it is easy to be implemented in the flash memory system.

Algorithm2 : SGC2 (Sequential Garbage Collection V2)

Input: *free_block, fcnt, seq, index, and IPF*
Output: *null*

```
 1: if free_block = 1 then
 2:     if fcnt = 0 then
 3:         seq ←(seq+1) MOD BlockSize;
            /* BlockSize is number of blocks in the flash*/
 4:         index ←seq;
 5:         GarbageCollection(index);
            /* Request the garbage collection to erase the selected
               block and copy valid pages to the free block*/
 6:     end
 7:     else
 8:         while IPF[index]=0 do
 9:             index ←(index+1) MOD BlockSize;
10:         end
            /* IPF=1 means more than 75% invalid pages
               in the block */
11:         GarbageCollection(index);
            /* Request the garbage collection to erase the selected
               block and copy valid pages to the free block*/
12:         index ←(index+1) MOD BlockSize;
13:     end
14: end
```

III. EXPERIMENTAL RESULTS

TABLE I. SETTING FOR THE SIMULATION.

Storage capacity	2GB MLC
Storage block number	4096
Page number per block	128
Per page size	4kb
Endurance	10k-5k
Page read time	60μs (2,400 cycles)
Page write time	800 μs (32,000 cycles)
Block erase time	1.5ms (60,000 cycle)

We build a cycle-accurate simulation environment with a 2GB flash memory [10] to verify the effectiveness of the proposed SGC algorithm. The simulation parameters are listed in Table I. As shown in Table I, the erase operation takes much longer clock cycles than the other operation.

For comparisons with existing wear-leveling algorithms, we rebuild the GC operation with greedy algorithm [1] and named as **GA**. In GA, GC operation will select blocks that have the highest number of invalid pages, and thus GA doesn't consider the wear-leveling problem. We also rebuild the static wear-leveling algorithms [4],[6] (with threshold=10) and named as **SW**. In the simulation, totally 120GB data are written to verify the performance of wear-leveling algorithms.

The maximum block erase count after writing 120GB data is shown in Fig. 1 (a). In GA, greedy algorithm performs GC operation on the blocks with highest number of invalid pages. Therefore it may often erase on the same block which stored the hot data. The maximum block erase count for GA is 632 in this simulation.

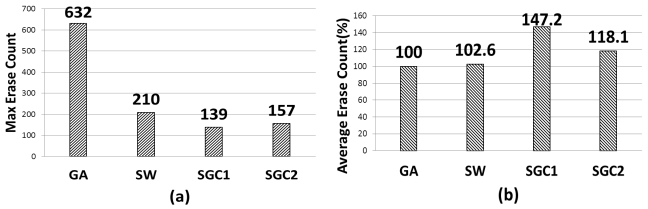

Figure 1. (a) maximum block erase count (b) normalized average erase count.

In SW, when the difference between the maximum block erase count and minimum block erase count is larger than the specified threshold, SW moves the static data to the hot block to balance the erase count in each block. The maximum block erase count for SW is 210 in this simulation, and the maximum block erase count is reduced as compared to GA.

The proposed SGC1 algorithm always erases blocks in a sequential manner, and therefore it can achieve the lowest maximum block erase count which is 139 in this simulation. However it may have many extra living pages copy overhead, and thus, the proposed SGC2 algorithm uses the IPF vector to reduce the overhead in the SGC1 algorithm. The maximum block erase count of SGC2 is 157, and is a little larger than SGC1. The proposed SGC algorithm can effective reduce the maximum block erase count, and thus the lifetime of the flash memory can be greatly lengthened.

Fig. 1 (b) shows the normalized average block erase count for four algorithms. SW still uses the greedy algorithm to perform GC operation, but it requires extra GC operations to balance the erase count in each block, thus the average erase count is larger than GA. In SGC1, the efficiency in recycling the space occupied by invalid pages is not as good as greedy algorithm, thus it requires many extra erase operations. Therefore, SGC2 uses the IPF vector to reduce the overhead in SGC1, and then the average erase count is greatly reduced.

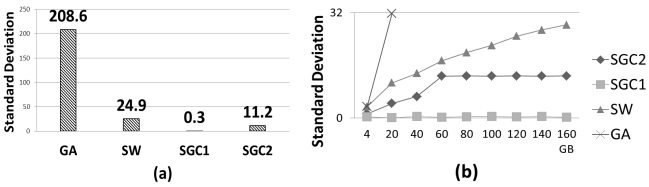

Figure 2. (a) standard deviation of block erase count (b) changes in standard deviations.

Fig. 2 (a) shows the standard deviation of block erase count in four algorithms. The SGC1 algorithm always erases

blocks in a sequential manner, and thus it can achieve a lowest standard deviation value. SGC2 uses the IPF vector to reduce the overhead in SGC1, and it also achieves a very small standard deviation as compared to SW. Fig. 2 (b) shows the changes of the standard deviation versus the amount of written data. As we can expect, the standard deviation in SGC1 will not have any changes, and SGC2 also achieves a very small standard deviation. However, in SW, the standard deviation is grown up when the amount of written data is increased.

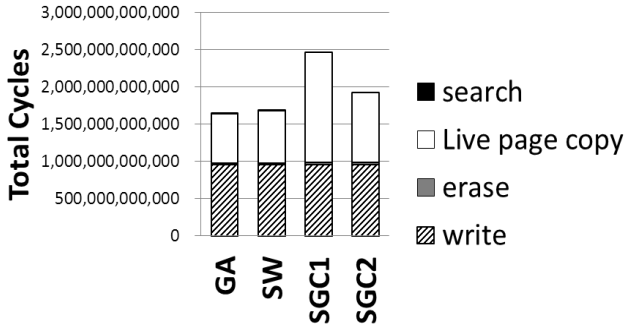

Figure 3. Total cycles for writting 120GB data.

TABLE II. PERFORMANCE COMPARISONS.

	GA	SW	SGC1	SGC2
Average erase count	93.9	96.3	138.1	110.8
Max erase count	632	210	139	157
Standard deviation	208.6	24.9	0.3	11.2
Normalized living pages copy (%)	100	106.6	217.6	143.0
Total cycles for writing 120GB data ($\times 10^9$)	1,661	1,707	2,471	1,958
Normalized total cycles for writing 120GB data (%)	100	102	148	117

Fig. 3 shows the total cycles to write 120GB data in four algorithms. In Fig. 3, the "write" means the total cycles taken in page write operations, and "erase" means the total cycles in erase operations. The "search" is the total cycles consumes in each algorithm to find the block to be erased. The "live page copy" means the total cycles used in living pages copy.

In SGC1, the efficiency in recycling the space occupied by invalid pages is not as good as greedy algorithm, thus it requires many extra erase operations and results in more cycles spent in living pages copy. SGC2 uses the IPF vector to reduce the overhead in SGC1, and it can reduce the total cycle time as compared to SGC1. Although the proposed SGC2 algorithm has longer execution cycles than SW for writing 120GB data, it is rare to continuously write so many data in a 2GB flash memory. As a result, the read and write speed degradation will not be so worse in a real application. The performance comparisons are summarized in Table II.

IV. CONCLUSION

In this paper, a low-complexity high-performance wear-leveling algorithm for flash memory system design is presented. The proposed SGC algorithm combines GC operation and wear-leveling into a single process, and it doesn't require any tuning threshold parameter. Simulation results show that the proposed SGC algorithm has the lowest maximum block erase count and smallest standard deviation. In addition, the low-complexity low-cost SGC algorithm makes it is easy to be implemented by firmware-based or hardware-based approaches. Thus the lifetime of the flash memory can be greatly lengthened by the proposed SGC algorithm.

ACKNOWLEDGMENT

The authors would like to thank their colleagues in S3 Lab of National Chung Cheng University for many fruitful discussions. The process technology files supported by UMC are acknowledged as well.

REFERENCES

[1] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda, "A flash-memory based file system, " *in Proceedings of USENIX Annual Technical Conference*, Jun. 1995, pp. 155-164.

[2] Mendel Rosenblum and John K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computers Systems (TOCS)*, vol. 10, no. 1, pp. 26-52, Feb. 1992.

[3] Li-Pin Chang, "On efficient wear leveling for large-scale flash-memory storage systems," *in Proceedings of ACM Symposium on Applied Computing (SAC)*, Mar. 2007, pp. 1126-1130.

[4] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo, "Improving flash wear-leveling by proactively moving static data, " *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 53-65, Jan. 2010.

[5] Lin-Pin Chang and Tei-Wei Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation, " *ACM Transactions on Storage (TOS)*, vol. 1, no. 4, pp. 381-418, Nov. 2005.

[6] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo, "Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design," *in Proceedings of ACM/IEEE Design Automation Conference (DAC)*, Jun. 2007, pp. 212-217.

[7] Li-Pin Chang and Li-Chun Huang, "A low-cost wear-leveling algorithm for block-mapping solid-state disks," *in Proceedings of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools and Theory for Embedded Systems (LCTES)*, Apr. 2011, pp. 31-40.

[8] Wen-Kai Hsieh and Hsi-Pin Ma, "Conditional threshold wear-leveling algorithm for multi-channel NAND flash memory, " *in Proceedings of International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*, Apr. 2010, pp. 147-150.

[9] Kee-Hoon Jang and Tae-Hee Han, "Efficient garbage collection policy and block management method for NAND flash memory," *in Proceedings of International Conference on Mechanical and Electronics Engineering (ICMEE)*, Aug. 2010, pp. 327-331.

[10] Samsung Electronics, "K9GAG08U0M 2G×8bit NAND Flash Memory Data Sheet, " 2006.