A Technique to Improve Garbage Collection Performance for NAND Flash-based Storage Systems

Jaehyeong Jeong, and Yong Ho Song, *Member*, IEEE

Abstract — *Garbage collection (GC) is a time-consuming operation incorporated in flash memory storage systems, and thus, it has a significant impact on the storage performance. Flash memory provides an internal copy-back operation that can perform page data transfer efficiently during GC. However, this operation is not widely accepted due to some limitations: bit errors can be propagated during the transfer between the pages without being detected, and the operation can be used for blocks within the same plane. In order to address these problems, the page data must be read to a buffer and then transmitted to another page. When this method is used, not only does a temporal overhead occur but also I/O bus utilization and power consumption by the two data transfer operations are increased.*

In this paper, we propose a technique for reducing the data transfer overhead of GC, using which devices transmit data directly when garbage is collected in a flash memory-based storage system in which several flash memory devices share a single I/O bus, while only the partial updated corrected data is updated when an error is detected. The proposed technique improves the performance of GC and reduces I/O bus utilization compared to the existing methods. [1]

Index Terms —Garbage Collection, Copy-back, NAND Flash Memory, SSD (Solid-State Drive)

I. INTRODUCTION

NAND flash memory has many advantages including shock resistance, low power consumption, and lightweight construction; hence, it is widely used as the data storage system in many consumer products such as thin-and-light laptops, tablet PCs, and smartphones. However, it also has drawbacks such as asymmetric operation unit size, no support for in-place update (also called *erase-before-update*), limited erase/write (E/W) cycles, etc. In order to circumvent these limitations, the flash storage controller adopts a special software layer, called the *Flash Translation Layer* (FTL), responsible for address-mapping, wear-leveling, and other management tasks [1], [2].

In order to overcome the limitation exerted by erase-before-update, FTL updates existing page data out of place. That is, it stores an update to the old data in an empty flash page, updates the mapping information reflecting the new location, and invalidates the old page data. This approach helps to temporarily avoid a time-consuming erase operation during updating of the data in the storage system, hence contributing to an improved write performance. However, it is possible for invalidated data to occupy flash storage, which then results in a decrease in free available space for further incoming data.

When the amount of free space drops below a threshold due to the increase in the number of invalidated pages, a garbage collection operation begins to recycle the flash pages that are being occupied by these invalidated pages. This operation increases the number of usable free pages by erasing a flash block, called a *victim block*, with invalidated pages. In order to preserve any valid pages remaining in the victim block, the GC copies these page data to other empty pages before performing an erase on the victim block. Therefore, one GC consists of page copy operations (also called *merge operations*) and a block erase operation.

Some flash memory systems provide the *internal copy-back operation (ICB)* that is used to transfer a data page between source and destination blocks in the same device without using the I/O bus. This operation is effective because the page data move internally. However, ICB is becoming obsolete because of several limitations. First, when there are bit errors in the source page, they can be carried over to the destination because no error correction mechanism is incorporated in the device. Second, even when it is provided, this operation often requires both source and destination pages to be on the same plane of a device. Third, ICBs, if used, require that program and erase operations take place on the same plane of a device, which may cause an unbalanced E/W cycle increase over blocks in the flash storage. Therefore, a merge operation tends to read out page data from the source to the controller buffer, correct bits errors, and write the page data to the destination; this is called *external copy-back (ECB)*. The two data transfer operations that occur in ECB not only increase the delay in the on-going GC, but also increase I/O bus utilization and thus power consumption.

GC greatly affects the performance of flash memory storage due to its long latency. While GC is in progress, it blocks new accesses to the storage at the flash channel or at the device. In some cases, it may cause a very long and unpredictable latency that delays new outstanding accesses from the host system.

Many studies have been performed in an attempt to reduce the overheads of GC. Some approaches include a reduction in the frequency of GC, cost-effective criteria for selecting victim blocks, etc. [2], [3]. However, only a few studies on the reduction of merge cost during GC have been published.

[1] This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0017147).

Jaehyeong Jeong and Yong Ho Song are with the Department of Electronics and Computer Engineering, Hanyang University, Seoul, Korea (e-mail: {jhjeong, yhsong}@enc.hanyang.ac.kr).

Contributed Paper
Manuscript received 04/15/12
Current version published 06/22/12
Electronic version published 06/22/12.

This paper proposes a storage control scheme that can be used to improve GC performance in flash storage systems. The scheme is based on a *direct data transfer (DDT)* technique between the flash devices in the same channel. The motivation of this technique is the fact that the flash devices in the same channel share the I/O data path. That is, when a flash controller reads out page data from the source device through the I/O data path, the destination device is able to sense the data from the data path. If the controller issues a proper program command to the destination prior to the data read-out, the destination device can absorb the data into its data register. The DDT technique effectively decreases the number of data transfers during GC. Now, there are two implementation problems regarding this technique: first, *orchestrating the two devices to implement the DDT-based GC*, and second, *handling the case when the read-out data contain bit errors*. The solutions to these problems are addressed in the proposed control scheme. The experimental results show that the proposed scheme improves the performance of GC and reduces I/O bus utilization compared to the traditional external copy-back operation.

The remainder of this paper is organized as follows. Chapter 2 introduces the internal organization and operations of flash memory as well as flash memory storage architecture. Chapter 3 describes three copy-back schemes and the DDT technique. The design and implementation issues of the proposed technique are presented in Chapter 4. Chapter 5 and 6 evaluate the performance of the proposed scheme analytically and empirically, respectively. Finally, Chapter 7 concludes the paper.

II. BACKGROUND

A. Flash Memory Organization

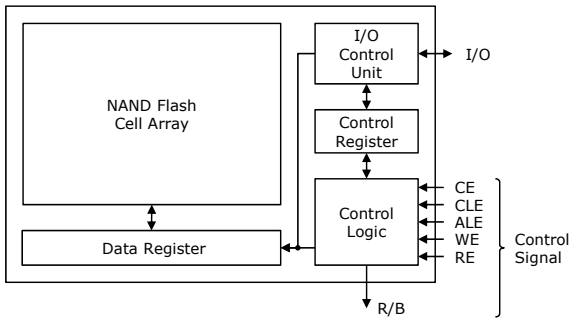


Fig. 1. NAND Flash Memory Block Diagram

The NAND flash memory chip consists of the *NAND flash cell array* (cell array), *data register*, *control logic*, *I/O control unit*, and *control register*, as shown in Fig. 1. The cell array constitutes the data storage area, while the data register is the I/O buffer for the cell array, via which data is transferred. There are two types of interface signals in flash memory: I/O signals and control signals. The I/O signals are used to deliver page data from/to the flash memory, while the control signals are for orchestrating the control logic (the control unit of flash memory) and setting the interface mode of flash memory. The control registers are needed for flash operations, along with the address register, command register, and status register.

B. Flash Memory Operations

1) Read and Program Operation

Read operation retrieves page data from the cell array to the outside via its data register, while *program operation* transfers page data from the outside to the cell array. Flash memory operations are divided into three stages: the command/address stage, the operation busy stage, and the data transfer stage. The operation busy state incurred by a read operation is referred to as *read busy*, while that incurred by a program operation is referred to as *program busy*. The program busy stage starts only after a confirmation command is issued to the device. Before this command, the page data for programming resides in the data register, and it can be updated with the location and data value.

2) Copy-back Operation

Copy-back operation is used to copy data from one page to another within the same plane through the data register. This operation consists of a page read and a page program, but without transferring the page data between the device and an external buffer. In this paper, the built-in copy-back operation is called *internal copy-back*.

3) Cache Program Operation

Cache program operation is used to allow page data to be loaded into flash memory during the busy period of the on-going operation. The data is temporarily stored in the I/O buffer of flash memory, called the *cache register*. This operation effectively hides the data transfer overhead from the program operation [12]. Fig. 2 shows the timing diagram for the program operation and the cache program operation. This figure shows that the overlapped loading of the page data contributes to decreasing the time taken to program multiple pages. However, the cache program can be used only within the same block. If the GC moves the pages inside the block, then this limitation may cause no problem.

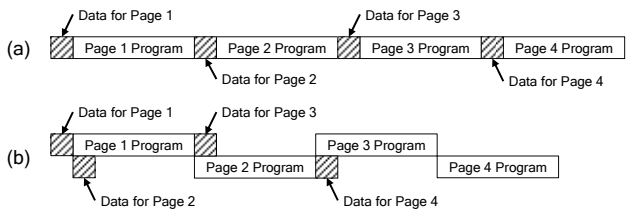


Fig. 2. Comparison of Flash Program Operations: (a) Normal Program Operation; (b) Cache Program Operation

Flash memory also supports the cache read operation. This paper, however, mainly discusses the benefit of using the cache write operation, because the cache read should be used to read consecutive pages. This is because the victim blocks for GC are assumed to have invalid pages so that the cache read cannot be used.

C. Flash Memory Storage Systems

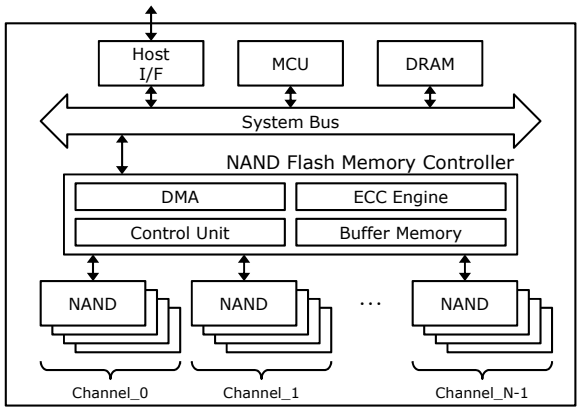


Fig. 3. NAND Flash Memory Storage Systems

A flash memory storage system consists of the *MCU*, *flash memory device array*, *flash memory controller*, and *DRAM buffer*, as shown in Fig. 3. Each I/O bus is shared among multiple flash memories to achieve more efficient utilization. Each flash device constitutes a *way* assuming that there is only one die and one plane in the device. When multiple flash memories are configured to share the I/O bus, it is called *multi-way* configuration. The shared I/O bus is called the *channel*. In the channel organization, while one flash memory device is in the operation busy stage, another device sharing the channel can perform its own flash memory operation in an overlapped way using the so-called *interleaving technique* in order to improve the storage performance [4]-[7]. In this study, for the sake of simplicity, a single flash memory device is assumed to have one way.

The flash memory controller is responsible for orchestrating the internal operations of storage systems. It has the *DMAC* (Direct Memory Access Controller), *control unit*, *ECC* (Error Correction Code) *engine*, and *buffer memory*. The control unit asserts the control signals necessary for the read/write/erase operation in flash memory devices. The ECC engine performs error detection and correction while the page data is transferred to the buffer memory.

III. Copy-back Schemes for Fast Garbage Collection

A. Global Copy Back

GC uses ECB operations to copy page data. In this study, ECBs are classified into two types, depending on the location of the source and destination blocks: *local copy-back (LCB)*, which is used when the source and destination block are located in the same device, and *global copy-back (GCB)*, otherwise. Fig. 4 illustrates the ICB and ECB operations. In this figure, *Wx* refers to a device (way) within the same channel. Furthermore, *Data Out* and *Data In* refer to the time required to perform data output and input, respectively. Similarly, *Read Busy* and *Program Busy* refer to internal flash memory operations (command and address cycles were omitted because the execution time is relatively short). Fig. 4(a) and (b) show the case when both the source and destination blocks are in *W0*. In Fig. 4(c), however, the source block is in *W0*, while the destination is in *W1*. During ECBs, the time to transfer the page data out of the device is called *Data Out*. Likewise, the time to transfer the buffer data into the device is called *Data In*.

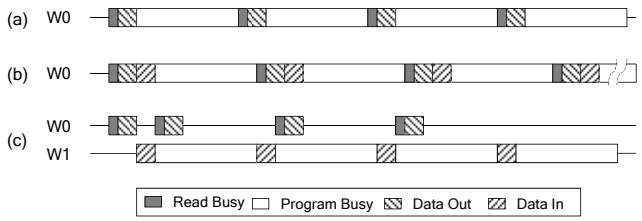


Fig. 4. Copy-back operations: (a) ICB; (b) external LCB; (c) external GCB

As mentioned earlier, the ICB requires that the source and destination blocks reside in the same plane and device. The next operation can take place only after the on-going operation has completely finished. When ICB is performed, Data Out is required to check the existence of bit errors. If there are no errors, the page program can start with the data in the data register. LCB is a copy-back between different planes in the same device. Unlike ICB, it requires that the page data be loaded into a data register for programming. Therefore, when LCB is used, merge operations experience a longer latency due to the increased data transfer overhead. In GCB, the page read operation can be overlapped with the program busy period, except for that on the first page.

The GCB has an advantage in that it creates free blocks earlier. Considering that the purpose of GC is to clean victim blocks to create free blocks, early generation of free blocks would contribute to performance enhancement. More specifically, when ICB or LCB is used, an erase operation can start after the completion of all merge operations. However, in GCB, an erase operation can be performed immediately upon the completion of a read operation on the last valid page.

B. Resource Conflicts during Garbage Collection

In flash memory storages, GCs directly affect the overall system performance, because the storage resources are engaged by GC, which temporarily blocks (or delays) the next request from the host system.

1) Conflict with flash memory

When a flash memory device is involved in GC, it is not able to accept any more requests. As a result, the responsiveness of the storage may temporarily suffer. This problem can be remedied partially by dividing GC into a set of read/program operations [8]. That is, instead of merging all the pages continuously, GC merges a smaller number of pages at a time. While this may increase the overall execution time, the cache operation can be used to improve the performance of individual merge operations.

Fig. 5 shows the merge operation using cache operation. In GCB, the data read from *W0* go through the error detection/correction stage, and then move to *W1*. By using the cache operation and overlapping data transfers and program operations for *W1*, the performance of GCB is improved.

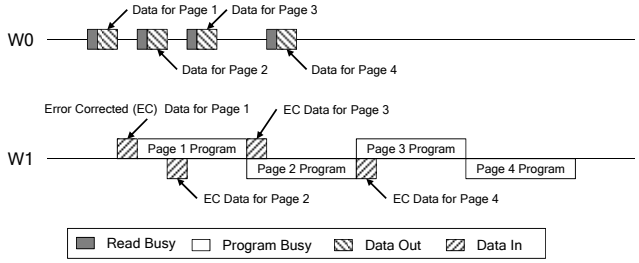


Fig. 5. GCB with Cache Operation

2) Conflict on Shared Channel

While one flash memory device is using the channel, other flash memory devices that share the bus cannot read or write commands/data. However, as GC causes frequent transfer of page data, channel utilization is increased. As a result, other devices in the channel unrelated to the GC will not be able to accept new requests, causing a delay. This is because, even if such devices are idle, they cannot gain control of the channel and therefore cannot initiate any operations. This problem occurs even if GCB is used, because the channel is heavily utilized during GCB. Using cache operation cannot be a fundamental solution to this problem, because it does not reduce channel utilization.

One possible solution is to increase the data transfer speed. More recent types of synchronous flash memory (toggle flash memory) have short data transfer times, and they can therefore be used to increase the data transfer speed [10]. Storages that use these types of flash memory, however, have it so that numerous flash memory devices share the channel, in order to hide the long delays of individual devices. For such a structure, this solution would not succeed because, even if channel utilizations by individual devices are low, there are many devices that share the channel.

C. Global Copy Back using Direct Data Transfer

ICB and LCB are limited in terms of execution speeds and operation range. Although GCB is effective in improving the execution speed, it results in high channel utilization, which increases the blocking effect on the channel.

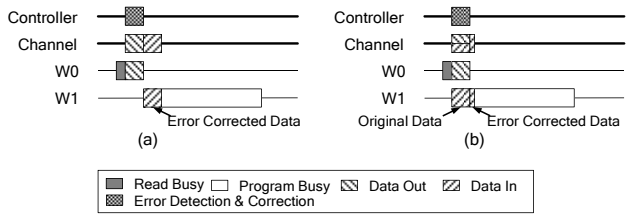


Fig. 6. Comparison of GCB and DGCB (a) GCB (b) DGCB

In order to reduce data transfer overheads, the technique proposed in this paper uses DDT between flash devices during GCB: when page data is transferred via the I/O channel, it is also delivered to the destination device attached to the same channel. That is, Data Out and Data In times are completely overlapped. The GCB using DDT is called *DDT-based GCB (DGCB)* in this paper. Note that there may be bit errors in the transferred data. If bit errors exist, they have to be corrected, and an additional data transfer should take place for the delivery of the corrected data.

Fig. 6 shows the timing diagrams for GCB and DGCB, where *W0* and *W1* are the source and destination ways, respectively. In Fig. 6 (a), the page data is transferred from *W0* to the controller, and the error detection/correction operation runs at the same time. The data is then transferred back to *W1*. In Fig. 6 (b), however, the page data move from *W0* to both *W1* and the controller at the same time. If any bit errors exist, they are corrected. Later, the corrected data is supplied to *W1* using an additional data transfer. Considering that the maximum of bit errors allowed is much smaller than the page size, this additional data transfer takes only a fraction of the page transfer time. Afterward, a confirmation command is issued to *W1* in order to initiate the program operation. If the additional data transfer is not required, which would be the common case, the channel utilization of DGCB is half that of GCB. Obviously, the overhead of corrected data transmission in DGCB is proportional to the number of bit errors. Although the likelihood of an error occurring varies depending on the flash memory characteristics and E/W cycles, the actual number of bits of errors is expected to be small [11].

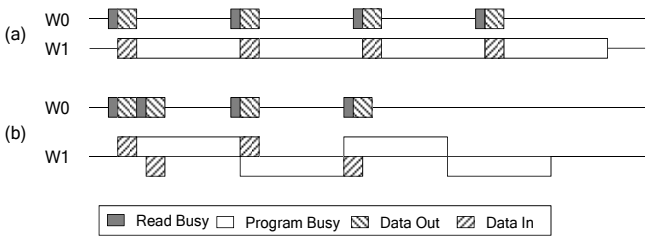


Fig. 7. DGCB Operations (a) without Cache Operation, and (b) with Cache Operation

Fig. 7 illustrates the impact of cache operations on DGCB. Fig. 7(a) shows the DGCB operation without utilizing the built-in cache operation. In this case, DGCB looks similar to the ICB shown in Fig. 2 in that there is only one data transfer out of the source device. However, compared to ICB, free blocks can be created earlier, because the *W0* can be erased immediately after the last page read. Compared to GCB, DGCB requires a smaller channel bandwidth, but releases the source block for erasure slightly later. This is because in DGCB, the last page read should take place after the previous page is programmed, while in GCB the read can be overlapped with the program busy at *W1*.

When cache operation is used (see Fig. 7(b) and Fig. 3), it is expected that the execution times of GCB and DGCB will be nearly the same; the only difference occurs when the first page is copied back. In GCB, there is no overlap with the read operation on the first page, while in DGCB the Data Out and Data In operations are performed at the same time.

IV. CONTROLLER DESIGN SUPPORTING DGCB

A. Control Unit Architecture

Fig. 8 shows the control unit architecture of the flash memory controller. The control unit has one *channel controller* per channel; the channel controller has a *way arbiter* and *way controllers*. The flash memory device is driven by a way controller, and each way controller operates independently. The purpose of the way arbiter is to choose one of the requests (*REQ*) for channel use, and issue a grant (*GRANT*) to the selected requestor.

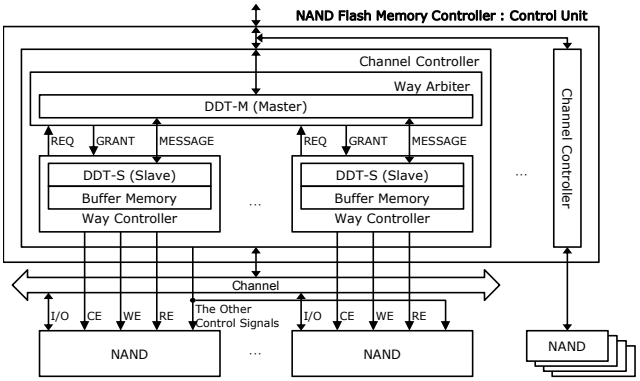


Fig. 8. The Control Unit Architecture

In order to implement the DGCB's capability, the way controller for the source and for the destination devices should be synchronized during data transfer. For instance, before data is read from the source, the destination should be ready to accept the data. In addition, before the page data is delivered to the destination, the source should be prepared to send the data. When the two controllers are well synchronized, it is possible to minimize the delay time incurred by the two operations and use the channels efficiently.

TABLE I
TYPES OF MESSAGE

Message	Description
READ_INIT	Initialization of read operation
READ_START	Start of read busy
READ_END	End of read busy (Ready for data output)
PROG_INIT	Initialization of program operation
DIN_READY	Ready for data input
ERR_RETRANSMIT	Error detected in read data (Corrected data retransmit)
PROG_START	Start program busy (Confirmation command input)
PROG_END	End program operation (Program operation completion)
DDT_BEGIN	Start of DDT
DDT_DONE	End of DDT (Read operation completion)

In the presented architecture, the *DDT master* (*DDT-M)* is responsible for synchronizing the operations of *DDT slaves* (*DDT-S).* In order to implement the synchronization mechanism, DDT-M and DDT-S communicate using the signal-based messages shown in Table 1. No direct communication is allowed among DDT-Ss.

Fig. 9 shows the communication sequence used by DDT-M and DDT-S to synchronize by exchanging the messages. DDT-M instructs a DDT-S to initiate a read operation through *READ_INIT*. The source issues a read command to the corresponding flash memory, which then enters into Read Busy. The DDT-S sends back *READ_START* to DDT-M to indicate the beginning of page read. DDT-M orders the destination device to prepare a program operation by sending *PROG_INIT*.

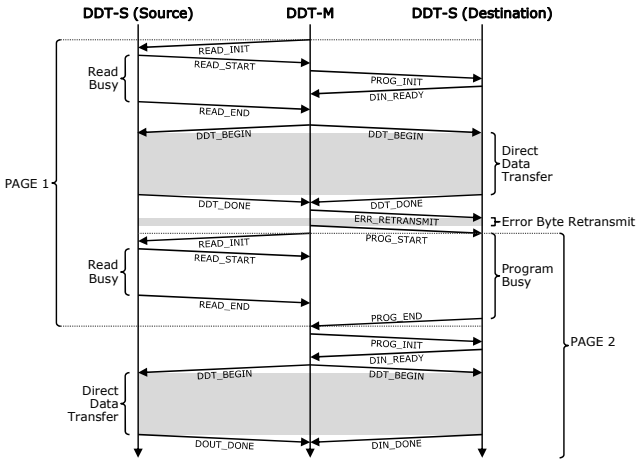


Fig. 9. Message Communication Sequence

When DDT-M has received both *READ_END* and *DIN_READY* from the source and the destination, it sends *DDT_BEGIN* to the source and destination at the same time to request them to start DDT. When DDT finishes, they send *DDT_DONE* to DDT-M. Then DDT-M checks if the ECC engine detects any error. If there is an error, DDT-M sends *ERR_RETRANSMIT* to the destination. When the destination receives this message, it waits for the corrected data being delivered. If an error is not detected, this message will be omitted. Now DDT-M sends *PROG_START* to the destination to start the program operation. During the Program Busy, the source reads the next page.

B. Control Signals and Timings

DDT-S is responsible for asserting the necessary signals to implement DDT. The interface mode of the flash memory is determined by a combination of control signals [13]. Each way controller should be able to drive the proper signals.

In general, most of the control lines are shared among the devices in the channel, except *CE (Chip Enable)* [4]-[7]. These include *CLE (Command Latch Enable)* and *ALE (Address Latch Enable)*, which are used for command and address input, respectively. However, for the implementation of DGCB, some of signals, that is, *WE (Write Enable)* and *RE (Read Enable)*, should not be shared among the devices. This is because they are asserted at the same time, but to different devices.

Fig. 10 is a timing diagram of the control signals and data path during the DDT operation. As can be seen in this figure, different way controllers independently drive two signals, RE and WE, while I/O is a data path shared by the source and the destination. *DVW* (Data Valid Window) is a time window during which read data is valid, while *DRW* (Data Required Window) is another time window during which the data should be valid at the destination for the correct program operation.

The size of each window is determined by the flash memory's timing parameters. The size for DVW is determined by *tRC*, *tRP*, *tREH*, and *tREA*, while the size for DRW is determined by *tDS* and *tDH* [13]. In fact, the DRW size should be equal to or greater than the sum of *tDS* and *tDH*. The read and write cycle times are represented as *tRC* and *tWC*, and they are assumed to be the same for the sake of convenience.

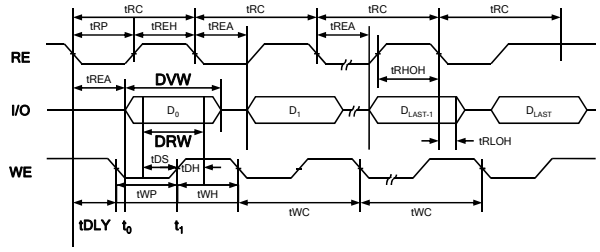


Fig. 10. Data Input and Output Cycles tRC: RE cycle time, tRP: RE pulse width, tREH: RE hold time, tREA: RE access time, tRHOH: RE high to output hold, tRLOH: RE low to output hold, tWC: WE cycle time, tWP: WE pulse width, tWH: WE high hold time, tDS: data setup time, tDH: data hold time

In order for DDT to work correctly, DVW should be equal to or larger than DRW. Otherwise, DVW needs to be widened by increasing tRC, which is not desirable due to its side effect of increasing data transfer time. In Fig. 10, t_0 is the time when the DVW starts, and (t_1 - t_0) constitutes a setup time for the next write operation. Therefore, t_1 is the earliest time when the WE signal can be asserted. In addition, *tDLY* is the time skew between the WE and RE signals. This skew is required to ensure the correct operation of the write. When a control unit is designed, it is important to choose the correct amount of time skew. One of the conditions that the tDLY should satisfy is that the time of (*tDLY* + *tWP*) is equal to that of (*tREA* + *tDS*). Since the manufacturers determine tWP and tDS, *tREA*, reflecting channel characteristics, is a key component that is used for choosing the proper *tDLY*.

V. Performance Analysis

In order to scrutinize the performance benefit of GCB and DGCB, the read and program operations were divided into the command/address stage, operation busy stage, and data transfer stage. The time taken by the command and address stage is very short compared to that of the entire operation time. However, it is still included in the equations just for evaluation accuracy. It is assumed that tRC, tWC and tRWC are the same in the following discussion without challenging the correctness of the analysis.

$$tDO = pageSize \times tRC \quad (1)$$

$$tDI = pageSize \times tWC \quad (2)$$

$$tDIO = pageSize \times tRWC + tDLY + tDI_{EC} \quad (3)$$

$$tDIO_{GAIN} = (tDI + tDO) - tDIO \quad (4)$$

The execution time of data transfer stages is shown in (1) to (3). The data transfer time is dominated by the page size and the flash memory cycle time. Equations (1) and (2) describe the time taken for data input and output, respectively, using GCB, while (3) describes the time consumed by DDT using DGCB. In (3), tDI_{EC} refers to the time it takes to retransmit corrected data. Equation (4) shows the $tDIO_{GAIN}$ is the performance gain that can be obtained for DGCB from DDT. However, the $tDIO_{GAIN}$ is almost the same as tDI or tDO, because *tDLY* and tDI_{EC} are very small in many cases.

$$tSR = tSR_{SET} + tR + tDO \quad (5)$$

$$tSP = tSP_{SET} + tDI + tPROG \quad (6)$$

$$tSR_{EXC.D} = tSR - tDO \quad (7)$$

$$tSP_{EXC.D} = tSP - tDI \quad (8)$$

Flash memory operations may experience internal delays caused by switching of the internal data path between stages. Specifically, *tADL* is the delay that occurs between the command/address stage and data transfer stage in a program operation, while *tWB* is that between the command stage and the operation busy stage. tSR_{SET} and tSP_{SET} are the sum of the command/address stage time and such internal delays, for the read and program operation, respectively. Likewise, *tR* and *tPROG* are the operation busy stage for the read and program operation. Equations (5) and (6) describe the overall execution time for the read and program operation, respectively. In addition, (7) and (8) present the overall execution time, excepting the data transfer.

$$tICB_N = (tSR + tSP_{EXC.D} + tDI_{EC}) \times N \quad (9)$$

$$tGCB_N = tSR + tSP \times N \quad (10)$$

$$tDGCB_N = tSR_{EXC.D} + (tDIO + tSP_{EXC.D}) \times N \quad (11)$$

Equations (9) to (11) calculate the execution time for three copy-back methods for merging *N* pages in a GC. First, (9) is the execution time of ICB. As already explained, ICB performs read and program operation inside the memory. Second, (10) is the execution time of GCB. Although two data transfer stages are included in GCB, the page read operation could be overlapped with the program busy of its previous page, if more than two pages are merged consecutively. As a result, the execution time increases by tSP for every additional page merged. Lastly, the execution time of DGCB is similar to that of ICB. This is because the data transfer time for a merge is the same in both methods. Note that tDIO includes tDI_{EC}. In fact, DGCB performs slightly better than ICB, because tSR_{SET} and tR of the current page are overlapped with program busy of the previous page in DGCB.

From (9) to (11), it is can be seen that ICB, GCB, and DGCB are expected to perform similarly. The difference in the performance of these methods is mainly due to the data transfer time of the first page: the transfer of the first page cannot be hidden in GCB.

$$tGCB_{N\_cache} = tSR + tSP_{SET} + tDI + tPROG \times N \quad (12)$$

$$tDGCB_{N\_cache} = tSR_{EXC.D} + tDIO + tSP_{SET} + tPROG \times N \quad (13)$$

Equations (12) and (13) explain the execution time of GCB and DGCB for the case when cache operations are used. As already known, ICB has no way to utilize the cache mechanism of flash memory. When the cache operation is used, not only is the read operation overlapped, but also the data input of program operation is overlapped with program busy of the previous page program operation. Thus, the data transfer time disappears from (12) and (13). From the equations above, the performances of GCB and DGCB are expected to be the same, because tPROG is the biggest component.

In order to improve the flash memory performance, a flash memory device may have multiple flash memory planes integrated in a device. This kind of flash memory architecture is referred to as *multi-plane architecture* [12]. One of the architecture's benefits is that multiple planes can simultaneously perform the same flash memory operation, as they can be controlled with a single command. Consequently, the performance of the flash memory can be improved. However, as this architecture requires separate data transfers for different planes, the data transfer time is longer than that of single plane operation. As a result, the data transfer time occupies a sizable portion of the execution time. If GCs were performed using multi-plane operation, the amount of data I/O would be increased and merge operations would be faster compared to single plane operation. When a multi-plane is mentioned in this paper, it is assumed that the flash memory is organized with two planes. The equations for multi-plane operation are omitted due to the page limit. However, they are very similar to (10)-(13), except for the length of data transfer time.

VI. EXPERIMENTS

A. Experiments Setup

In order to measure the impact on GC performance of three copy-back approaches, we performed simulation experiments using FlashSim [9], which is an event-driven simulator for flash memory based storage.

TABLE II
NAND FLASH CHARACTERISTICS

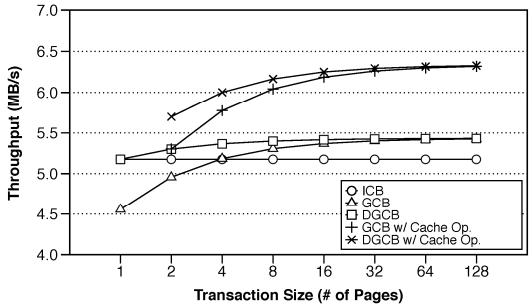
Parameter	Symbol	Time	Unit
Read Page operation time	tR	75	μs
Program Page operation time	tPROG	1300	μs
Erase Block operation time	tBERS	3	ms

Table 2 shows the characteristics of multi-level cell flash memory. The flash memory cycle time, tRC and tWC, was set to 25ns, and tRWC and tDLY were assumed to be 25ns and 20ns, respectively. The data page size was 8KB, and the size of the spare area was 448 bytes. In addition, as a 24-bit ECC is required for 1KB data, the minimum size of ECC needed per page was 192 bits.

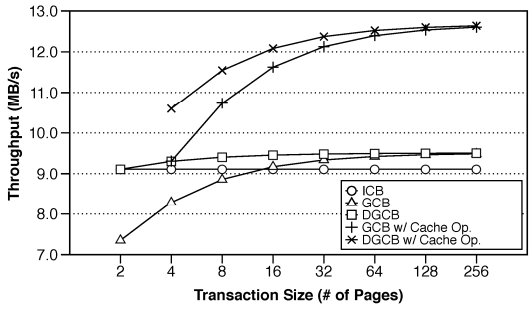
As mentioned earlier, the delay caused by GC reduces both the storage performance and responsiveness. In many real-time systems that have a narrow margin for storage delays, the GC overhead should be minimized. One of the possible ways to achieve this is to reduce the *transaction size*, that is, the number of pages involved in a GC [8]. Furthermore, in ordinary systems too, it is more desirable, in terms of storage efficiency, for GC to be performed in a smaller grain. If the size of each transaction is small, multiple transactions should be repetitively performed.

We assumed that the number of pages per flash memory block was 128. To measure the performance of GC based on transaction size, we assumed that it can be done via multiple transactions. In other words, the transaction size can be varied, which changes the number of transactions to be performed. For single plane merge operations, the transaction size was varied from 1 to 128 pages. In addition, for multi-plane merge operations, the transaction size was varied from 2 to 256 pages.

B. Throughput of Copy-backs



(a) Single Plane



(b) Multi-plane

Fig. 11. Throughput Comparison

Fig. 11 shows the throughput of merge operations when ICB, GCB, and DGCB schemes were used. In this result, the retransmission overhead incurred by error correction was excluded for ICB and DGCB. In addition, the merge throughput was measured under the assumption that cache operations were used. This is because the cache operation can be used when two or more page programs are performed consecutively. For ICB, operations did not overlap but were performed in due sequence, resulting in the same throughput regardless of the transaction size. For GCB and DGCB, however, there were overlapping effects from interleaving and the cache operation, and the throughput increased as the transaction size grew.

The GC performance of GCB was lower than of the other copy-backs when the transaction size was smaller than 8 pages in single plane, and 32 in multi-plane. This is because in the copy-back of the first page, data transfer does not overlap with other operations. This occurs in the first page merge of every single transaction whether or not cache operation is

used. Therefore, as the transaction size decreases, the throughput drops consequently.

GCB performed better than ICB when the transaction size was eight pages or more for single plane copy-back. For multi-plane copy-back, the same was true when the transaction size was 32 pages or more. However, when the cache operation was used, GCB performed better than ICB regardless of the transaction size. This is because, for GCB, there are gains in performance starting with the second page merge due to the benefits of cache operation, as explained.

DGCB outperformed both ICB and GCB regardless of the transaction size. When the cache operation was not used, the performance of DGCB was slightly better than that of ICB. When the transaction was small, DGCB outperformed GCB by a considerable margin, but the larger the transaction size, the narrower became the performance gap between the two. This is because the performance of GCB decreased when the transaction size was small, due to the effects of the data transfer overhead that occurs in the first page merge on the overall merge operation. As the transaction size increased, however, the effects of data transfer overhead became insignificant in relation to the total execution time, so that the performances of GCB and DGCB were similar. This shows that GCB is not suitable for use with real-time systems where small-sized transactions have to be repetitively performed to achieve storage responsiveness. Similar results were obtained for the multi-plane operation.

C. Overall Execution time of Garbage Collection

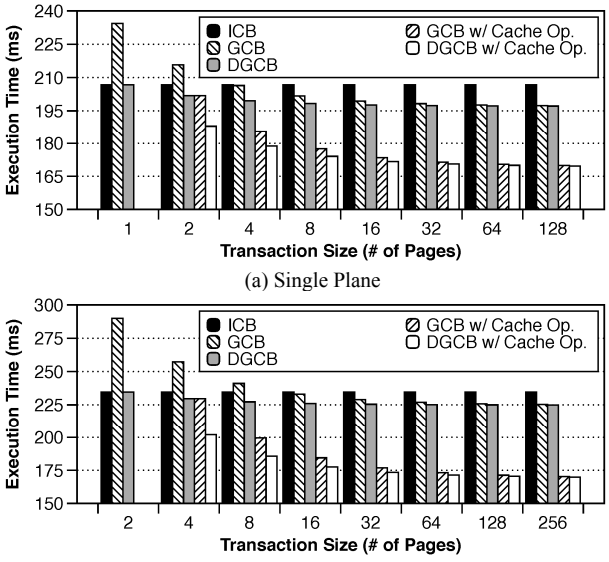


Fig. 12. Overall Execution Time of Garbage Collection

Fig. 12 shows the total execution time of GC, including that of erase operations. As shown in the figure, the execution time of ICB was the same, irrespective of transaction size, while GCB performed worse than ICB for single plane operation, when the transaction size was eight pages or less. For multi-plane operation, the same was true when the transaction size was 16 pages or less. When a transaction comprised a smaller number of pages, there should be more transactions for the same amount of data access. The first page in a merge usually experiences higher overhead than do others, because it is not possible to hide the page read latency.

The performance benefit of GCB and DGCB increased when the cache operation was enabled. For single plane, when the cache operation was not used, the execution time of GCB and DGCB decreased by up to 4.52% and 4.63%, respectively, as compared to ICB; when cache operation was used, however, the execution time improved by up to 17.81% and 17.92%, respectively. A similar trend is also observed for multi-plane.

D. Corrected data retransmission overhead of DGCB

In order to quantize the impact of data retransmission overhead on the performance of DGCB, the error rate was varied from 0% (no error) to 100% (192 bits of error), where each page was allowed to have up to 192 bits of error.

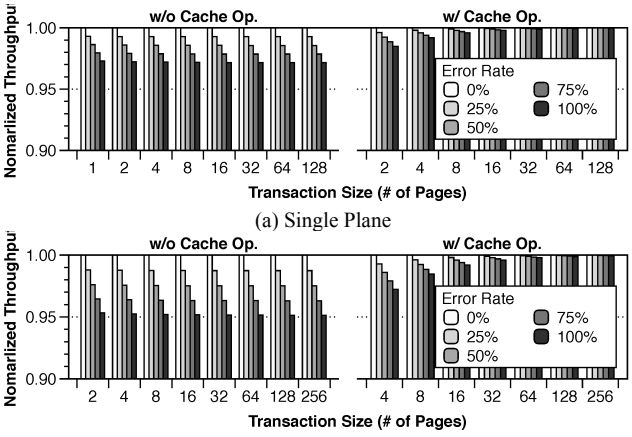


Fig. 13. Normalized Throughput Comparison

Fig. 13 shows the GC performance, including retransmission overheads for DGCB. This figure shows that the throughput was inversely proportional to the error rate. When the cache operation was not used, the performance reductions were the same regardless of the transaction size, because the retransmission of corrected data did not overlap with other operations. Therefore, the throughput was reduced by up to 2.71% and 4.86% for single plane and multi-plane, respectively.

When cache operation was used, however, such overheads from retransmitting corrected data overlapped with program busy of the previous page, and so the performance reduction could be avoided. The overhead from retransmitting corrected data for the first page merge, however, was not overlapped. This problem becomes more significant when the transaction size becomes smaller, because the overhead would occur for every first page merge. However, the percentage reduction in the throughput caused by the overhead was small even in the worst case: 1.51% and 2.77% for single and multi-plane, respectively.

E. Channel Utilization during Garbage Collection

Fig. 14 shows the channel utilization during GC. For DGCB, the channel utilization was measured assuming that the error rate was 100%.

When cache operation was used, the channel utilization level was very high. In particular, when the multi-plane operation was used for GCB, the channel utilization exceeded 60% when the transaction size was 16 pages or more.

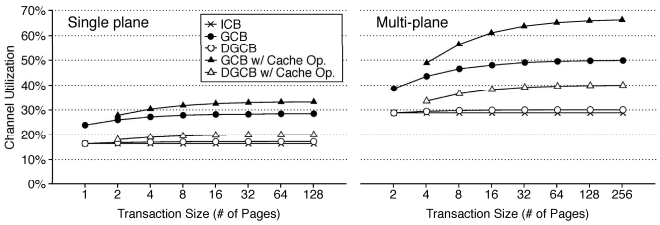


Fig. 14. Channel Utilization

The high channel utilization when cache operation was used is related to the effects of overlapping, which reduces execution time, resulting in the data transfer time being a relatively large proportion of the overall execution time. This is also the reason why the channel utilization of ICB was lower than that of the copy-backs. If the channel utilization is low, the channel is idle for longer periods during GC. During these longer idle periods, another device that shares the channel can therefore perform its own operation, in an interleaved way. If the channel idle time is short, it is difficult to use interleaving. Even if it is used, it could potentially reduce the performance of GC. While another device that shares the channel is performing an operation, the device that is performing GC may be blocked from using the channel, delaying the GC. Therefore, the low channel utilization is advantageous in terms of both the responsiveness and the execution time of the storage.

VII. CONCLUSION

In this paper, we proposed a new technique for reducing the data transfer overhead, which enables devices to directly transmit data in a flash memory-based storage system sharing I/O bus, while updating only the corrected data when an error is detected. The proposed technique improves the performance of GC and reduces I/O bus utilization, compared to existing techniques. The performance of this technique varies depending on the ratio of data transfer time and program busy time, and, as in the case where GC is performed with the multi-plane operation, this method can increase GC's efficiency more as the data transfer time gets longer in the merge operation, compared to the existing methods. Moreover, since I/O bus utilization is quite low during GC, this method can be used more effectively in a storage system that has channels, each of which have multiple flash memories sharing the I/O bus.

REFERENCES

[1] S. Jung, Y. Lee, and Y. H. Song, "A process-aware hot/cold identification scheme for flash memory storage systems," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 339-347, May 2010.

[2] S. Lee, D. Shin, Y. Kim, and J. Kim. "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," *In Proceedings of the International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED2008)*, Feb. 2008.

[3] X. Jin, S. Jung, and Y. H. Song, "Write-aware buffer management policy for performance and durability enhancement in NAND flash memory," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 4, pp. 2393-2399, Nov. 2010.

[4] J. Kang, J. Kim, C. Park, H. Park, and J. Lee, "A multi-channel architecture for high-performance NAND flash-based storage system," *Journal of Systems Architecture*, vol. 53, no. 9, pp. 644-658, Sept. 2007.

[5] N. Agrawal , V. Prabhakaran , T. Wobber , J. D. Davis , M. Manasse , R. Panigrahy, "Design tradeoffs for SSD performance," *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pp. 57-70, June 2008.

[6] C. Dirik and B. Jacob, "The performance of PC Solid-State Disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," *Proceedings of the 36th annual International Symposium on Computer Architecture*, pp. 279-289, June 2009.

[7] H. Jung, S. Jung, and Y. H. Song, "Architecture Exploration of Flash Memory Storage Controller through a Cycle Accurate Profiling," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1756-1764, Nov 2011.

[8] Li-Pin Chang , Tei-Wei Kuo , Shi-Wu Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3 no. 4, pp.837-863, Nov. 2004.

[9] Y. Kim, B. Tauras, A. Gupta, D. M. Nistor, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," *Proceedings of the 2009 First International Conference on Advances in System Simulation*, pp.125-131, Sept. 20-25, 2009.

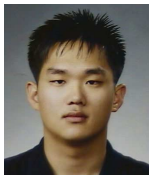
[10] Michael Abraham, "NAND Flash Architecture and Specification Trends," *Flash Memory Summit 2011*, Aug. 8-11, 2011

[11] Mielke, N., Marquart, T., Wu, N., Kessenich, J., Belgal, H., Schares, E., Trivedi, F., Goodness, E., and Nevill, L. R., "Bit error rate in NAND flash memories," *In Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*, pp. 9-19. April, 2008.

[12] Ryan Fisher, "Optimizing NAND Flash Performance," *Flash Memory Summit 2008*, Aug. 12-14, 2008.

[13] OnFi Standard 2.1, *Open NAND Flash Interface Working Group*, Jan. 2009.

BIOGRAPHIES



Jaehyeong Jeong received his Master's degree in the Department of Electronics and Computer Engineering in 2007 from Hanyang University and his Bachelor's degrees in the Department of Information and Computer Engineering in 2005 from Pai Chai University, Daejeon, Korea. Currently, he is a Ph.D. student in the Department of Electronics and Computer Engineering at Hanyang University, Seoul, Korea. His research interests include computer architecture, systems-on-chip, embedded systems, and non-volatile memories.

Yong Ho Song (M'09) received his Ph.D. degree in Computer Engineering from the University of Southern California in 2002 and his Master's and Bachelor's degrees in Computer Engineering in 1991 and 1989, respectively, from Seoul National University in Korea. He is currently an associate professor in the Department of Electronics Engineering, Hanyang University, in Seoul, Korea. His research interests are system architecture and software systems of mobile embedded systems, including systems-on-chip, networks-on-chip, multimedia on multi-core parallel architecture, and NAND flash-based storage systems. His research achievements have been published in numerous academic journals and conferences. He has served as a program committee member of many prestigious conferences, including the IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE International Conference on Parallel and Distributed Systems, and IEEE International Conference on Computing, Communication, and Networks.