A Hybrid Solid-State Storage Architecture for the Performance, Energy Consumption, and Lifetime Improvement

Guangyu Sun, Yongsoo Joo, Yibo Chen
Dimin Niu, Yuan Xie
Pennsylvania State University
{gsun, ysjoo, yxc236, dun118, yuanxie}@cse.psu.edu

Yiran Chen, Hai Li
Seagate Technology
{yiran.chen, helen.li}@seagate.com

Abstract

In recent years, many systems have employed NAND flash memory as storage devices because of its advantages of higher performance (compared to the traditional hard disk drive), high-density, random-access, increasing capacity, and falling cost. On the other hand, the performance of NAND flash memory is limited by its "erase-before-write" requirement. Log-based structures have been used to alleviate this problem by writing updated data to the clean space. Prior log-based methods, however, cannot avoid excessive erase operations when there are frequent updates, which quickly consume free pages, especially when some data are updated repeatedly.

In this paper, we propose a hybrid architecture for the NAND flash memory storage, of which the log region is implemented using phase change random access memory (PRAM). Compared to traditional log-based architectures, it has the following advantages: (1) the PRAM log region allows in-place updating so that it significantly improves the usage efficiency of log pages by eliminating out-of-date log records; (2) it greatly reduces the traffic of reading from the NAND flash memory storage since the size of logs loaded for the read operation is decreased; (3) the energy consumption of the storage system is reduced as the overhead of writing and reading log data is decreased with the PRAM log region; (4) the lifetime of NAND flash memory is increased because the number of erase operations are reduced. To facilitate the PRAM log region, we propose several management policies. The simulation results show that our proposed methods can substantially improve the performance, energy consumption, and lifetime of the NAND flash memory storage[1].

1 Introduction

As solid-state technologies advance, the past several years have been an exciting time for NAND flash memory. The cost of NAND flash memory has fallen dramatically as fabrication becomes more efficient and the market grows; the density has been improved with better process technologies. Consequently, NAND flash memory has been widely adopted by various applications such as laptops, PDA, and mobile phones. For example, Toshiba and SanDisk have produced 16GB NAND flash memory [31], which may be used in the popular mobile devices such as iPhone. In addition, because of its faster performance compared to traditional hard disk drives, NAND flash memory has also been proposed to be used as a cache for hard disk drives [13], or even as the replacement of hard disk drives in enterprise applications [27] [21]. At the same time, some research has been done on database management system (DBMS) implemented using NAND flash memory [20] [19].

One well-known limitation of NAND flash memory is the "erase-before-write" requirement. It cannot update the data by directly overwriting it [33] [22]. In NAND flash memory, read and write operations are performed in the granularity of a page (typically 512 Bytes$\sim$8 KBytes) [7]. A DRAM data buffer is normally employed to store a subset of pages of NAND flash memory for frequent accesses [22]. The modified pages are written back to flash memory when they are evicted from the DRAM data buffer. These pages, however, cannot be directly overwritten to the same place in flash memory. In another word, the NAND flash memory does not allow direct "in-place updating". Instead, a time-consuming erase operation must be performed before the overwriting. To make it even worse, the erase operation cannot be performed selectively on a particular data item or page but can only be done for a block of NAND flash memory called the "erase unit". Since the size of an erase unit (typically 128 KBytes or 256 KBytes) is much larger than that of a page, even a small update to a single page requires all pages in the same erase unit to be erased and written again.

In order to overcome the performance degradation and the energy overhead caused by the "erase-before-write", various techniques have been proposed [4, 5, 9, 20, 28]. One popular idea is to use "out-of-place updating". It means that the updated data is written to other clean pages and the original data is invalidated. File systems employing this method are so-called "log-based file system" [28], and extensive research has been done in this field [15, 28]. Some research has pointed out that the performance of these file system is not good for the access pattern with frequent and small random updates, which is common for many applications including on-line transaction processing (OLTP) [4] [20]. Recently, Lee *et al.* proposed an In-Page Logging (IPL) approach, which can outperform the traditional log-based file systems for such an access pattern [20]. It caters for the characteristic of NAND flash memory by partitioning an erase unit into a data region and a log region. The data

[1]This work was supported in part by NSF grants (0702617, 0903432, 0905365) and SRC grant.

region stores normal data pages and the log region stores updates to the data pages in the form of logs. When a modified data page is evicted from the data buffer, only the updated data is written back to NAND flash memory. Consequently, the data pages in an erase unit are not erased until clean space of the log region runs out. When the log region is full, a "merge operation" is triggered, through which data pages are written to a clean erase unit with all updates applied.

Although the IPL method helps in reducing the number of erase and write operations, it cannot completely overcome the inherent limitation of NAND flash memory because the log region itself is still implemented with NAND flash memory. For example, if data pages of an erase unit are frequently updated, these updates will quickly fill the log region and cause many merge operations. Especially, when a data item is repeatedly updated, many redundant log records are generated for the same data item, but only the latest one is valid. Hence, the utilization of log sectors becomes inefficient for this pattern of updates.

In recent years, various emerging non-volatile memory (NVM) technologies are proposed, such as phase change random access memory (PRAM), magnetic random access memory (MRAM), and resistive random access memory (RRAM). These technologies are considered as competitive candidates for future universal memory. As such emerging NVM technologies are getting mature, it has been a hot topic in computer architecture research to explore the usage of such emerging NVM technologies at different level of memory hierarchy to enable novel architecture design, such as NVM-based cache design [8,32,34,35], NVM-based memory architecture [17,26,37], or NVM-based storage architecture [14, 25]. Compared to NAND flash memory, all these memory technologies have the important advantage of allowing direct in-place updating. The process technologies of these NVMs, however, are not as mature as that of NAND flash memory. It is still not feasible for them to directly replace NAND flash memory as massive storage because of their limitations of manufacture and high cost [16] [8]. Alternatively, we consider using a hybrid storage architecture to take advantages of NAND flash memory and other memory technologies.

In this work, we employ PRAM as the log region of our hybrid storage for several reasons. First, PRAM has the highest cell density among these emerging memory technologies [16]. Second, the capacity of the up-to-date PRAM production is qualified for the log region of NAND flash memory that works as massive storage [18]. Third, prior work has shown that the feasibility of integrating PRAM with NAND flash memory [25] [14]. Our contribution can be summarized as follows:

- We propose to use PRAM as the log region of the NAND flash memory storage system.
- We develop a set of management policies for PRAM log region to fully exploit its advantages of in-place updating ability, fine access granularity, long endurance, etc.
- We study the lifetime of the PRAM log region. The technique is proposed to promise that the PRAM log region will not wear out before the NAND flash memory.

Compared to the IPL method, the hybrid architecture has the following advantages: (1) the ability of "in-place updating" can significantly improve the usage efficiency of log region by efficiently eliminating the out-of-date log data; (2) the fine-granularity access of PRAM can greatly reduce the read traffic from the NAND flash memory to the DRAM data buffer since the size of logs loaded for the read operation is decreased; (3) the energy consumption of the storage system is reduced as the overhead of writing and reading log data is decreased with the PRAM log region; (4) the lifetime of the NAND flash memory could be increased because the number of erase operations are reduced, and the endurance of the PRAM log region can be traded off to further improve the performance of the storage system. The simulation results show that, with proper PRAM management policies, both performance and endurance of the hybrid storage are significantly improved.

2 Background

In this section, we first illustrate the erase-before-write limitation for NAND flash memory. Then, we describe the existing IPL method and its limitation. Finally, we present a brief overview of PRAM technology and the potential to solve the limitation of IPL method.

2.1 The Erase-Before-Write Limitation

NAND flash memory shows asymmetry in how they read and write. While we can read any of the pages of a NAND flash memory, we must perform an erase operation before writing data to a page. The erase operation is performed in the granularity of an "erase unit" consisting multiple adjacent pages [33] [20]. Therefore, writing new data to the same page storing the old data, namely "in-place" update, cannot be performed directly. Instead, it is finished in several steps: (1) back up the other pages in the same erase unit; (2) perform an erase operation; (3) write both the new data and the backed up pages to the erase unit. Such a process is very time consuming.

Because of this erase-before-write limitation, log-based file systems are commonly used for flash memory devices. When some data in a page is updated, the whole page holding the data item is written to another erased page, which is called the log page of current page. Then, the old copy of page is invalidated. This process is called an "out-of-place" updating. The merge operations will happen when the device may run out clean pages, and some invalid pages need to be reclaimed. This process is also called *garbage collection*. During the garbage collection, valid pages in these two erase units need to be merged into a third erase unit. This merge operation is very costly because all valid pages of the erase unit should be written to another erased unit.

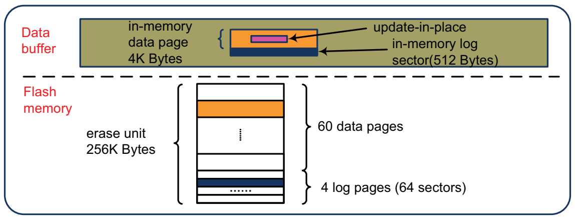

Figure 1. An illustration of the IPL method.

2.2 IPL Method

It is easy to find that even if only a small portion of a page is updated, the whole page must be written to its log page in the log-based file systems, which can significantly degrade the write performance of the file system. Unfortunately, small-to-moderate sized writes are quite a common access pattern for many cases including database applications such as OLTP [4] [20].

Recently, the IPL method [20] is proposed to overcome such a weakness of the log-based file systems for NAND flash memory. Figure 1 gives an illustration of the IPL method for NAND flash memory. Each erase unit consists of 60 data pages and 4 log pages. Each log page is divided into 16 *log sectors*. Unlike the traditional log-based file system, the log page in IPL method is used to record updates of data pages rather than copy a whole data page directly. Whenever an update is performed on a data page in the DRAM data buffer, the copy of data item in the buffer is updated in-place. At the same time, the IPL buffer manager adds a log record in the data buffer, which is allocated in the log sector assigned to the data page. A log sector in the data buffer can be allocated to the data page on demand, and it can contain more than one update record. A log sector in the data buffer can be released when its log records are written back to the log sector in flash memory. The log records are written to NAND flash memory when the log pages of an erase unit become full or when a dirty data page is evicted from the data buffer. When a dirty page is evicted, the data page itself does not need to be written back to flash memory, because all of its updates are stored in the form of log records. Consequently, only its log sectors are written back to NAND flash memory.

Since the number of the log sectors is fixed for each erase unit, the erase unit may eventually run out of empty log sectors after certain number of updates are performed to the pages in the erase unit. For such a case, the IPL manager performs a merge operation as follows. First, it merges the old data pages with their log sectors to create up-to-date data pages. Second, it allocates a clean erase unit to write the generated up-to-date data pages. Finally, it invalidates all the pages in the old erase unit, so this erase unit can be reclaimed for future use.

In the IPL approach, only the log sectors that contain the updated data are written back to flash memory, and the usage of data pages is more efficient than that of traditional log-based flash memory. The size of log pages in an erase unit is limited, and these log sectors themselves do not allow in-place updating. This may cause significant performance degradation for some cases. Particularly, if there are frequent updates to the same erase unit, it would quickly run out its log sectors and cause merge operations frequently. Moreover, if there are multiple updates to the same data, only the latest updated one is valid. Consequently, the effective log sector capacity of an erase unit becomes smaller, which would worsen the problem of the IPL approach.

2.3 PRAM Process Technology

Different from the conventional RAM technologies (such as SRAM/DRAM), the information carrier of PRAM is chalcogenide-based materials, such as $Ge_2Sb_2Te_5$ and $Ge_2Sb_2Te_4$ [12]. The crystalline and amorphous states of chalcogenide materials have a wide range of resistivity, about three orders of magnitude, and this forms the basis of data storage. The amorphous, high resistance state is used to represent a bit '0', and the crystalline, low resistance state represents a bit '1'.

Nearly all prototype devices make use of a chalcogenide alloy of germanium, antimony and tellurium (GeSbTe) called GST. When GST is heated to a high temperature (normally over 600 °C), it will get melted and its chalcogenide crystallinity is lost. Once cooled, it is frozen into an amorphous and its electrical resistance becomes high. This process is called RESET. One way to achieve the crystalline state is by applying a lower constant-amplitude current pulse for a time longer than the so-called RESET pulse. This is called SET process [10]. The time of phase transition is temperature-dependent. Normally, it takes several 10ns for the RESET and takes about 100ns for the SET [10, 36].

As PRAM technologies improve, PRAM has shown more potential to replace NAND flash memory with advantages of allowing in-place updates and fast access speed. Table 1 compares the characteristics of PRAM and NAND flash memory, which are estimated from prior work [24] [29] [30]. Note that the units of write and read operations for PRAM and NAND flash memory are different. The PRAM can be accessed in a fine granularity (byte-based). We will show that this advantage makes the access to the log region much more flexible, compared to the traditional IPL method. Prior research has also shown that the PRAM could achieve the same cell size as that of NAND flash memory with a vertically stacked memory element over the selection device [16, 36]. It means that it is feasible to replace NAND flash memory with PRAM without inducing area overhead.

Currently, it is still not feasible to replace the whole NAND flash memory with PRAM entirely due to its high cost and the limitation of manufacture [16] [18]. Consequently, we propose to use the PRAM as the log region of NAND flash memory instead. A cell of NAND flash memory could be implemented to store multiple bits of data. The multi-level PRAM is still under research [23]. In this work,

Tech.	Cell Size	Write Cycles	Access Time			Access Energy		
			Read	Write	Erase	Read	Write	Erase
Flash	$4F^2$	10^5	284μs/4KB	1833μs/4KB	$>$20ms/Unit	$9.5\mu J$/4KB	$76.1\mu J$/4KB	$16.5\mu J$/4KB
PRAM	$4F^2$	10^8	$80ns$/word	10μs/word	N/A	$0.05n$J/word	$0.094n$J/word	N/A

Table 1. The comparison between PRAM and NAND flash memory technologies.

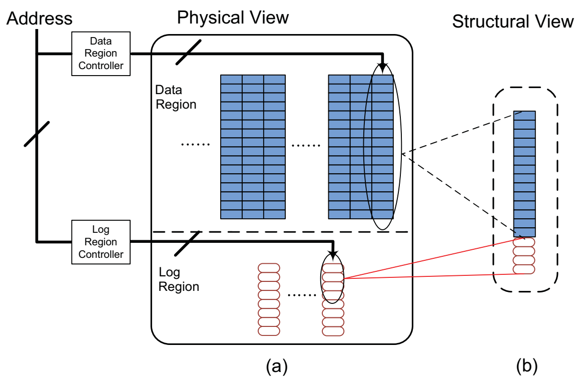

Figure 2. Physical and structural views of the hybrid architecture.

the single-bit PRAM is used.

3 Overview of the Hybrid Architecture

Figure 2 shows the physical and structural views of NAND flash memory storage system with the PRAM-based log region. Since the process technologies of PRAM and NAND flash memory are different, it is difficult to physically place the PRAM-based log region together with the flash-based data region. Unlike the IPL method, the PRAM log region is separated from the flash-based data region in our work as shown in Figure 2(a). Based on log region management policies and run-time operations of applications, log sectors are dynamically assigned to each erase unit to store its own updates. The structural view of such relationship is shown in Figure 2(b). We need to maintain metadata for the relationship between data pages and their corresponding log sectors. The *log region controller* in Figure 2 takes the responsibility of decoding addresses and managing the metadata for the log region. The access to the hybrid architecture is described as follows:

- For the read operation, the address of the accessed data is sent to both the data region and the log region. If there exist log sectors for the requested data page, they are loaded into the data buffer as well as the original data page to create the up-to-date data page.
- For the write operation, only updates are sent back to the PRAM log region. There are several scenarios:
  - Case1: If there are no existing log records for the accessed data page, a new log sector is allocated to the accessed data page. Then, the current update is written to the log sector.
  - Case 2: When some log sectors have already been allocated to the accessed data page, the log records in these log sectors are compared with the current update. If there is a log record that has the same data address of the current update, the existing log record is overwritten by the current update, and no extra log record is generated.
  - Case 3: If the current update does not match any of existing log records, a new log record containing the current update is written to the log sectors of the data page. When log sectors assigned to the data page are fully occupied, a new log sector is requested for the current update as in Case 1.
- Merge operations are triggered when the merge conditions are satisfied. The merge conditions depend on the log region management policies, which will be introduced in following sections. During merge operations, updates in these log sectors are applied to corresponding data pages, and the data pages are written to another free erase unit. The out-of-date data pages are invalidated and the corresponding log sectors are released as clean ones for future use.

The data address of the current update needs to be compared to those of existing log records in order to achieve the in-place update. A relative address ($addr_{Relative}$) is used to represent the position of an update inside the accessed data page. It can be calculated by $addr_{Relative} = addr_{Update} - addr_{Page}$. The $addr_{Update}$ and $addr_{Page}$ represent the real addresses of the update and the accessed data page, respectively.

As we mentioned, an access to a PRAM log region is managed with its own controller. The access to the log region is operated in parallel with that to the data region. Since the size of a log region is much smaller than that of a data region, the accessing delay on the peripheral circuitry of the log region is shorter than that to the data region. It means that using hybrid architecture will not induce extra delay. Instead, the total access latency could even be reduced in some scenarios. For example, in write operations, the latency is decided by the time of writing the update to the log region. Then, the performance could be improved with a shorter decoding time.

Besides the shorter access latency, the more important thing is that the hybrid architecture can take advantages of in-place updating capability of the PRAM log region. Although it takes extra time to search whether the current update does not match existing log records, the overhead is trivial compared to the write latency because the read operation is much faster than the write one. If the in-place updating happens, much more benefits can be achieved. For example, since the update is written to the existing record, the time of allocating new log record and modifying metadata is saved. It is also beneficial in that the log region is used more efficiently: it would reduce the numbers of erase and write operations by increasing the effective capacity of the log region.

Furthermore, because the PRAM log region can be accessed with byte granularity, the performance of read operations can also be improved. In the IPL method, since the log data is loaded in the page granularity, some log sectors that do not belong to the accessed data page are also loaded. If the log records of one data page are stored in more than one physical page of NAND flash memory, the overhead of loading the log records can be even higher than that of loading the data page. On the contrary, in our hybrid architecture, only the log records that belong to the accessed data page are loaded. It can greatly enhance the performance of read operations.

4 Management Policy of PRAM Log Region

The in-place updating and fine access granularity of PRAM log region provide opportunities of optimizations. In this section, we propose a few assignment and corresponding management policies for the PRAM log region.

4.1 Static Log Region Assignment

In the IPL method, each erase unit is uniformly assigned a fixed number of log pages. It is obvious that such a static assignment also works with the PRAM log region. This method of assignment is named as the *basic static assignment*. Its illustration is shown in Figure 3(a). When the data pages in the erase unit are updated, only the log sectors that belong to the erase unit are assigned for these updates. Although the static log sector assignment is the same as that in IPL method, the performance can be improved using the in-place update capability of PRAM. As in the IPL method, the merge operation is triggered when all log pages of an erase unit are fully occupied and no free log sector can be assigned to a new update. Note that the new update means that the incoming update is not matched to any existing log records. With the PRAM log region, when all log pages of an erase unit are used, it is possible that the logs are written in-place without causing merge operations.

The main advantage of the static assignment is its simplicity in implementation. Since the log sectors assigned to each erase unit are predetermined and fixed, the overhead of keeping and searching the metadata of log sectors is negligible. Such uniform assignment, however, becomes inefficient when updates are not evenly distributed among erase units. In applications such as OLTP, the access/update intensity to each erase unit is typically not uniform. Normally, the distribution of update numbers in respect of erase unit is highly skewed for a trace of TPC-C benchmark. Note that TPC-C is a popular benchmark that simulates a complete computing environment where a population of users executes transactions against a database [1]. Thus, for such applications, it is inefficient to divide and assign the log region equally to each erase unit.

One way to mitigate the effect of uneven access distribution is to organize erase units into groups, which is named *group static assignment*. In this way, log pages in a group can be shared among erase units, as shown in Figure 3(b). After sharing log pages in a group, the frequently updated erase units in a group could be assigned more log pages. Note that such a method of sharing log pages among erase units is not feasible for the IPL method using only NAND flash memory. It is because the log pages are placed together with data pages inside each erase unit of the IPL method. If we want to assign data pages with external log pages from other erase units, a pointer-like structure is needed to record locations of external log pages, which may increase the design complexity and timing overhead. Furthermore, since the log region is managed in page granularity, sharing log pages among erase units may generate some data pages, which have too many log pages. On the contrary, in our PRAM log region, all log pages are placed together. There is no difference whether we manage log pages in the granularity of an erase unit or in a group with several erase units. Similar to the basic static assignment, the merge operation is also triggered when the log pages assigned to each group run out. Note that all erase units in the group need to be merged together.

Although the grouping method could help in mitigating the effect of unbalanced accesses, there are some limitations with it. First, if most of erase units inside a group are frequently updated, the log pages shared in this group cannot reduce the number of merge operations much. Second, it is possible that only a few erase units in a group are frequently updated and cause many merge operations. Since the whole group are erased and written together in the merge operation, most pages are erased even without any modifications. Therefore, the size of a group could not be too large in order to avoid the overhead of such redundant erases. Third, it takes more time to access and manage log pages as the size of a group increases.

4.2 Dynamic Log Region Assignment

In order to overcome the limitations of the static assignment, we propose a dynamic log page allocation method. The basic idea of the dynamic allocation method is that the number of log sectors of an erase unit could be assigned on-demand based on the number of its updates. In other words, the log region is *shared* among all erase units. If an erase unit is frequently updated, the log region controller assigns more log sectors to it unless there is no remained free log sectors in the log region. Note that such a dynamic assignment is different from the group static assignment. The log sectors of each erase unit are managed individually, and the merge operation of one erase unit will not affect the others.

Figure 3(c) gives an illustration of the dynamic assignment. At the beginning, there are no updates to any data page, and all log sectors are free. During the access pro-

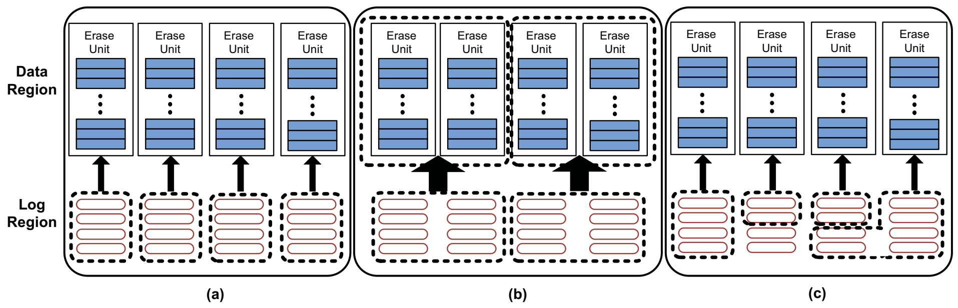

Figure 3. Examples of (a) the basic static log assignment, (b) the group static log assignment, (c) the dynamic static log assignment.

cess, the number of log sectors assigned to each erase unit grows up based on its updates. The frequently updated erase unit are assigned more log sectors.

The dynamic assignment promises that frequently updated erase units can get more log pages so that the number of merge operations are significantly reduced for applications having unbalanced updates. Such a dynamic assignment is not feasible for the IPL method with the NAND flash memory log region, which does not allow in-place updating. It is obvious that the number of the log sectors is increased linearly in proportion to the size of updates when the in-place updating is not allowed. When the updates to erase units are unbalanced, the frequently updated erase unit may have too many log sectors, and the overhead of accessing such erase unit is not tolerable. On the contrary, in our PRAM log region, the number of log sectors is not increased when some data are repeatedly updated. In the worst case, the size of logs is the same as that of the data. Therefore, accessing overhead for the dynamic assignment is moderate with the PRAM log region.

Intuitively, the merge operations are triggered when the whole PRAM log region is fully occupied. In that case, however, once the merge operation is triggered, all log sectors should be merged to the corresponding data pages at the same time, which makes the storage system not available for a long time. Hence, if the burst of updates arrives when the log region is nearly full, the whole system has to be stalled for a long time waiting for merge operations to be completed, which is not desirable. Instead, we set up a threshold of free log sectors, and the merge operations are triggered when the capacity of free log sectors are lower than it. Its advantage is that there are always some free log sectors reserved for the burst of updates. In addition, it is more likely to process merge operations during the idle time of the storage system.

With the dynamic assignment method, the whole log region is managed together. It is possible to reduce the number of merge operations by merging eras units selectively instead of merging them all together. Such policies are described as follows,

- If some erase units are just frequently updated, it is highly possible that these pages will be accessed repeatedly in the near future. Such erase units are named as *hot units* in this work. If we could prevent these hot pages from being merged, the number of merge operations could be reduced. A simple FIFO queue is employed to record most recently updated erase unit. When the merge operation is triggered, erase units in the queue are left untouched so that hot data will not be merged.
- The data of whole erase unit is copied to a clean space, and the timing overhead mainly depends on the erase time. Therefore, it is not worth merging the pages that have only a few log sectors. In another word, merging such erase units is not efficient because it will not releases many log sectors. Consequently, we could set up a threshold based on the number of log sectors of the erase unit, which is named as *merge threshold*. The erase unit is kept untouched during the merge process if the number of its log sectors are lower than the merge threshold.

The hot unit queue size and the threshold of log sector number have the important impact on the number of merge operations and the lifetime of the system. These issues are further discussed in the experimental sections. The size of metadata for dynamic assignment is larger than that for static assignment. Since we want to share the whole log region among data pages, we need more bits in metadata to store the location (address) of the log sectors. In this work, the metadata is located in the PRAM because it is normally frequently accessed and modified [25]. It means that the available log region capacity is reduced by storing metadata. In addition, we need to keep the record of hot erase units with a FIFO queue. In this work, a link based table structure is used for keeping the metadata of log sectors. Note that some data structures, such as the hash table, could be employed to reduce the space and timing complexity of searching and accessing the metadata. Such topics are out of the scope of this work and are not discussed. We will discuss overhead of storing metadata later, and results show that the performance is still improved with the overhead.

5 Endurance of the Hybrid Architecture

The lifetime endurance is one important issue of the NAND flash memory. There are various good approaches proposed for wear leveling of NAND flash memory [3, 6, 11]. When the in-place updating is enabled in the hybrid architecture, the endurance of the storage system can be improved. Since updates are written to the PRAM-based log region, the write intensity to the data region is greatly reduced compared to the pure NAND-flash memory. For the same applications, the lifetime of the data region is increased with the PRAM-based log region. Because PRAM has much better endurance than the NAND flash memory, the log region may still wear slower than the data region does. If we promise that the log region will not wear out before the data region, the endurance of the whole system is increased. As we mentioned in the previous section, we can use selective merge operations to reduce the number of merge operations. The lifetime of the data region is further improved at the same time, but the log region may wear faster. It is a trade-off between performance and the endurance of the log region. Consequently, the wear-leveling is necessary for the PRAM-based log region. In this section, we discuss the issues related to the lifetime of the log region and propose the technique for wear-leveling.

5.1 Lifetime of the Log Region

In the IPL method, the log pages of an erase unit can only be written once before being merged with data pages. Therefore, log and data pages have the same level of wearing. On the contrary, in our PRAM hybrid system, the merge operations should be controlled to promise that the log region will not wear out before the data region. Besides the basic merge conditions introduced in the previous section, several other merge conditions are enforced to prevent the log region from wearing too fast.

For the basic static log assignment, since the log sectors assigned to each erase-unit are fixed, the lifetime of each erase unit could be controlled individually. Table 1 shows that the number of allowed writes of PRAM is about 1000 times larger than that of NAND flash memory. Theoretically, if there are 1000 updates to the same log record, the erase unit should be merged. Then, the data and log regions have the same wear level. In this work, a threshold is set up for each log sector. If a log sector is updated up to 1000 times, a merge operation is triggered for the erase unit. In the worst case, a single log record in the log sector will not be updated for more than 1000 times before the erase unit is merged. Therefore, the log region will not wear out faster than the data region does. Such method also works with the group static log assignment. In fact, even for applications with high intensive write operations, such case of forced merge operations rarely happens. With the static assignment, the lifetime of the hybrid system is decided by the data region. Since the number of merge operations is reduced with the hybrid architecture, the endurance is also improved. Note that a 10-bit counter is needed for each log sector.

For the dynamic log assignment, the case is more complicated because the log region is shared among all data pages. It is possible that some log sectors are reused repeatedly by data pages with intensive updates. Therefore, using a threshold as in the static assignment may not efficiently prevent the log region wearing out before the data region does. In addition, the lifetime of the system is related to the management policies. First, the number of merge operations could be reduced if the policy prevents the hot pages being merged. The log sectors belonged to these hot pages may wear faster because they may be used for a long time without being merged. Second, the number of merge operations could be reduced if only data pages with large number of log sectors are merged. It is possible that some data pages are updated for a few times, then, they are not accessed for a long time, which are called *cold pages*. Since these cold pages would not be merged for a long time, their log sectors will not also be used for a long time, which contributes to the unbalanced wear-leveling. We introduce the techniques of wear-leveling to deal with these problems and improve the lifetime of the log region.

5.2 Wear Leveling

With the dynamic log assignment, when a data page requires for a new log sector, the controller chooses one from the pool of free log sectors. The write intensities of log sectors may be unbalanced with random assignments of log sectors. Therefore, for each request, the log sector with the lowest write numbers should be assigned to even out the write intensities. Theoretically, we should keep tracing the total number of write operations for each log sector. As the PRAM can normally endure 10^8 write cycles, it needs a 30-bit counter to record the total number of write operations. In order to reduce the area and timing overhead, we can use a small counter instead and reset its value periodically to track the approximate wear-level of each log sector. As we introduce in Section 5.1, a 10-bit counter is needed for each log sector in the static log assignment. This counter can be employed in the dynamic log assignment for wear leveling. A wear-aware dynamic log assignment is described as follows:

- The free log sector pool is organized as a link structure. Initially, all log sectors are randomly linked together.
- For each requirement, the head sector of the link is evicted and assigned.
- After each merge operation, the released log sectors are inserted into the link structure based on the number of write operations recorded in its counter. The link structure is kept in sorted order according to the number of writes of each log sector.
- When any of the counter reaches the maximum counting number, the write number of the log sector at the tail of the link structure is subtracted from all counters.

With this structure, the free log sectors are sorted approximately based on their write cycles. The write intensities are balanced among log sectors.

6 Metadata and Management Overhead

The metadata should record log sectors that belong to each data page. In this work, the information (address) of each log sector is stored individually in the structure named *Record Entry*. The log page table points to the entry recording the first log sector that belongs to the data page. The rest entries of log sectors, which belong to the same data page, are stored in the link structure. In each entry of the log sector, the address of the next entry is also recorded. Consequently, all log sectors of a data page can be accessed sequentially with this link structure. Since the number of log sectors are fixed, the address of the log sector stored in each entry can be fixed. It means that only the "next entry" part need to be modified when log sectors are assigned to different data pages. Note that this is not the only structure of metadata. This structure, however, promises that the metadata will not wear out before the corresponding log sector. It is because the metadata space of a log sector is fixed in the PRAM region, and the metadata is only modified after the log sector is updated.

The management of log assignments also incurs timing overhead. For the read operation, the latency of accessing the metadata (record entries) varies for each page. With the in-place updating ability, the log size of a data page is no larger than that of one data page. Therefore, it takes less than $100\mu s$ to access all log record entries for each data page. For the write operation, the metadata need to be updated if there is a new log sector assigned to a data page. In the worst case, the whole log sector is occupied with the new updates, and a new log sector is assigned to the data page. In this case, the size of updated PRAM log region is the same as that in the NAND flash log region. Since the write latency to PRAM is less than that to NAND flash memory, the latency of this part is reduced.

During the merge operation, it takes less time to release log pages in PRAM log region than that in NAND flash one, because the average log size of a data page is reduced with the PRAM log region. In order to achieve the wear-aware log assignment, it takes extra latency to update the sorted tree of free log sector. In the worst case, the timing complexity of searching the binary tree is less than 100us, which is trivial compared to that of a merge operation ($> 20ms$).

7 Experimental Results

In this section, we presents the simulation results of our hybrid architecture with different configurations. Then, the results are analyzed and compared with those of prior work.

7.1 Experimental Setup

We use TPC-C benchmarks [1] for the evaluations, which are generated using an open source tool *Hammerora* [2]. The tool runs with a mysql database server on a Linux platform under different configurations. In our experiments, the configuration of transactions in these benchmarks are described in the Table 2. Note that different sizes of data buffers are simulated. The current operating

1G.(20-100)M.100u:	1GByte database, 100 simulated users, the size of buffer pool varies from 20 MBytes to 100 MBytes

Table 2. Configurations of benchmarks.

systems do not support the management of data buffer as shown in Figure 1. In order to obtain proper log-based access traces to NAND flash memory, we implemented a simulation tool of data buffer with the structure shown in Figure 1. This simulation tool uses memory requests from the processor to the database as the input. Then, the accesses to NAND flash memory storage system are generated based on these memory requests with the log-based data buffer. In order to obtain quantitative evaluations of performance, power consumption and lifetime, we implement the models of PRAM and corresponding simulation tools in device, circuit and system levels. For the NAND flash memory, we study recent work and products specifications from industries [20, 29–31], then, we adjust and integrate proper parameters into our simulation tools.

The sizes of a page, an erase unit, and a log sector are set to be 4 KBytes, 256 KBytes, and 512 Bytes, respectively. For the configuration of the IPL method using only NAND flash memory, there are four log pages in each erase unit. We have mentioned in Section 2 that if the multi-level storage is supported with the PRAM log region, the area of log region is not increased if we replace NAND flash memory with PRAM of the same capacity. In order to explore the advantages of using PRAM, the size of PRAM log region is reduced to be half of the flash log region. Therefore, the area of log region is still kept the same even if the multi-level storage is not employed for the PRAM log region. We will show that performance is still improved. Thus, for the simple static assignment configuration of hybrid architecture, there are 8KBytes of PRAM logs in each erase unit. For the dynamic assignment, the merge operations are triggered when the size of free log sectors are lower than 30% of the total size of PRAM. Then, we ensure that the system will not be stalled for a long time when burst write operations happen.

7.2 Write Performance Simulation

As we addressed before, the large overhead of write and erase operations is the obstacle of improving performance of NAND flash memory. Our main goal of using the PRAM log region is to improve the write performance. Similar to prior work, we also consider the impact of the data buffer size on the write performance. As shown in Figure 4(a), the number of write operations is reduced as the size of data buffer increases. Note that the different buffer sizes are considered to show the efficiency of PRAM log region under different environments. The actual buffer size may be decided in real cases for different applications.

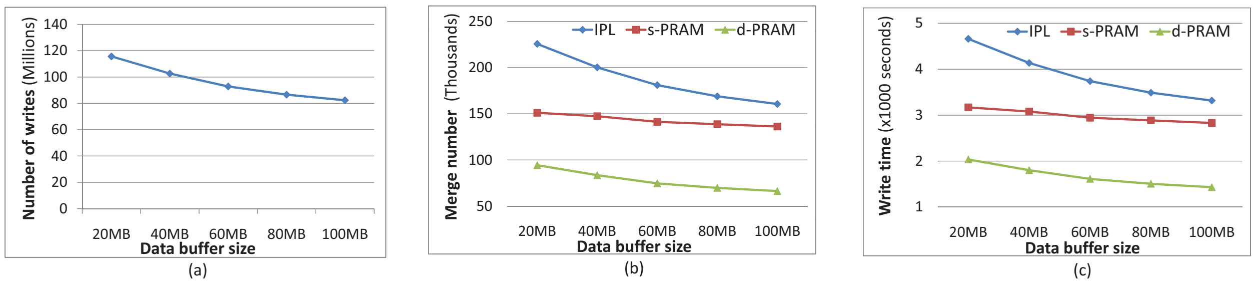

Figure 4. (a) The impact of data buffer size on the write operations; (b) The comparison of merge numbers; (c) The comparison of write time.

The comparison of merge numbers with different methods is shown in Figure 4(b). The *IPL* represents the pure NAND flash memory using the IPL method. The *s-PRAM* represents the hybrid architecture using the basic static log assignment. The *d-PRAM* represents the hybrid architecture using the basic dynamic log assignment. It means that there are no hot unit queue or merge threshold for optimizations of merge operations.

The results show that the number of merge operations is reduced when the in-place updating is enabled in the PRAM log region. For the static log assignment, the number of merge operations is reduced by 22.9% on average, compared to that of the IPL method. We can find that the hybrid architecture works better when the size of data buffer is smaller. The reason is that, some write operations to the storage system may be filtered when the size of data buffer is increased. Then, the write intensities become more unbalanced. Therefore, the static log assignment method works less efficient (as discussed in Section 3). We can also find that using the dynamic assignment of log pages further reduces the number of merge operations significantly. On average, the number of merge operations is reduced by 58.5%, compared to the IPL method. This observation is consistent with the previous discussion. All log sectors are assigned on demand with the dynamic assignment. Consequently, we get more benefits from the in-placing updating with the dynamic assignment of log pages. The more important thing is that the size of data buffer has little impact on the working efficiency of the dynamic assignment method. As we discussed before, the dynamic assignment method works well even when the write intensities are highly unbalanced.

The write time is also simulated and compared in Figure 4(c). The similar conclusion can be drawn for the write time because the results have the same trend as that of merge operations. It is reasonable because the time consumed by merge operations is dominating in total time of write operations.

In Figure 5(a), the numbers of merge operations are shown for the hybrid architecture using the group static log assignment. The results include the numbers of merge operations with different group sizes. The number of merge operations decreases as the group size increases, but the improvement is not significant. We have mentioned that the overhead of management is increased with the size of group. The results show that we can get more benefits from the group method when the size of data buffer is larger. It is because the group method is used to mitigate the effect of unbalanced write intensities, and the write intensities become more unbalanced with a larger size of data buffer.

In Figure 5(b), the results of using different sizes of hot unit queues are compared. At the beginning, the number of merge operations decreases as the size of hot queue increases. It means that we can get more benefits from in-place updating of hot erase units, if they are not merged. The number of merge operations is increased when the queue size is too large. It is because too many erase units are kept untouched during the process of merge operations. The capacity of free log sectors is reduced greatly. Furthermore, many erase units in the queue are not hot ones when the queue size is too large. The results show that the method works best with the queue size of 16. The number of merge operations is reduced by 6.9% on average.

Figure 5(c) shows the results after we set up the threshold number of log sectors for merge operations. The number of merge operations could be reduced significantly after we set up the threshold because the log sectors are released more efficiently for each merge operation. Similarly, the threshold cannot be too large. Otherwise, the total number of free log sectors are reduced greatly, and the number of merge operations is increased. The results show that the method works best with the merge threshold of 128. The number of merge operations is reduced by 35.2% on average.

7.3 Read Performance Simulation

For different methods, the sizes of data and log that are read from the NAND flash memory are listed in Table 3. The *g-PRAM-4* represents the hybrid architecture using the group static log assignment, and there are 4 erase units in each group. The *d-PRAM-16-128* indicates the hybrid architecture, which uses the dynamic log assignment with the hot unit queue and merge threshold. The size of hot unit queue is set to 16, and the merge threshold is set to 128. The *Average Overhead* shows the average percentage of log

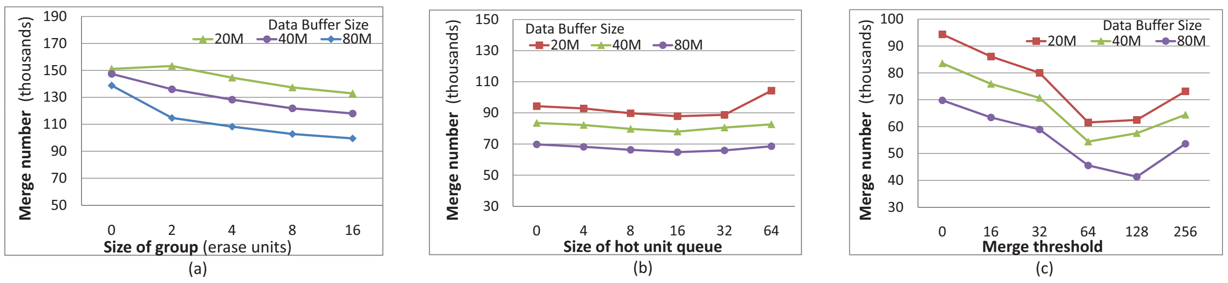

Figure 5. (a) The impact of the group size on the group static assignment. (b) The impact of the hot unit queue size on the dynamic assignment. (c) The impact of the merge threshold on the dynamic assignment.

data in total data per read operation. The results of static and dynamic assignments are compared to the IPL method. The *Reduction* represents the decrease of overhead for reading log data in each read operation, when results are compared to the IPL ones.

The results show that the average time of a read operation is also reduced with the PRAM log region because of two reasons. First, for each read operation, the number of loaded log records is reduced with the in-place updating. In addition, the read speed of PRAM is faster than that of NAND flash memory. The effect of the data buffer is also considered for the read operation. The three sets of results show that we can get similar benefit from using the PRAM log pages with different sizes of data buffers. One interesting observation is that more log pages are read if we use the dynamic assignment method. It is because the frequently accessed erase units are assigned more log pages. The frequent updates also generate more log pages for the same data page. And the numbers of read operations to these erase units are much larger than those of other ones because of intensive accesses. Consequently, the total number of read operations are increased. Nevertheless, it is worth using the dynamic allocation method because the write performance is increased greatly. With write and read evaluation results, the estimated total execution time is compared for managements policies, which is shown in Figure 7(a).

Methods	Log Records Read	Avg. Overhead	Reduction (%)
1GB data, 20MB data buffer, Pages Read:103220			
IPL	6178 pages	5.99%	-
g-PRAM-4	23680 KBytes	2.87%	52.1%
d-PRAM-16-128	33320 KBytes	4.04%	32.6%
1GB data, 40MB data buffer, Pages Read:87600			
IPL	5715 pages	6.52%	-
g-PRAM-4	21040 KBytes	3.00%	54.0%
d-PRAM-16-128	29768 KBytes	4.25%	34.9%
1GB data, 80MB data buffer, Pages Read:68240			
IPL	4742 pages	6.95%	-
g-PRAM-4	18080 KBytes	3.31%	52.3%
d-PRAM-16-128	24304 KBytes	4.45%	35.9%

Table 3. Read Performance Evaluations

7.4 Energy Consumption Evaluation

The energy consumption for different methods is compared in Figure 6. The write operations energy is shown in Figure 6(a). The dynamic log assignment consumes the least energy for the same benchmark because the number of merge operations is greatly decreased with this method. Furthermore, the total size of log written to the log region is reduced with the ability of in-place updating. For the read operation, using PRAM log region can also help to reduce the energy for two reasons. First, the average size of log data loaded with one read operation is reduced. Second, it takes less energy to access the PRAM than that to access the same size of NAND flash memory. Note that the dynamic assignment method consumes a little more read operation energy than that of the static method. It is also because the frequently accessed erase units are assigned more log pages. The total energy consumption is also compared in Figure 6(c). Since the energy of write and merge operations dominates, the dynamic log assignment still consumes the least energy consumption when both read and write operations are considered.

7.5 Lifetime Evaluation

In this work, we assume the write cycles of NAND flash memory are 10^5, and the write cycles of PRAM are 10^8. In order to evaluate the lifetime of the data region, the benchmark traces are kept feeding into the simulation tool. We keep tracking the number of write operations till one cell of data region wears out. The lifetime of the log region is simulated in the same way. Then, based on the traces, we can estimate and compare the lifetime.

The lifetime of data and log regions with the best dynamic assignment(d-PRAM-128-6) are compared in Figure 7(b). The first column is the lifetime of the data region. The second column is the lifetime of the log region without using any wear leveling technique. The third column is the optimized lifetime of the log region with the wear leveling technique. We can find that, without wear leveling,

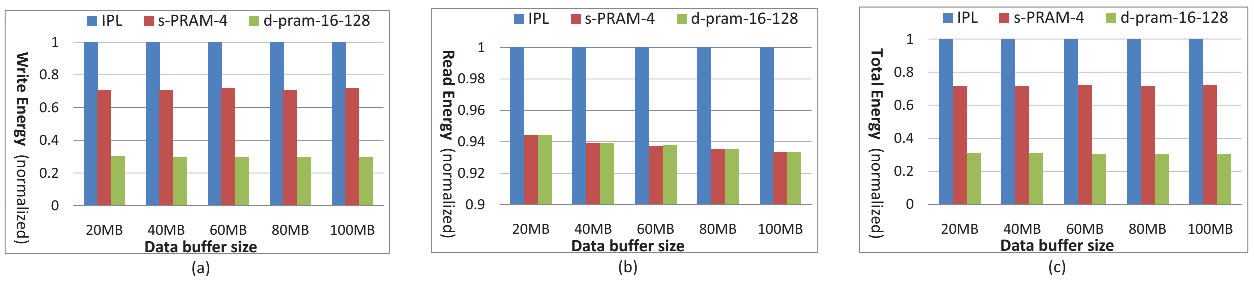

Figure 6. (a) The comparison of write operation energy; (b) The comparison of read operation energy; (c) The comparison of total energy consumption.

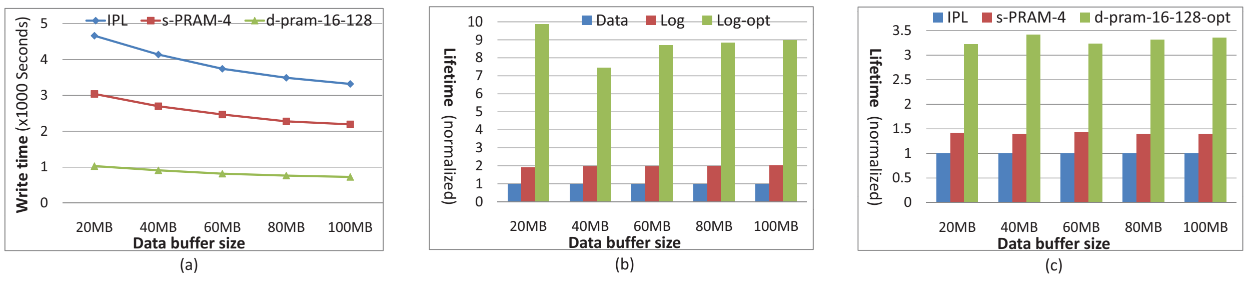

Figure 7. (a) The comparison of the total execution time; (b) The comparison of lifetime between the data region and the log region; (c) The comparison of the whole storage system lifetime with different configurations.

the lifetime of the log region is just a little longer than that of the data region for benchmarks in this work. It is possible that the log region wears out first for benchmarks with a little higher write intensities. After we use the wear leveling technique, the lifetime of the log region is increased to about 10 times higher than that of the data region. It promises that the log region will not wear out before the data region does, even for benchmarks with much higher write intensities. Then, the lifetime of the whole storage system is decided by that of the data region. The lifetime of the whole storage system are compared in Figure 7(c) with different configurations. The results show that the lifetime of the whole storage system is improved after using the PRAM log region. It is because the number of erase operations is decreased with the same write intensity. Consequently, the hybrid storage system with the dynamic log assignment has the longest lifetime.

8 Conclusion

The performance of NAND flash memory is limited because of its erase-before-write requirement, which does not allow in-place updating. The optimizations and managements on NAND flash memory architecture are also limited by this problem. The hybrid architecture using NAND flash memory and another non-volatile memory, such as PRAM, makes it possible to exploit the advantages of both technologies. Our PRAM-based log region method has shown that the performance of the flash memory could be improved significantly, if the log region allows the in-place updating even without the support of multi-level storage. We also show that our method can decrease the overhead of read operations and increase the lifetime of the storage system. Furthermore, the energy consumption of both read and write operations is reduced. More important, the in-place updating capability of PRAM provides more flexible management policies that enable further optimizations of performance and lifetime. In the future, we plan to study the hybrid non-volatile memory architecture that combines other emerging memory technologies (such as MRAM and RRAM) with the NAND flash memory to further explore the efficiency of using non-volatile memories.

References

[1] http://www.tpc.org.

[2] http://hammerora.sourceforge.net/.

[3] Increasing flash solid state disk reliability. *Technical Report, SiliconSystems*, 2005.

[4] A. Birrel, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. *Technical Report MSR-TR-2005-176, Microsoft Research*, 2005.

[5] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters

for data-intensive applications. In *Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 217–228, 2009.

[6] Y. Chang, J. Hsieh, and T. Kuo. Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design. In *Proceedings of Design Automation Conference*, pages 212–217, 2007.

[7] M. Chiang and R. Chang. Cleaning policies in mobile computers using flash memory. *Journal of Systems and Software*, 48(3):213–231, 1999.

[8] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen. Circuit and Microarchitecture Evaluation of 3D Stacking Magnetic RAM (MRAM) as a Universal Memory Replacement. In *Proceedings of Design Automation Conference*, pages 554–559, 2008.

[9] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 229–240, 2009.

[10] S. Hudgens. OUM nonvolatile semiconductor memory technology overview. In *Proceedings of Materials Research Society Symposium*, 2006.

[11] D. Jung, Y. Chae, H. Jo, J. Kim, and J. Lee. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 160–164, 2007.

[12] D. Kang, D. Ahn, K. Kim, J. Webb, and K. Yi. One-dimensional heat conduction model for an electrical phase change random access memory device with an $8f^2$ memory cell (f=0.15 μm). *Journal of Applied Physics*, 94:3536–3542, 2003.

[13] T. Kgil, D. Roberts, and T. Mudge. Improving NAND flash based disk caches. *In Proceedings of International Symposium on Computer Architecture*, pages 327–338, 2008.

[14] J. Kim, H. Lee, S. Choi, and K. Bahng. A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In *Proceedings of ACM International conference on Embedded software*, pages 31–40, 2008.

[15] S. Kim and S. Jung. A log-based flash translation layer for large NAND flash memory. *In Proceedings of International Conference on Advanced Communication Technology*, 3:1641–1644, 2006.

[16] C. Lam. Cell design considerations for phase change memory as a universal memory. In *Proceedings of International Symposium on VLSI Technology, Systems and Applications*, pages 132–133, 2008.

[17] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 2–13, 2009.

[18] K. Lee et al. A 90nm 1.8V 512Mb diode-switch PRAM with 266MB/s read throughput. *In Proceedings of IEEE International Solid-State Circuits Conference*, pages 472–616, 2007.

[19] K. Lee, H. Kim, K. Woo, Y. Chung, and M. Kim. Design and implementation of MLC NAND flash-based DBMS for mobile devices. In *Journal of Systems and Software*, March, 2009.

[20] S. Lee and B. Moon. Design of flash-based DBMS: an in-page logging approach. *In Proceedings of ACM International Conference on Management of Data*, 2007.

[21] S. Lee, B. Moon, C. Park, J. Kim, and S. Kim. A case for flash memory SSD in enterprise database applications. In *Proceedings of ACM International Conference on Management of Data*, pages 1075–1086, 2008.

[22] A. Leventhal. Flash storage memory. *Journal of Communications of the ACM*, 51(7):47–51, 2008.

[23] T. Nirschl et al. Write strategies for 2 and 4-bit multi-level phase-change memory. In *Proceedings of IEEE International Electron Devices Meeting*, pages 461–464, 2007.

[24] S. Park, D.Jung, J. Kang, J. Kim, and J. Lee. CFLRU: a replacement algorithm for flash memory. In *Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 234–241, 2006.

[25] Y. Park, S. Lim, C. Lee, and K. Park. PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash. In *Proceedings of ACM Symposium on Applied Computing*, 2008.

[26] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 24–33, 2009.

[27] D. Reinsel and J. Janukowicz. White paper: Datacenter SSDs: Solid footing for growth. *http://www.samsung.com/global/business/semiconductor/products/flash/FlashApplicationNote.html*, 2008.

[28] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. *ACM Transaction on Computer Systems*, 10(1):26–52, 1992.

[29] Samsung Electronics. datasheet K9G8G08UOM. 2006.

[30] Samsung Electronics. datasheet KPS1215EZM. 2006.

[31] N. Shibata et al. A 70 nm 16GB 16-Level-Cell NAND flash memory. *In Proceedings of IEEE Symposium on VLSI Circuits*, 43(4):929–937, 2007.

[32] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 239–249, Feb. 2009.

[33] Toshiba America Electronic Components, Inc. NAND flash applications design guide. 2004.

[34] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 34–45, 2009.

[35] W. Zhang and T. Li. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2009.

[36] Y. Zhang et al. An integrated phase change memory cell with Ge nanowire diode for cross-point memory. In *Proceedings of IEEE Symposium on VLSI Technology*, pages 98–99, 2007.

[37] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pages 14–23, 2009.