ISSN:0975-9646

An Innovative Algorithm for Flash memory

M. N. Kale , A. S. Jahagirdar

PDVVPCOE Department of IT - MITCOE Department of IT, Pune university- Pune University
Ahmednagar, Maharashtra, India - Pune, Maharashtra, India

Abstract **–Today NandFlash Memory(hence fort simply referred to as Flash) is used in handhold electronic devices like mobile, cameras, iPODS, music players, PDAS due to it's characteristics light weight, low power consumption, shock resistant and faster access. Today Flash is also used as an alternative storage medium for Hard Disk Drives (HDD) in PCs and Labtops. Though Flash has a faster access and is a good alternative storage media for HDD, it has some limitations too. Firstly Flash does not support update operation, unlike HDD which supports the basic operations directly .i.e. read, write and update, Flash, instead supports read, write and erase operations. Secondly Flash is neither byte addressable like Random Access Memory(RAM) nor it is sector addressable like HDD but instead the units of read, write and erase operations in Flash are uneven, it reads and writes in pages and erases in block. Third limitation of Flash is that it can not overwrite (update) data but it can write data only to an already erased place. i.e. Flash can write data only to a clean place, therefore an update must be preceded by erase operation in Flash Memory. Hence to update any data Flash does not write the data at the same place but it has to provide a mechanism to write the data to be updated at some clean (erased) place in Flash. This is called as "out of place update". To support the out of place update operation a middleware called as Flash Translation Layer (FTL) is accompanying the Flash memory, which resides inside a small controller that is mounted within the flash storage medium.**

Various trade offs arises in the design of FTLs, depending upon the various address mapping schemes used for logical to physical address translation.

Various researchers have invented different FTL design to optimize the performance while achieving the objectives and goals. This paper takes a review of all these FTL schemes and presents an innovative idea to improve the performance of the FTL presented earlier. The paper is organized into four sections. Section I is introduction. Section II takes review of different FTL schemes. Section III describes proposed innovative idea for FTL namely Meta paged Flash Translation Layer (MPFTL) . Section IV describes software design for proposed system. Section V describes the experimental setup required to test the results. Finally section V is conclusion.

I. INTRODUCTION

Address mapping or address translation is the functionality used to support out of place update operation in Flash. For an out of place update operation the functionality of address mapping or address translation works as follows: receives the logical address say La of the page to be updated from the host system (may be an operating system which is sending the read or write requests to Flash), write that page in some clean area in Flash say at physical address Pa, store the address La and its' corresponding physical address Pa into a table called as mapping table. It invalidates the pages containing the old data. This is shown below in figure 1.

Thus out of place update also requires frequently initiating the process of garbage collection to recycle all the invalid pages.

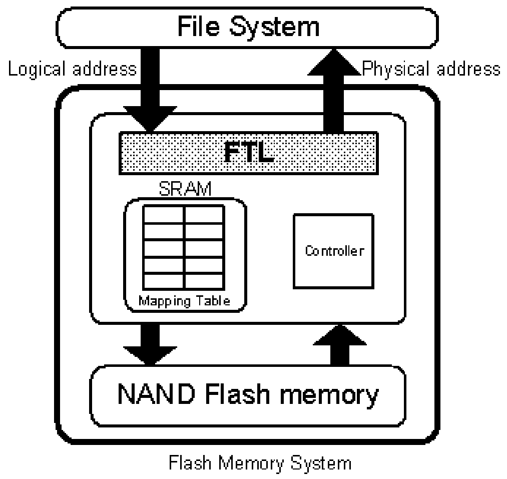

Fig. 1 Flash memory interface

Depending upon the granularity of mapping unit either page or block the mapping scheme has been categorized as page level, block level and hybrid mapping scheme. These schemes are described in next section.

II. REVIEW OF PREVIOUS FTL SCHEMES

A. Page level Mapping

This is a very flexible scheme in that a logical page can be mapped into any physical page in flash memory [1]. In addition to this feature, since it does not require expensive full merge operations described in the next subsection, it shows a good overall performance for both read and write operations. This intrinsic merit, however, brings about its critical demerit–large size of memory requirements. That is, the size of mapping table may be too large to be resided in SRAM of FTL. For example, let us consider a flash memory of 4GB size and a page size of 2 KB, this requires 2 million numbers of page mapping information to be stored in SRAM. In this case, assuming each mapping entry needs 8bytes, 16MB memory space is required only for the mapping table. This may be infeasible for the economic reasons.

B. Block level Mapping[2]

In a block level address mapping, a logical page address is made up of both a logical block number and its corresponding offset. This approach retains an inevitable disadvantage. When the overwrite operations to logical pages are issued, the corresponding block must be migrated and remapped to a free physical block as follows: The valid pages and the updated page of the original data block are copied to a new free physical block, and then the original physical block should be erased. When it comes to a block level mapping, this Erase-before-Write characteristic is an unavoidable performance bottleneck in write operations.

C. *Hybrid Mapping:[10]*

To overcome the shortcomings of the page level and block level mapping, a variety of hybrid schemes have been proposed. In these schemes the log blocks are used to temporarily record updates to improve the write performance.

1) BAST (Block Associative Sector Translation) scheme BAST[8] classifies blocks into two types, namely, data blocks for data saving and log blocks for overwrite operations. Each data block has associated with it a separate log block. Therefore this scheme suffers from low block utilization due to log block thrashing

2) F*AST (Fully Associative Sector Translation)* FAST[7] is based on BAST scheme but allows log blocks to be shared by all data blocks unlike BAST in which data blocks are associated to log blocks exclusively. This scheme subdivides log blocks into two types: sequential log blocks for switch operations and random log blocks for merge operations. Even though this accomplished better utilization of log blocks, it still remains in low utilization if overwrite operations are repeatedly requested only to the first page of each block. Moreover, random log blocks give rise to the more complicated merge operations due to fully associative policy.

3) Super Block scheme: Super Block Scheme[6] attempts to exploit the block level spatial locality in workloads by allowing the page level mapping in a superblock which is a set of consecutive blocks. This separates hot data (frequently updated data) and non-hot data into different blocks within a superblock and consequently the garbage collection efficiency is achieved thereby reducing the number of full merge operations. However, this approach uses a three-level address translation mechanism which leads to multiple accesses of spare area to serve the requests. In addition, it also uses a fixed size of superblock explicitly required to be tuned according to workload requirements and does not efficiently make a distinction between cold and hot data.

4) LAST (Locality-Aware Sector Translation): LAST[5] scheme adopts multiple sequential log blocks to make use of spatial localities in workload in order to supplement the limitations of FAST. It classifies random log buffers into hot and cold partitions to alleviate full merge cost. LAST, as the authors mentioned, relies on an external locality detector for its classification which cannot efficiently identify sequential writes when the small-sized write has a sequential locality. Moreover, the fixed size of the sequential log buffer brings about the overall garbage collection overhead.

*5) AFTL (Adaptive Two-Level Flash Translation Layer):*AFTL[3] scheme maintains latest recently used mapping information with fine-grained address translation mechanism and the least recently used mapping information with coarse-grained mechanisms due to the limited source of the fine-grained slots. Notwithstanding this two-level management, even though there are the large amounts of hot data, they all cannot move to fine-grained slots due to the limited size of fine-grained mechanism. That is, coarse-to-fine switches incur corresponding fine-to-coarse switches, which causes overhead in valid data page copies. Additionally, only if all of the data in its primary block appear in the replacement block, both corresponding coarse-grained slot and its primary block can be removed, which leads to low block utilization.

6) DFTL Scheme: DFTL[5] maintains two types of tables in SRAM, namely, Cached Mapping Table (CMT) and Global Translation Directory (GTD). CMT stores only a small number of page mapping information like a cache for a fast address translation in SRAM. Advantages and Disadvantages of DFTL: It achieves high block utilization, also it completely remove full merge operations. As a result, it improves overall performance and outperforms state-of-the-art hybrid FTLs in terms of write performance, block utilization, and the number of merge operations. However, DFTL suffers from frequent updates of translation pages in case of write dominant access patterns or garbage collection. To alleviate this problem, it uses delayed updates and batch updates in CMT with the aim of delaying the frequent updates. DFTL achieves a good write performance but cannot achieve as good read performance as hybrid FTLs under read dominant workloads due to its intrinsic two-tier address translation overhead. It costs an extra page read in flash when the request does not hit the CMT. Therefore DFTL cannot outperform hybrid mapping schemes using direct (i.e., one-level) address translation especially under randomly read intensive environments

III. PROPOSED SYSTEM

The proposed system is based on page level mapping. It is based on idea of storing the complete page map table in flash itself [1]. Initially it starts with block level mapping at the start up. Then complete flash is mapped using block mapping. This table is called as block map table, which is kept in SRAM i.e. in the controller's memory. But later on the mapping goes on switching from block level to page level as update request proceeds. This is done as follows: For read and write request the block mapping table is first searched and used if the block entry containing the requested page is found. But if update request is to be processed then a switch of mapping granularity takes place. In case of update request the block number of the requested page is calculated if the block entry is found in block mapping table all the pages in that block are switched to page mapping. The page map table is maintained in the Flash itself and the pages containing the mapping information in Flash are referred to as metapages. There is a directory maintained in RAM storing the addresses of these metapages. This directory is called as Metadirectory.

If for the issued read or write operation the corresponding block number entry is not found in block map table then only the page map table is concerned. For this the metapage i.e. page containing the mapping information is read into memory to locate the address of the requested page. Then the data is updated. This requires creating new metapage as mapping information for the requested page is to be changed. Hence it it as well requires invalidating the old metapage.

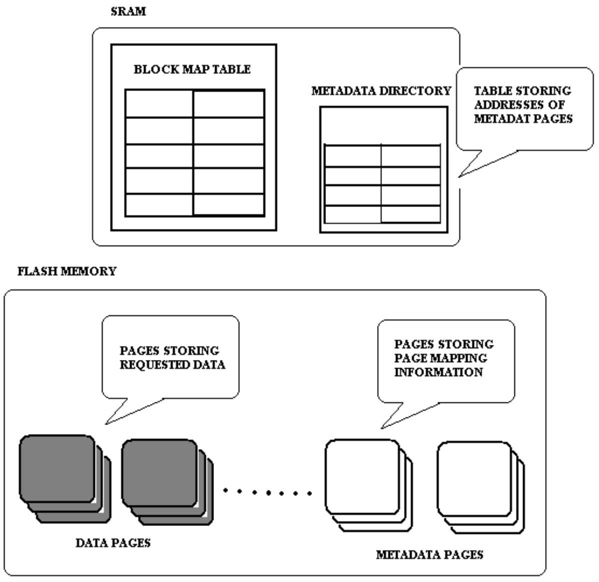
SRAM
BLOCK MAP TABLE
METADATA DIRECTORY
TABLE STORING ADDRESSES OF METADAT PAGES
FLASH MEMORY
PAGES STORING REQUESTED DATA
PAGES STORING PAGE MAPPING INFORMATION
DATA PAGES
METADATA PAGES

Fig. 2 Architecture of DPFTL

As seen earlier in hybrid schemes, log blocks eventually need to be erased and this will trigger a merge operation. A merge operation can be expensive and cause a number of log block erasures. Merge operations can be classified into three types: switch merge, partial merge, and full merge. A switch merge is triggered only when all pages in a block are sequentially updated from the first logical page to the last logical page. A partial merge is similar to the switch merge except for additional valid page copies. A full merge requires the largest overhead among merge operations. An FTL allocates a free block and copies the all valid pages either from the data block or from the log block into the free block. After copying all the valid pages, the free block becomes the data block and the former data block and the log block are erased. A variety of hybrid mapping schemes has been proposed.

A. Scope

NAND Flash is used as portable storage medium for computers, digital cameras, cell phones and other devices. In consumer devices, flash memory is widely used in:

- Notebook computers
- Personal computers
- Digital cameras
- Cell phones
- Solid-state music players such as Electronic musical instruments
- Embedded computers, Solid-state disk drives
- MP3 players
- Television set-top boxes
- Security systems
- Military systems
- Retail management products, Medical products
- Networking and communication products
- Wireless communication devices

In all the NAND Flash applications FTL is used required for doing the logical to physical mapping of data sectors/pages/blocks i.e. whatever is the concerned unit.

B. Goals

The goals of the project are: Reduce garbage collection overhead over the existing FTL schemes,. reduce the mapping table size over the existing FTL schemes, enhance the endurance of NAND Flash, enhance space utilization of NAND flash.

C. Constraints

Power-off recovery: When a sudden power-off event occurs during FTL operations, FTL data structures should be preserved and data consistency should be guaranteed

D. Steps to acquire input

The set of total I/Os issued by some realistic workload running on some computer is the input for this project. This I/O load is in the form of a text file called as trace file. The trace files for various categories of workload are available on the website of Storage Performance Council (CPU).

In order to create a list of the I/O parameters, the characteristics of the representative workloads are analyzed by SPC. To accomplish this, the individual I/O commands issued by the host processor(s) are collected and analyzed. Since many different analysis programs may be used, and since these programs will in all probability be run “after the fact”, a goal of the SPC is to collect I/O trace data from various systems for later analysis.

With the variety of hardware platforms, operating systems, and data collection techniques that exist, analysis of the collected trace data would be next to impossible without a well-defined common file format for the traces.

The trace file has been constructed with the goal to specify both a necessary and sufficient amount of information that is sufficient to allow reproduction of the essence of the original workload. This goal is constrained by the realization that not all relevant data may be collected from all operating systems, and that finite resources exist for collection, storage, and analysis of the trace data. Two I/O traces from OLTP applications running at two large financial institutions are available from web sites. These traces are made available courtesy of Ken Bates from HP, Bruce McNutt from IBM and the Storage Performance Council (SPC). These traces have been downloaded as trace files. The trace files can also be generated using Diskmon tool.

E. Steps to process input

FlashSim[11] is an event-driven simulator that follows the objected-oriented programming paradigm for modularity. The supplied input trace file is first processed by the program to generate a list of events, which then serves as the input for the FTL. The NAND Flash simulator is written as a single-threaded program in C++ for simplicity. C++ could provide a comprehensible object-oriented scheme where each class instance represented a hardware or software component. The Flash Simulator is integrated with Disksim’s C code

VI. Software Design

A. Classes in the simulator for hardware c components

SSD: The SSD class serves to provide an interface to Disksim and provide a single class to instantiate in order to create the simulator module. The SSD class creates event objects to wrap the Disksim *ioreq event* structures and returns the event time to disksim.

Package: The package class represents a group of flash dies that share a bus channel. The package class allocates its dies in its constructor and connects the dies to a bus channel. The package also facilitates addressing.

Die: A die is a single flash memory chip that is organized into a set of planes. Dies are connected to bus channels, but individual planes contained in the die buffer bus transfers. The highest level at which merge operations may take place should be at the die level. The corresponding event object is updated with the merge delay time.

Plane: Planes are comprised of blocks and provide a single page-sized register to buffer page data for bus transfers. The register is also used as a buffer for merge operations inside planes. The corresponding event object is updated with merge delays for merge operations and considers register delays.

Block: A block is comprised of pages and is the smallest component that can be individually erased. When a block is erased, all pages in it are erased and can then be written to again. The corresponding event object is updated with the erase delay time. A block can only be erased a finite number of times because of reliability constraints.

Page: Each page maintains its state and updates event objects with the read and write, delays of the given flash technology. Page states include free/empty after erasure, valid after a successful write, and invalid after being copied to a new location in a merge operation.

Controller: The controller class receives event objects from the SSD and consults the FTL regarding how to handle each event. The controller sends the virtual data for events to the RAM for buffering before sending the event object to the bus.

• *RAM:* The RAM class calculates how long it takes to read or write data to itself. The RAM buffers virtual event data for the controller to send across the bus.

• *Bus:* The bus class has a number of channels that are each shared by all the dies in a package. The bus examines addresses in events and passes the event object on to the proper channel.

• *Channel:* Channels must schedule usage for events and update the event time values. Each channel keeps a scheduling table that keeps track of channel usage, and new events are scheduled at the next available free time slot after dependencies have been met. The scheduling table size is synonymous to queue length.

B. Classes in the simulator for Software Components

Event: First, the event class keeps track of its corresponding Disksim I/O request event structure. Second, the event class holds methods and attributes to do all the record-keeping for the simulator's state, including SSD addresses. Simulator objects pass event class objects and update the event objects statistics.

Address: Addresses are comprised of a separate field for each hardware address level from the package down to the page. An address class instead of a *struct* to help make a clear interface to assign and validate addresses can be provided

FTL: The FTL provides address translation from logical addresses to physical addresses. It determines how to process events that involve many pages by producing a list of single-page events to be processed in-order by the controller. The FTL is responsible for taking advantage of hardware parallelism for performance. The FTL also has a wear leveler and garbage collector to facilitate its tasks.

Wear Leveler: The wear leveler class helps spread the block erasures over all blocks in the SSD. The wear leveler is responsible for keeping as many blocks functional for as long as possible because blocks of pages can only be erased for reuse a finite number of times.

Garbage Collector: The garbage collector is activated when a write request cannot be satisfied because the selected block is not writable or there is not enough free space in the selected block. The garbage collector seeks to merge partially-used blocks and free up blocks by erasing them.

The SSD simulator is instantiated as a SSD object designed to accept i/o request structures from Disksim. The SSD controller uses the FTL software module to create a list of events for a page request. The controller issues each event in the list to the data hardware through corresponding bus channels. The bus channels handle the scheduling and interleaving of events for the controller; this will simplify the controller implementation. Events continue through the package and are handled starting at the die level; merge events can be handled inside flash dies or planes. Erase events are handled inside blocks, and read and write events are handled inside pages. The SSD and package components are included in the call stack after consulting the bus channel because these components also keep track of wear statistics. Wear statistics stored in the SSD, package, die, plane, and block are updated every time an erase event occurs to keep a simple interface with lower algorithmic complexity for the FTL

C. Database design

Database is in the form of trace file. In order to create a list of the I/O parameters associated with a specific benchmark, the characteristics of the representative workloads should be analyzed and well understood. To accomplish this, the individual I/O commands issued by the host processor(s) are collected and analyzed. Since many different analysis programs may be used, and since these programs will in all probability be run "after the fact", a goal of the SPC is to collect I/O trace data from various systems for later analysis format. The trace file is composed of variable length ACSII records, rather than binary data. Although this format is somewhat wasteful of storage space and places higher CPU demands on analysis programs, it offers many advantages from a legibility and portability standpoint. Each record in the trace file represents one I/O command, and consists of several fields

The following is an example of the first few record of a trace file:

0,20941264,8192,W,0.551706,Alpha/NT
0,20939840,8192,W,0.554041

The individual fields are separated by a comma (hex 2C), with the trace record being terminated by a newline character (\n).

There is no special end-of-file delimiter; that function being left to the individual operating systems.

D. Implementation

The project is implemented using an SSD simulator. This simulator **[11]** is designed object oriented language and can be integrated with any FTL designed by the testers. Initially the project will create an SSD object, which will process the read and write requests. These read and write requests provided to the SSD are in the form of a text file called as trace file, whose format is described in earlier section. The SSD object requires a controller which. A controller uses an FTL. This FTL used will be the FTL designed by the simulator

V. Experimental Setup

For experimental result simulation of a 32GB NAND flash memory with configurations shown in table below is done.

TABLE I
FLASH MEMORY CONFIGURATION

Input Parameters	**Values**
Page read to register	25μ
Page write from register	200μ
Block Erase	1.5 ms
Serial Access to register data	50μ
Page size	4KB
Data register size	2KB
Block size	256KB

To test all the cases of workloads various types of workloads are selected as shown in table below. Websearch3 and financial1 and financial2 traces are made by Storage Performance Council (SPC). Websearch3 is a good read intensive I/O trace.

TABLE II
TYPES OF WORKLOADS

Workloads	**Requests Ratio (Read : Write)**	**Inter-arrival Time (avg.)**
Websearch3	R:99% W: 1%	70 ms
Financial1	R:22% W:78%	8 ms
Financial2	R: 82% W:18%	11 ms
Random_read	R: 99% W:1%	11 ms
Random_write	R:10%W: 90%	11 ms

Financial1 shows a good write intensive workload and is collected from an OLTP application running at a financial institution.

VI. Conclusion

Proposed idea i.e. Meta Paged Flash Translation Layer (MPFTL) is page level mapping scheme and avoids recursive merge and full merge required during garbage collection [1**]**. IT achieves high block utilization and hence improves overall performance.

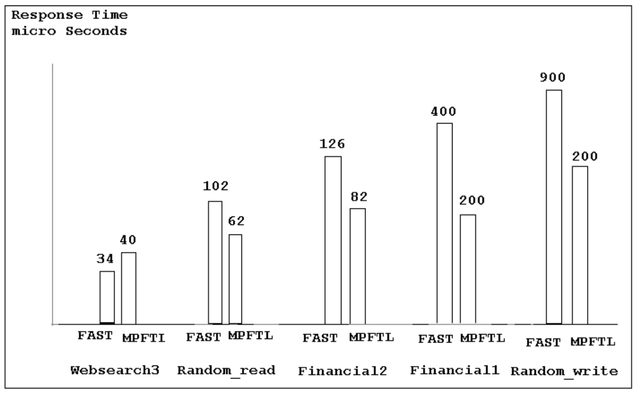

Fig. 2 Response time

From the response time got from the various workload it is observed that the proposed innovative algorithm outperforms the FAST scheme. FAST is taken as a representative scheme for comparison as it is has been considered an optimal one among the various FTL schemes that are available. The merge operation required during garbage collection specially in random write workload is eliminated and the random write performance is improved very much in case of the proposed scheme i.e. Mata Paged FTL (MPFTL). However in case of sequential read the MPFTL does not outweighs the FAST scheme but gives a comparative performance with FAST i.e. in case of the workload websearch3 both the schemes have near about same performance.

The future scope of this project is improve better data structure for managing the page table, so that the look time for the page search operation is improved.

References

[1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," in ASPLOS, 2009.

[2] Liu D., Wang Y., Qin Z., Shao Z., Guan Y.: A Space Reuse Strategyfor Flash Translation Layers in SLC NAND Flash Memory Storage Systems, IEEE Transactions on Very Large Scale Integration
(VLSI) Systems, pages 1-14, May – 2011, ISSN: 1063-8210 Volume: PP Issue:99

[3] Hsin Hung Lin, "An Adaptive Flash Translation Layer for High-Performance Storage Systems" IEEE Transactions on page: 953 – 965, June 2010, ISSN: 02780070, Volume: 29

[4] Shin I.: Light weight sector mapping scheme for NAND-based block devices, IEEE Transactions on Consumer Electronics, pages: 651 – 656, May 2010, ISSN: 0098-3063 Volume: 56 Issue:2

[5] 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), Feburary 2008.

[6] J. Kang, H. Jo, J. Kim, and J. Lee. A Superblock-based FlashTranslation Layer for NAND Flash Memory. InProceedingsof the International Conference on Embedded Software (EM-SOFT) , pages 161–170, October 2006. ISBN 1-59593-542-8.

[7] S. Lee, D. Park, T. Chung, D. Lee, S . Park, and H. Song. A Log Buffer based FlashTranslation Layer Using Fully Associative

Sector Translation. IEEE Transactions on Embedded Computing Systems, 6(3):18, 2007. ISSN 1539–9087.

[8] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of Flash Translation Layer," J. Syst. Archit., vol. 55, no. 5-6, 2009.

[9] UMass, "Websearch Trace from UMass Trace Repository," http://traces.cs.umass.edu/index.php/Storage/Storage, 2002.

[10] Chung, D. Park, S . Park, D. Lee, S. Lee, and H. Song. System Software for Flash Memory: A Survey. In Proceedings of the International Conference on Embedded and Ubiquitous Computing, pages 394–404, August 2006.

[11] "FlashSim: A Simulator for NAND Flash-Based Solid-State Drives" in Advances in System Simulation, 2009. SIMUL '09. First International Conference on pages 125-131 ISBN: 978-1-4244-4863-0