VSSIM: Virtual Machine based SSD Simulator

Jinsoo Yoo*, Youjip Won*, Joongwoo Hwang*, Sooyong Kang*, Jongmoo Choi†, Sungroh Yoon‡ and Jaehyuk Cha*
*Hanyang University, Seoul, Korea
Email: {jedisty | yjwon | tearoses | sykang | chajh}@hanyang.ac.kr
†Dankook University, Yongin, Korea
Email: choijm@dankook.ac.kr
‡Seoul National University, Seoul, Korea
Email: sryoon@snu.ac.kr

***Abstract*—In this paper, we present a virtual machine based SSD Simulator, VSSIM (Virtual SSD Simulator). VSSIM intends to address the issues of the trace driven simulation, e.g. trace re-scaling, accurate replay, etc. VSSIM operates on top of QEMU/KVM with software based SSD module. VSSIM runs in real-time and allows the user to measure both the host performance and the SSD behavior under various design choices. VSSIM can flexibly model the various hardware components, e.g. the number of channels, the number of ways, block size, page size, planes per chip, program, erase, read latency of NAND cells, channel switch delay, and way switch delay. VSSIM can also facilitate the implementation of the SSD firmware algorithms. To demonstrate the capability of VSSIM, we performed a number of case studies. The results of the simulation study deliver an important guideline in the firmware and hardware designs of future NAND based storage devices. Followings are some of the findings: (i) as the page size increases, the performance benefit of increasing the channel parallelism against increasing the way parallelism becomes less significant, (ii) due to the bi-modality in IO size distribution, FTL should be designed to handle multiple mapping granularity, (iii) hybrid mapping does not work in four or more way SSD due to severe log block fragmentation, (iv) as a performance metric, the Write Amplification Factor can be misleading, (v) compared to sequential write, random write operation can be benefited more from the channel level parallelism and therefore in multi-channel environment, it is beneficial to categorize larger fraction of IO as random. VSSIM is validated against commodity SSD, Intel X25M SSD. VSSIM models the sequential IO performance of X25M within 3% offset.**

I. Introduction

In 2011, worldwide market size of NAND flash device became larger than the worldwide market size of DRAM for the first time ever[1] and NAND flash worldwide usage is expected to grow at 250% CAGR during 2011-2015 time frame[2]. As NAND flash based storage device quickly gains a momentum as primary storage device, more efforts are being dedicated to properly understand the trade-offs between various SSD design parameters.

The study on modeling and simulation has always been an important tool to give the insight in the internal behaviors of the storage device, to understand the design trade-offs, to narrow down the design spaces, and to reduce the prototyping efforts. Developing an SSD is not an exception. Although hardware implementation produces the realistic results, it is not only time consuming but also expensive to implement fully operating FTL, garbage collection algorithm, wear-leveling algorithm and etc. with physical NAND flash chip, which one prefers not to implement if there is a resort. From practitioner's point of view, it is critical that every aspect of the SSD controller is meticulously and thoroughly examined, tested and validated before tapeout[1].

This research is motivated by the simple yet intense demand from the practitioners: "Can we observe how the host performs in on-line manner with yet-to-come SSD?". Behaviors of the host ranges from a simple IO performance, e.g. bandwidth and IOPS, to more complicated one, e.g. time to install a software(for mobile device and PC), database transactions/min(`tpmC`)(for enterprise server), SQLite operations/sec(for Smartphone) and etc. The research community has come up with many seemingly efficient FTL algorithms and novel controller architectures. Fair amount of sophisticated tools have been proposed to study the effectiveness of the newly devised ideas. They include software based simulators [3], [4], [5], [6], [7], [8], [9] and hardware based emulators [10], [11], [12], [13]. Software based simulation tools can be further categorized into trace driven simulators, which run in off-line and virtual device based simulators, which are attached to the host as software based block device, e.g. `ramdisk`, and run in on-line. Trace driven simulators[3], [5], [7], [9] can model the SSD internals in detailed manner and are widely used to study the behavior of the individual components of the storage device. However, we cannot observe the real-time host performance in trace driven simulation. When we use the same trace across different target platforms for performance study, e.g. Smartphone and notebook computer, the rescale/replay issues need to be addressed: (i) the intervals between the trace entries need to be rescaled for the individual target platform properly incorporating the think-time, causal dependency and concurrency[14] and (ii) the IO request in the rescaled trace should to be issued at accurate timing specified at the trace[15], [16], [17], [18]. More serious issue in trace driven simulation is the need for rescaling address footprint in the trace. For the simulator results to be meaningful, the size of the simulated storage device should match the trace footprint. We cannot linearly scale the addresses in the trace since it will disintegrate the access locality embedded in the trace. The garbage collection, wear-leveling, victim selection for log block merge

VSSIM is publicly available at http://esos.hanyang.ac.kr/vssim

978-1-4799-0218-7/13/$31.00©2013 IEEE

[1]Producing approximately fifty sample chips costs well over 100K USD ($7mm \times 7mm$, shuttle run, 65 nm process). For massive production, single run costs over 1M USD.

algorithms of SSD are designed to exploit the access locality and exhibit widely different behavior subject to the locality. Without elaborate treatment on these issues, it is not possible to warrant the correctness of the trace driven simulation.

TABLE I. COMPARISON BETWEEN VSSIM AND OTHER SSD SIMULATORS (◯: SUPPORTED, △: SUPPORTED WITH A H/W LIMITATION, ×: NOT SUPPORTED)

	Measure Host perf.	Firmware change	# of Channel	# of Ways	NAND latency
Disksim	No	◯	◯	◯	◯
Nandsim	Yes	×	×	×	◯
Flashsim	No	◯	◯	◯	◯
Openssd	Yes	◯	△	△	×
Bluessd	No	◯	△	△	×
VSSIM	Yes	◯	◯	◯	◯

Virtual device based simulators[4], [6], [8] appears to the host as software based block device or filesystem extension similar to `ramdisk`. We can observe the performance of the host in online manner with virtual device based simulator. However, existing virtual device based simulators are very primitive. They do not model multi-channel/multi-way architecture, write buffer and page map cache, to list a few. Due to their simplicity, the existing virtual device based simulators have very limited application in studying the performance behavior of the storage device.

Hardware based emulator[10], [11], [12], [13] is the most accurate way of examining real-time behavior of SSD. Due to its intrinsic inflexibility, it cannot be used to examine the variety of hardware design choices, e.g. different NAND devices(latency, the number of pages per block, page size), different number of channels/ways, bus clock and etc.

In this work, we develop VSSIM, the **V**irtual Machine based **S**SD **Sim**ulator. VSSIM successfully addresses the original question: "Can we observe how the host performs in real-time under yet-to-come SSD? ". The contribution of VSSIM can be summarized as follows:

- **Detailed and Flexible Model**: VSSIM provides fairly detailed and flexible SSD model. SSD model of VSSIM includes the hardware components, e.g. degree of parallelism, channel switch delay, NAND flash delay, and software components, e.g. address mapping algorithm, page table caching scheme, wear leveling algorithm. All of these components are modularized and parameterized for flexible model update.
- **On-line and Real-time Execution**: VSSIM is attached to the host as real storage device and runs in real time. User can observe how the host performs with a given SSD design. Delay engine of VSSIM keeps track of all concurrent NAND operations across the multiple NAND devices and introduces proper amount of delay for each IO request from the host.
- **Accuracy**: VSSIM accurately models the behavior of physical SSD. Accuracy of VSSIM is validated against existing commercial SSD(Intel X25M). Since VSSIM runs in real-time as storage device, it is robust against classic issues of trace driven simulation, e.g. accurate replay and trace scaling which make the process of performance study cumbersome and may result in less accurate test results.

VSSIM uniquely distinguishes itself from the existing simulator based and emulator based modeling approaches. VSSIM harbors all advantages of each modeling approach: concreteness and flexibility of trace driven simulator, on-line execution of virtual device based simulator and accuracy of hardware based emulator. VSSIM enables the researchers and practitioners to examine the SSD internal behavior and the respective host performance in a unified framework in real-time on-line manner. VSSIM makes the SSD development process extremely versatile and can significantly shorten the overall SSD development process.

II. BACKGROUND

A. NAND Flash Memory

In NAND flash, a fixed number of cells, which hold one (SLC NAND) or more (MLC or TLC NAND) bits, constitute a fixed size page. A fixed number of pages (e.g. 128 pages or 256 pages) form a block. A page is the unit of read and write operations and a block is the unit of erase operation. A chip is a collection of blocks. Some vendors use the plane, which is a collection of blocks in a chip. The objective of introducing the notion of plane is to interleave the write operations across the planes via allocating separate page register to individual planes.

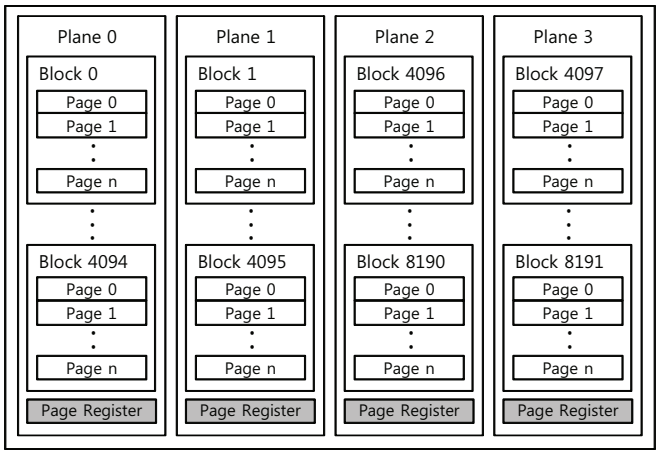


Fig. 1. Structure of NAND Flash Memory [19]

Figure 1 schematically illustrates the structure of NAND flash Memory [19]. Here, blocks are divided into four planes. There are three fundamental operations in NAND flash Memory: Read, Write, and Erase. A page in a block does not support in-place update. If we are to update a page, the page has to be erased before write. In the worst case, a block should be erased and entirely rewritten to update a single page on the block. SSD requires the technologies such as garbage collection and wear leveling to manage the NAND flash memory efficiently and economically.

B. SSD Architecture

Figure 2 depicts the internal structure of SSD. An SSD is composed of host interface, CPU, ROM, RAM, Flash Controller, and NAND flash chips. The CPU runs the FTL code stored in ROM to manage the overall operations. RAM is used as temporary Read and Write buffer to improve the

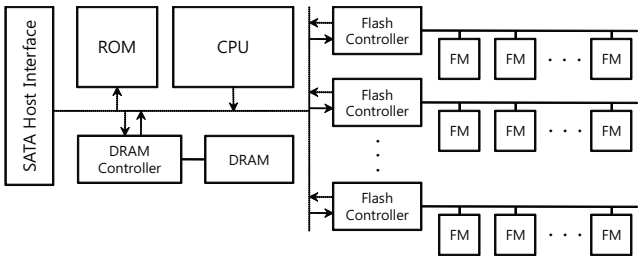


Fig. 2. Internal structure of SSD

performance of the storage device. It also saves various internal data structures for FTL, such as a mapping table, block status table which maintains key information for garbage collection, and wear leveling, among others. SSD uses multiple channels and multiple ways to exploit the parallelism in the system. The number of channels and ways defines the number of NAND operations that an SSD controller can process in parallel. The flash controllers can issue independent IO requests to each channel, simultaneously. Generally, the number of channels in an SSD is bounded by the tolerable peak current and the real estate availability in the SSD controller. The number of NAND flash memories that share the same channel corresponds the number of ways. SSDs can exploit parallelism across ways through interleaving IO commands among multiple flash chips in the same channel. The number of ways in a channel is generally bounded by the bus bandwidth of the channel. Exploiting the parallelism in the storage system is critical in accelerating the performance.

C. FTL (Flash Translation Layer)

FTL(Flash Translation Layer) [20] is responsible for exposing an array of logical blocks/pages to outside by providing Logical Block/Page Address (LBA/LPA) of each physical block/page in the SSD. It performs three main tasks: (i) maintains logical to physical address translation, (ii) occasionally collects and consolidates invalid pages, and (iii) evenly distributes the write and erase operations over the blocks. Numerous address translation techniques have been introduced, which can be categorized into block mapping, page mapping [21], and hybrid mapping [22], [23], [24], [25], [26]. Page mapping exhibits the best random write performance, but large mapping table size requirement needs to be resolved to make it practically feasible. Hybrid mapping categorizes the NAND blocks into two: data block and log block, where log block is for recording the updated data pages. It is called hybrid mapping since log block is maintained by page mapping while data blocks are maintained by the block mapping. It aims at delivering the page mapping like random write performance while reducing the mapping table. Fully Associative Sector Translation (FAST) [23] attempts to exploit the spatial locality of the workload. Locality Aware Sector Translation (LAST) [24] segments the random log block even further exploiting temporal locality. In hybrid mapping schemes, the data and log block associativity is the critical factor for write performance. While Block Associative Sector Translation (BAST) [22] uses one-to-one associativity between data and log blocks, FAST uses many-to-one associativity to increase the log block utilization and decrease the log block merge operations. LAST divides log blocks into random and sequential log blocks, which are maintained by page and block mapping, respectively. It uses one-to-one associativity for sequential log blocks and many-to-one associativity for random log blocks, to fully exploit the access locality of the workloads. Reconfigurable FTL [25] uses many-to-many block associativity and claims that different block associativity should be used for different workloads. A more recent hybrid mapping scheme [26] tries to dynamically changing the regions managed by block mapping and page mapping respective subject to the varying workload characteristics.

Hybrid mapping scheme has three types of merge operations (switch merge, partial merge, and full merge) to reclaim log blocks. A single merge operation reclaims only one log block while invoking one (switch merge and partial merge) or more (full merge) block erase operations. The number of block erase operations involved in a full merge operation depends on the number of blocks which the pages in the erased log block is consolidated against, *log block associativity*. Technically, hybrid mapping can invoke approximately as many erase operations as the number of pages in a block, in which case the merge latency becomes prohibitively long. Page mapping scheme writes an updated page to any empty page in the SSD and updates the corresponding entry in the page mapping table to point to the newly written page. In page mapping, instead of log block reclamation, it performs data block reclamation. The block reclamation process in page mapping is called Garbage Collection. In garbage collection, the blocks with the most invalid pages are selected for reclamation to minimize the valid page copy overhead.

Another function of FTL is wear leveling. FTL tries to balance the erase count of each block to minimize the number of worn-out blocks. The hot/cold block identification is needed for wear leveling. One of the popular approaches is that FTL stores hot data to the cold (or young) block and cold data to the hot (or old) block to evenly distribute the erase count of individual blocks.

III. Organization of VSSIM

A. SSD model for QEMU/KVM

VSSIM is built upon QEMU[27]. QEMU is a processor emulator which can host various Operating systems, e.g. Linux, Windows and etc. QEMU provides IDE device which is a dummy device processing SATA command. VSSIM uses QEMU emulated IDE to construct an emulated SSD. We developed a detailed SSD model under IDE interface. VSSIM provides a graphical user interface based real-time monitoring tool to examine the status of VSSIM. To enhance the system performance, we chose to run the QEMU on KVM[28].

VSSIM SSD model consists of FTL module, IO emulator module, SSD monitor, and Latency Manager. FTL module accepts SATA command and generates the NAND operations(read, write, or erase) and passes them to IO emulator. FTL module imports address mapping algorithm, garbage collection algorithm, and wear-leveling algorithm. In VSSIM, garbage collection module is implemented as a separate thread. In VSSIM, various firmware algorithms(address mapping, garbage collection, and wear leveling) can be integrated into SSD module in versatile manner.

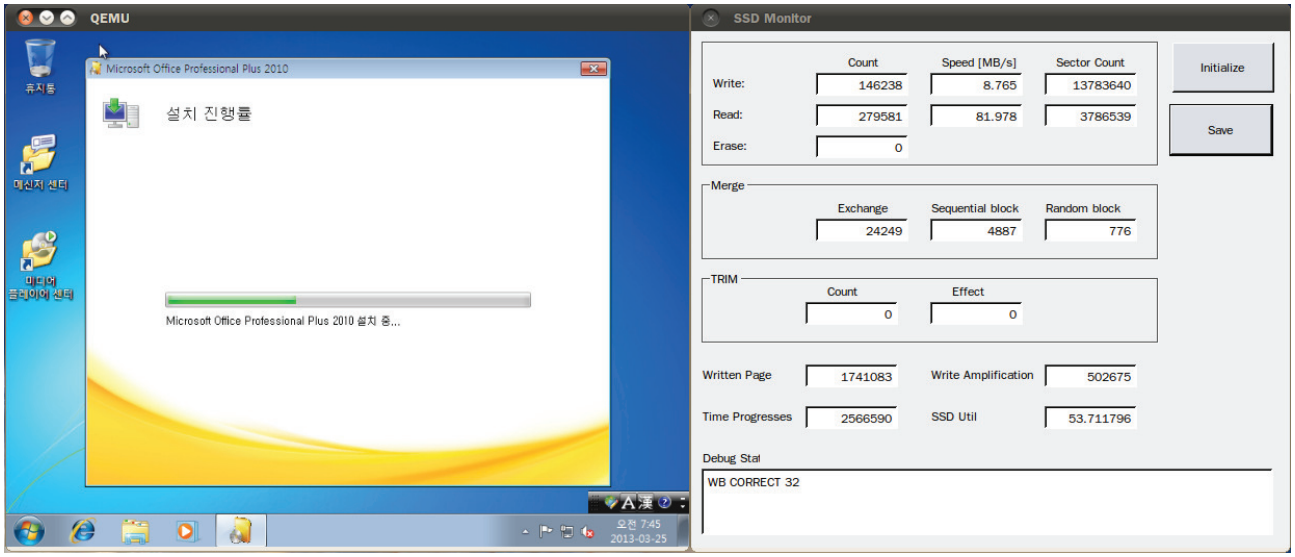


Fig. 3. VSSIM with SSD Monitor

VSSIM supports multi-channel, multi-way architecture with map cache(mapping table) and write buffer. SSD configuration file defines the hardware configuration of SSD: the number of channels, the number of ways, the number of planes(or banks), the size of map cache, and the size of write buffer. IO emulator receives NAND operation(read/write/erase) from FTL module, introduces the appropriate amount of latency, and performs actual NAND operation at the simulated Flash memory, e.g. ramdisk. IO emulator can record the information on the NAND operation which it has completed, e.g. type of IO, latency of IO, logical block number and etc. Since VSSIM is an emulator, the data is written to the designated device. As it stands currently, the VSSIM uses RAMDISK as storage medium. To emulate large size SSD, high performance SSD, e.g. ioDrive[2] and REVOdrive[3], can be used instead of main memory. We mmap the high performance SSD to memory address space and then we impose ramdisk on it so that it can be used as emulated SSD by VSSIM. With this method, the capacity of the emulated SSD is governed by the size of the high performance SSD, e.g. 512GByte. This approach works fine as long as the speed of the emulated SSD is slower than the physical SSD.

SSD monitor processes the traces recorded by IO emulator and generates real-time information on SSD device operation, e.g. IOPS, bandwidth, Write Amplification, the number of TRIM commands, the number of erase operations and etc. SSD monitor provides graphic user interface to visualize this information. Figure 3 illustrates the screen snapshots of VSSIM execution, where an application software is being installed.

Write Request: The number of write requests and total number of sectors written.

Read Request: The number of read requests and total number of sectors read.

Merge Operation: The number of each type of merge operations, switch/partial/full, performed when we use hybrid mapping FTL.

Garbage Collection: The number of garbage collections.

TRIM: The number of TRIM commands.

Write Amplification: The number of re-written pages by merge operation or garbage collection.

We can obtain the statistics on various behaviors of SSD, e.g. ratio of read and write requests, average request size, WAF(Write Amplification Factor, the ratio between the number of page writes from the host to the number of page writes which actually happen into flash memory) and etc.

One of the key ingredients of VSSIM is Latency Manager. Latency Manager is responsible for introducing right amount of delay to properly emulate the NAND operation. VSSIM is capable of specifying various delays in the emulated SSD: channel switch delay, way switch delay, latencies of NAND read/write/erase operations. Figure 4 illustrates the overall organization of VSSIM.

B. Allocating Free Block

FTL module maintains the status of the blocks, including mapping information and erase count. Block allocation module defines free block selection policy. When FTL module needs a new block, it calls block allocation module. In multi-channel/multi-way SSD, the performance of SSD varies widely subject to how to distribute the individual pages of a single write request among the channels, ways, or banks. VSSIM takes modular approach. We can flexibly change the block allocation policy.

FTL manages the metadata for each block: block type, wear level, number of valid pages in a block, validity of each page, and the page index for the most recently written page. Block type is one of `empty`, `data` or `log`, which corresponds to empty block, data block and log block in hybrid mapping. Wear-level and the number of valid pages in a block are also used in wear-leveling and garbage collection.

[2]http://www.fusionio.com/products/iodrive/
[3]http://ocz.com/consumer/pci-express-ssd

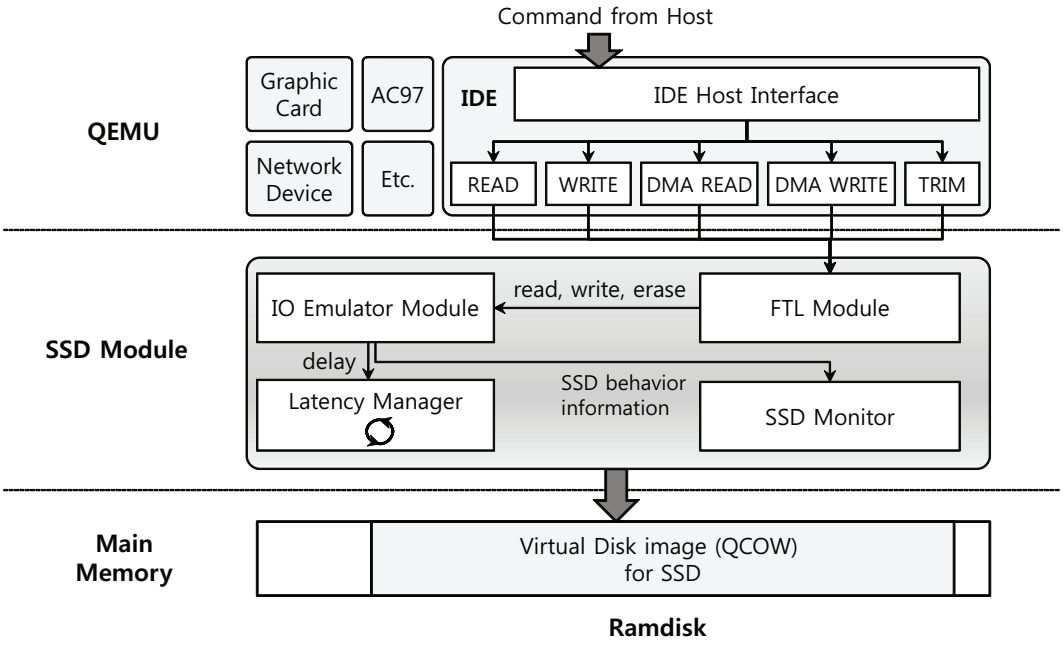


Fig. 4. VSSIM organization

C. IO emulator

IO emulator performs an actual NAND operation introducing proper amount of delay. VSSIM uses a file to maintain the IO data persistently. From SSD controller's point of view, process of reading and writing the data to and from NAND flash device consists of two phases: the data transfer between the buffer and the page register and the data transfer between the page register and the NAND device. We call the operations related to the former and the latter as channel operation and the NAND operation, respectively. There are two types of channel operations: *RegisterRead* and *RegisterWrite*, each of which corresponds to the operation of transferring the data from the page register to buffer and from the buffer to the page register respectively. The NAND operation has one of the three types: *CellRead*, *CellProgram*, and *Erase*. These operations correspond to the operations between the page register and the NAND device. We add prefix *Cell* to distinguish the NAND operations from the read and write operations from the host.

IO request from the host is split into the appropriate number of NAND operations. Each NAND operation accounts for a single page IO, e.g. 8 KByte. For read operation, address mapping mechanism determines the physical location of the requested data. For write operation, block allocation policy determines where to designate the write operation. Subject to the block allocation policy, multiple NAND write operations can be directed to the same physical block or can be interleaved across the multiple blocks across the channels.

IO emulator keeps track of all concurrent NAND flash operations. It maintains the *status* and the *timer* for each channel and for each flash memory (or for each plane of flash memory). For the channel, the status denotes the type of the last command (*RegisterRead* vs. *RegisterWrite*) and the timer indicates the time when the respective command has initiated. For each flash memory (or each plane of flash memory), *status* denotes the type of the last flash memory operation (*CellRead*, *CellProgram* or *Erase*). These fields are used to synchronize the various channel and NAND operations guaranteeing correctness.

Pseudocode 1 Functions for Flash

```
1:  procedure RegisterWrite(F_n)
2:      Ch_n ← F_n mod N_channel
3:      ChannelAccess(F_n)
4:      chStatus[Ch_n] ← REG_WRITE
5:      chTimer[Ch_n] ← timestamp
6:      return
7:  end procedure

8:  procedure CellProgramming(F_n, B_n)
9:      IOWait (F_n, B_n)
10:     F_n ← F_n * 2
11:     plane ← B_n mod 2
12:     fmStatus[F_n + plane] ← PROGRAM
13:     fmTimer[F_n + plane] ← timestamp
14:     return
15: end procedure

16: procedure BlockErase(F_n, B_n)
17:     IOWait (F_n, B_n)
18:     F_n ← F_n * 2
19:     plane ← B_n mod 2
20:     fmStatus[F_n + plane] ← ERASE
21:     fmTimer[F_n + plane] ← timestamp
22:     return
23: end procedure
```

Write operation consists of two serial tasks: transferring data from the buffer to page register and programming the respective page, each of which corresponds to `RegisterWrite()` and `CellProgramming()`, respectively. `Write` operation receives flash number (F_n) and block number (B_n). When transferring the data to page register, SSD module first checks whether the channel is available.

`ChannelAccess()` function determines if the channel is free and waits until the channel becomes available. When the channel becomes available, status and the timer of the respective channel are updated to *REG_WRITE* and current time. When it finishes transferring data to page register, write operation starts `CellProgramming()`. It first checks if a given flash chip is available and waits until the currently on going operation completes. `IOWait()` is responsible for this task. When the currently ongoing operation completes, `CellProgramming()` updates the status and the timer of the flash memory to *PROGRAM* and current time. Note that our implementation takes account of the plane type of each chip. A block is associated with even or odd plane depending on the block number. When it saves the status of a block and type of a command, it considers the planes. A chip can perform two operations at the same time by interleaved manner.

To access a channel, the channel access module examines the current status of the channel and the timer. Based upon the configuration parameter of VSSIM (e.g. RegisterWrite = 800 usec), channel access module determines if the last channel operation has finished. If it has not completed yet, the channel access module waits until the last channel operation completes.

D. Latency Manager

One of the key technical ingredients of VSSIM is accurate IO latency. Latency Manager is responsible for introducing accurate amount of delay. Latency Manager incorporates not only the latencies introduced by various hardware components, e.g. data transfer between NAND cell and page register, channel/way switch delay, but also the overhead of VSSIM itself to determine the latency it is going to introduce.

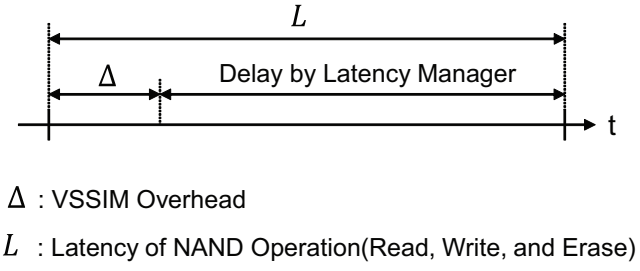


Fig. 5. IO Delay Management Method

Figure 5 illustrates the way in which Latency Manager introduces the delay. It determines the total delay which needs to be introduced, e.g. L=950μsec for NAND write operation, and determines the overhead of VSSIM, e.g. Δ=20μsec. The VSSIM overhead may include CPU overhead of VSSIM operation and the time to access data at the storage. Latency manager introduces a certain amount of delay that total latency in VSSIM precisely matches the latency in the real NAND operation. As long as the VSSIM overhead Δ is less than NAND IO latency, L, VSSIM can simulate the given NAND based flash storage. VSSIM uses busy-wait to introduce the latency. READ and PROGRAM operation of NAND flash take 50 - 200 μsec and 900 - 2000 μsec, respectively. According to our measurement, a single iteration of busy wait loop including system call overhead of `time()` takes 76 nsec. The granularity of the busy waiting is at least two orders of magnitude smaller than the flash memory operations. Time resolution of busy waiting is fine enough to model the latency of flash command accurately. Also, the CPU overhead of VSSIM is small enough compared to the latency of NAND flash operations. VSSIM uses RAMDISK or high performance storage device[4] to store actual data so that VSSIM overhead is smaller than the latency of NAND flash operations.

E. Firmware Components

In VSSIM, we have implemented three address mapping algorithms: page mapping and two hybrid mappings [23] [24]. FTL module of VSSIM exports a set of well-defined interfaces to easily port the new address mapping algorithms. FTL module implements three block deallocation(TRIM) algorithms for each FTL that VSSIM has implemented. In page mapping, block deallocation manager immediately scans the page table and invalidates the pages upon receiving the block deallocation command(TRIM). In hybrid mapping scheme, FTL module maintains the list of invalid blocks supplied by TRIM command. The blocks specified by the TRIM command are invalid ones from the host's point of view, e.g. blocks belonging to the deleted file. SSD controller does not have knowledge on whether a given NAND flash page contains valid data for the filesystem. The NAND flash pages which has belonged to the deleted file are still valid. We call the list of invalid blocks informed by the TRIM command as TRIM list to distinguish it from the list of invalidated blocks in SSD. FTL consults TRIM list when performing merge and switch merge operations. If a page of the log block is in the TRIM list, it is regarded as invalid page and is excluded from the set of pages moved to the new block. It is possible that incoming write operation is for the page in the TRIM list. This may happen when the file system reuses the data block which belongs to the deleted file. Then, SSD controller allocates new flash page for the incoming write request. The logical page is removed from the TRIM list and the respective physical page is invalidated.

VSSIM triggers garbage collection when the available free blocks goes below a certain threshold value (currently 30%). It applies "block with the most invalid pages first" policy[29]. New garbage collection module can be easily plugged into VSSIM. VSSIM keeps track of erase and write count for each block, and utilizes it for wear leveling. When FTL requests an empty block to Block Manager, it selects the channel from which the new block is allocated in round robin manner, and within a channel a block with lowest wear level is returned.

F. Hardware Components

VSSIM parameterizes the NAND flash characteristics. The parameters include page size, the number of pages in a block, the number of blocks in a chip, the size of metadata for each page, read latency, program latency and erase latency. Also, we can specify the maximum number of erase cycles as input parameter. VSSIM parameterizes the data transfer time between the host and the page register. VSSIM can configure the SSD with arbitrary numbers of channels, ways and the sizes of NAND chips and packages. Also, it can specify the channel switch delay and way switch delay. Table. II shows the VSSIM hardware configuration parameters.

[4]Fusion IO, Revo drive

TABLE II. VSSIM SSD CONFIGURATION

Parameter	Description
PAGE_SIZE	The size of a NAND flash page
SECTOR_SIZE	The size of a sector
FLASH_NB	The total number of flash chips
BLOCK_NB	The number of blocks in a flash
PAGE_NB	The number of pages in a block
CHANNEL_NB	The number of channels
WAY_NB	The number of ways
PLANE_PER_FLASH	The number of planes in a flash
LOG_RAND_BLOCK_NB	The number of rand log block in a flash
LOG_SEQ_BLOCK_NB	The number of seq log block in a flash
REG_WRITE_DELAY	1 Page register write latency
REG_READ_DELAY	1 Page register read latency
CELL_PROGRAM_DELAY	1 NAND flash page write latency
CELL_READ_DELAY	1 NAND flash page read latency
BLOCK_ERASE_DELAY	1 NAND flash block erase latency
CHANNEL_SWITCH_DELAY_R	Channel switching delay for a read
CHANNEL_SWITCH_DELAY_W	Channel switching delay for a write
CACHE_IDX_SIZE	Total index size of the map cache
WRITE_BUFFER_SIZE	The size of write buffer(KByte)

IV. CASE STUDIES

A. Overview

We study the various aspects of SSD behavior and the respective host performance by varying the hardware configurations (NAND page size, IO latency and degree of parallelism) and the controller algorithms (FTLs, TRIM command). In this study, we use sequential and random IO as well as application workloads such as operating system and application installation. We built five different 16 GByte SSDs with VSSIM. Three of them use eight 2GByte Flash chips(2KByte page) and two of them use four 4GByte Flash chips(8 KByte page size). Table III summarizes the configuration of SSDs used in our study. SSD-A, SSD-B, and SSD-C use the same 2GB Samsung chip [19]. They are configured as 8 by 1, 4 by 2 and 2 by 4 configurations[5], respectively. SSD-D and SSD-E use 4GB Intel chip [30]. They are configured as 4 by 1 and 2 by 2, respectively. For each SSD, we implemented three FTLs (page mapping, FAST and LAST). Each FTL has TRIM capability.

TABLE III. SPECIFICATION OF SSDS (16 GBYTE)

SSD Label	A	B	C	D	E
No. of Flash	8			4	
Channel	8	4	2	4	2
Way	1	2	4	1	2
Flash Model	K9LAG08U0M			JS29F32G08	
Flash Vendor	Samsung [19]			Intel [30]	
Flash size	2 GByte			4 GByte	
Block size	256 KByte			1 MByte	
Page size	2 KByte			4 KByte	
Chnl Clk/Width	33MHz/8bit			50MHz/8bit	
Read	60 usec			50usec	
Program	800usec			900 usec	
Erase	1.5 msec			2 msec	

Before we move on, let us briefly revisit the basics of address mapping schemes used in this study. In page mapping, each logical page can be mapped to any physical pages in SSD. Since modern NAND based storage device can easily go over several hundreds of GByte, the page table size of page mapping puts significant burden the storage controller which has very limited memory. In block mapping, mapping happens at the block granularity and therefore the overhead of mapping table is orders of magnitude smaller than the overhead in page mapping. In this study, we examine the behavior of two hybrid mappings, FAST [23] and LAST [24] along with the page mapping. In hybrid mapping, there are two types of blocks: data block and log block. Only data blocks are visible to upper layer. In hybrid mapping, data blocks are maintained by block mapping and the log blocks are maintained by page mapping. For an incoming write request, hybrid mapping first consults block mapping table to find the physical address. If the respective physical page contains valid content, FTL invalidates the respective location and writes an incoming page to one of the log blocks.

When there run out of free blocks, FTL selects one or more blocks, consolidates valid pages in these block into a new block, and erases the blocks whose pages are consolidated. Each of hybrid mapping schemes has its own way of categorizing the log blocks and managing them. To make the consolidation more efficient, FAST partitions the log blocks into random and sequential log blocks, where the random write and sequential writes are directed, respectively. The problem of FAST is its high log block associativity: a single log block can harbor the data pages from all different data blocks. This makes the consolidation cost extremely high especially in multi-channel/multi-way environment. LAST has been proposed to effectively address this issue. LAST further categorizes the random log blocks into *hot* and *cold* ones. It establishes a time window and if the write request for the same logical page arrives again within this time interval, it is directed to *hot* log block. It significantly reduces the log block utilization and equivalently the overhead of log block merge operation. The objective of this study is not evaluating the mapping algorithms. Rather, we like to examine how the tool developed in our work can benefit the understanding on the various behaviors of SSD. Further interested users are referred to [23], [24].

We chose two scenarios (copying a video file and MP3 files) for sequential IO and IOzone to measure the performance of random IO. We also used five application installation workloads (Windows7, MS Office, Photoshop, Ubuntu, and Xen Compile). We have described the workload characteristics in Table IV. These workloads are carefully chosen to properly incorporate the modern computer and mobile device usage.

B. Model Accuracy

VSSIM is accurate.

A fair number of simulation tools have been proposed to study hardware and software aspect of NAND based storage device. Few of the simulation tools have been validated their accuracy against the commodity NAND based storage device. Flashsim [5] is the only one which we are aware of to compare the performance of the simulated device against the real storage device. To validate VSSIM with real SSD, it is mandatory that we acquire detailed understanding on the behavior of a given SSD. With few exceptions, SSD vendors do not disclose the SSD firmware algorithms and often treat them as trade secret. Yoo et.al. [31] successfully identifies the internals of X25M in fairly detailed manner: (i) X25M adopts plain page mapping, (ii) It parallelizes the write operation in 4 KByte granularity, and (iii) channel switch entails 30 μsec

[5] m by n denotes m channels with n way configuration

TABLE IV. USER SCENARIO WORKLOAD CHARACTERISTICS

	Total Request	Read / Write Ratio (%)	Read / Write size (GB)	Seq. Write (%)
Windows7	286,472	62.1 / 37.9	1.81 / 10.65	45.9
MS Office	50,008	26.5 / 73.5	0.62 / 4.55	53.2
Video File	13,381	12.1 / 87.9	0.12 / 2.52	86.2
100 MP3	8,267	19.6 / 80.4	0.10 / 1.55	96.8
Photoshop	56,834	23.2 / 76.8	1.49 / 4.77	48.3
Ubuntu10.04	24,010	30.1 / 69.9	0.35 / 6.16	64.0
Xen Compile	19,982	79.0 / 21.0	0.23 / 1.45	64.5

delay. This research data enable us to validate VSSIM against the commercially available SSD, Intel X25M (10 channel, 2 way, 2 plane, cell programming delay 900 μsec).

According to vendor specification[32], sequential write bandwidth of X25M is 70 MByte/sec with 900 μsec cell programming delay. When we set the cell programming delay of VSSIM version of X25M to 900 μsec, VSSIM version of X25M yields 68.9 MByte/sec sequential write performance. With physical measurement, our 1.5 year old X25M device yields 62 MByte/sec for sequential write and 907 IOPS for 4KB random write(direct IO, No write buffer).

We adjust the NAND programming delay of VSSIM from 900 μsec to 1100 μsec to match the physically observed performance of the target SSD. As NAND device ages, NAND flash gets exposed to various types of physical defects, e.g. charge detrapping, Stress Induced Leakage Current(SILC), Random Telegraph Noise(RTN), and etc. NAND device becomes vulnerable to disturbance(program, read, erase), interference, and parameter spread. As a result, it takes more time to program or read a NAND cell[33].

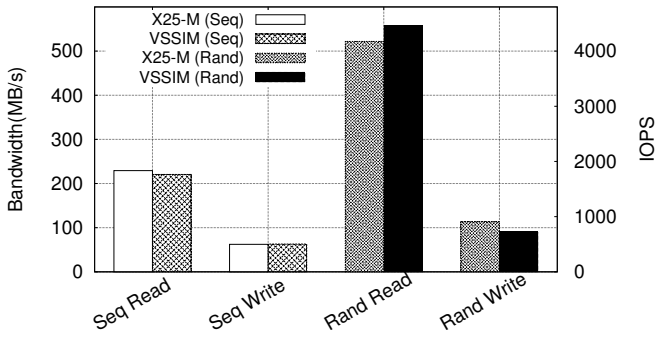
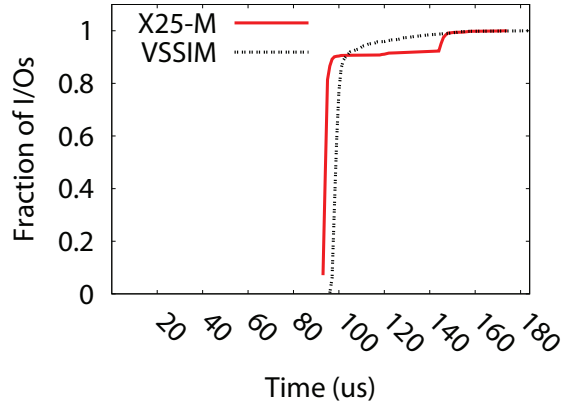


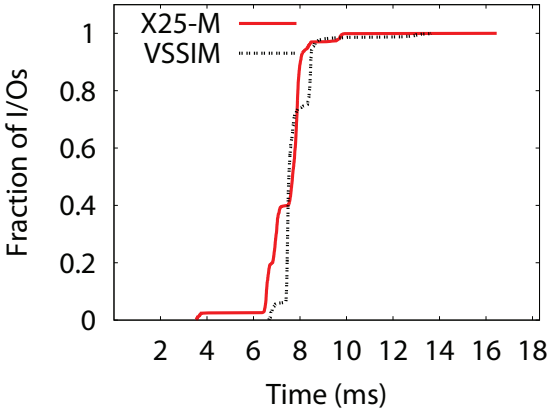
Fig. 6. Performance Comparison: X25M vs. X25M$_{VSSIM}$

We examine the performance of sequential IO (MByte/sec) and random IO (IOPS) of X25M and VSSIM model of X25M, X25M$_{VSSIM}$. The performance numbers below are the average of ten runs. X25M device is secure-erased before the start of each run.

X25M$_{VSSIM}$ yields 63 MByte/sec and 220 MByte/sec for sequential write and read, respectively. For sequential IO, VSSIM accurately models the performance of X25M within 3% offset. Figure 6 illustrates the result of the experiment. With random IO(IOZONE, 4 KByte record size, 512 MByte file), X25M yields 4.1 KIOPS and 0.9 KIOPS for read and write, respectively. X25M$_{VSSIM}$ yields 4.4 KIOPS and 0.7 KIOPS for random read and write, respectively (Figure 6). In simulating the random IO performance, VSSIM is less accurate than in simulating the sequential IO. We applied Demerit [34] to validate VSSIM. We examine the CDF of IO latency. Figure 7 illustrates the result. Write and Read difference in mean of X25M and X25M$_{VSSIM}$ is 5.4% and 9%, respectively(Table V).



(a) read



(b) write

Fig. 7. CDF of IO latency, IO size: 512KByte data using demerit

TABLE V. QUANTILE STATISTICS (WORKLOAD: RECORD SIZE 512KBYTE, FILE SIZE 512MBYTE, O_DIRECT)

SSD	IO	Min	25%	Median	Mean	75%	Max
X25M	Write	3.5	6.9	7.7	7.4	7.9	16.5
VSSIM	Write	6.7	7.5	7.5	7.8	8.3	13.6
X25M	Read	0	93	94	91.6	94	174
VSSIM	Read	0	97	98	99.9	99	184

The performance of an SSD is governed by the hardware configuration, e.g. channels, ways, planes and etc. and the software components, e.g. mapping, garbage collection, buffer

management, map caching, and etc. Hardware components determine sequential IO performance and VSSIM models the existing SSD with 3% performance offset. On the other hand, random IO performance varies widely subject to the internal firmware algorithms details of which are unknown to public and are difficult to reverse engineer. X25M$_{VSSIM}$ is not loaded with any acceleration technique to improve the write performance and its performance is slightly lower than real X25M. In this regard, VSSIM provides sufficiently accurate baseline platform to study the effect of various SSD design parameters.

C. Page Size and Channel Parallelism

Channel Parallelism becomes less beneficial as Flash page size becomes larger.

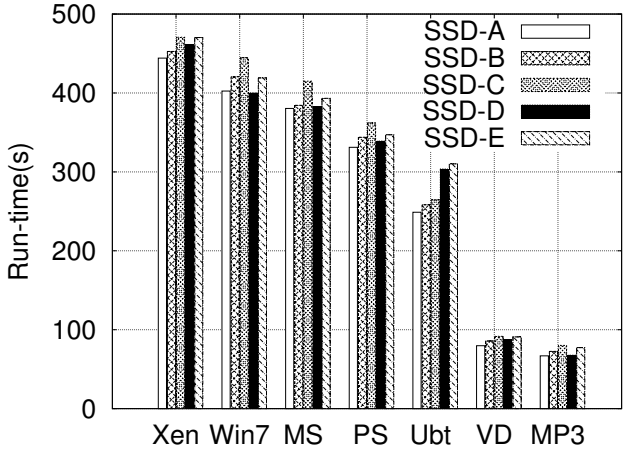


Fig. 8. Host Performance (page mapping) (Xen: Compile Xen, WIN7: Install Windows7, MS: Install MS Office, PS: Install Photoshop CS4, Ubt: Install Ubuntu 10.04, VD: Copy a video file, MP3: Copy 100 Mp3 file)

While increasing the number of channels is widely perceived as a right way to improve performance, it does not come for free. If we increase the number of channels, SSD controller needs more pins. Also, as SSD has more channels, the peak current increases, which makes the SSD controller circuit more susceptible to noise, ground bounce, blackouts and etc [35]. Increasing the number of chips per channel, i.e. increasing the *way* parallelism can be a possible resort to resolve pin count and circuit size issues in increasing the channel parallelism.

We tested the performance for five SSDs with page mapping. SSD A, B and C have eight, four and two channels, respectively. SSD D and E have four and two channels, respectively. Figure 8 illustrates the results. As we double the number of channels and reduces the ways, the time to install softwares decreases by 4-8%. Special care needs to be taken to determine if this performance gain deserves its complexity.

Another important aspect of modern SSD is the flash page size. When we use a larger page size chip, we can improve the sequential read performance. This is important in reducing the booting time in the mobile devices. However, a larger page size means more page invalidation. Figure 9 illustrates the IOPS for 4 KByte random write. For SSD-A, SSD-B and SSD-C, there exist approximately 10% improvement when we increase

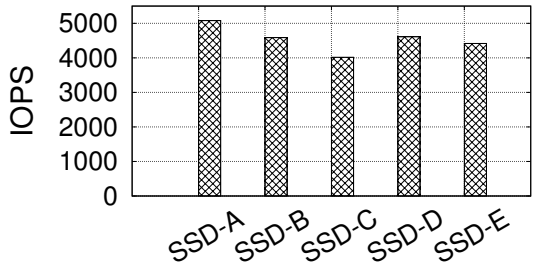


Fig. 9. 4Kbyte Random Write (IOZONE)

the number of channels. For SSD-D and SSD-E, on the other hand, the performance difference is less than 3%. The former SSDs and the latter SSDs use 2 KByte page and 4 KByte page NAND, respectively. As the cell program time constitutes more dominant fraction of page write latency, i.e. a page gets larger, the performance advantage of channel interleaving over way interleaving becomes less significant. The current state-of-the-art NAND flash uses 8 KByte page and the page size is expected to grow. As the experiment result shows, the advantage of channel interleaving against way interleaving will become less significant in the future flash memory and even further so if we take into account the hardware overheads (circuit complexity, pin count, peak power consumption, etc.) involved in realizing more channels.

D. IO size and Parallelism in FTL

FTL should support variable mapping unit size.

We examine the IO size distribution of the write operations for seven workloads (Window7 installation, MS office installation, Photoshop installation, video file copy, MP3 file copy, Install Ubuntu 10.04, Compile Xen) and how the channels are utilized for the respective workloads. In all workloads, 76% of write operations are either less than 4 KByte or larger than 64 KByte (Figure 10). For file download, most of the IO are 128KByte.

The IO size directly governs the effectiveness of the internal parallelism. We examine the channel usage pattern in 8 channel SSD (SSD-A) and 4 channel SSD (SSD-D), respectively. Figure 11 illustrates the results. In software installation workloads (Windows 7, MS Office, and Photoshop)

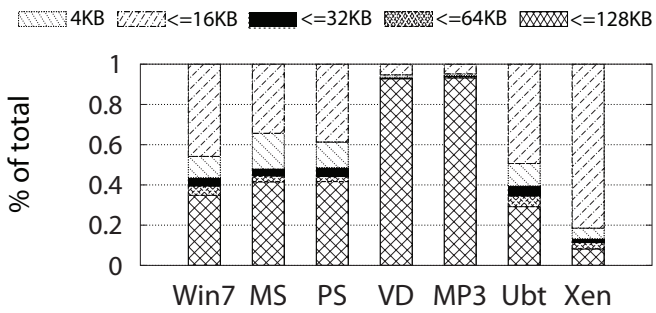
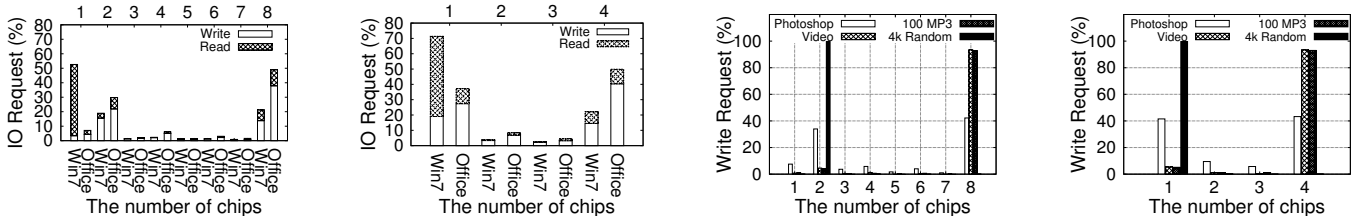


Fig. 10. IO size distribution (Write) (WIN7: Install Windows7, MS: Install MS Office, PS: Install Photoshop CS4, VD: Copy a video file, MP3: Copy 100 Mp3 file, Ubt: Install Ubuntu 10.04, Xen: Compile Xen)



(a) 8 ch: Read and Write (b) 4 ch: Read and Write (c) 8 ch: Write only (d) 4 ch: Write Only

Fig. 11. Channel Parallelism in Page Mapping

(Figure 11(a), Figure 11(b) and Figure 11(c)), single channel IO and full channel IO together accounts for more than 84% of the IO requests. In file copy operations (MP3 and video file), 97% of the writes exploit all channels (Figure 11(c) and Figure 11(d)). In random write, 99% of the writes use only one channel.

FTL exploits internal SSD parallelism either via managing channels independently (Intel X-25M [31]) or via managing them as a group (Samsung XMP [31]). The former and the latter are good for small random IO and for large IO of multiple pages, respectively. SSD should be able to handle small size IO's (one page or less) and large size IO's (64KB or more) both efficiently. However, most (if not all) FTL's are designed for either of the two. Numerous hybrid mapping algorithms attempt to address this issue via handling random write (small write) and sequential write (large write) differently. However, hybrid mapping cannot be used in multi-channel/multi-way environment, which is to be shown shortly. This suggests a new avenue for future FTL design. FTL should be able to dynamically adjust the mapping granularity ranging from subpage size to maximum IO size, e.g. 512 KByte in eMMC. Benefit of multi-granularity mapping approach will be more significant as SSD capacity increases and as a single SSD harbors multiple filesystem partitions each of which exhibits unique access characteristics. If multi-granularity mapping accompanies significant overhead in FTL design, establishing physical partition on SSD subject to the file system partition information, e.g. `/etc/fstab`, and managing them with different FTL (with different mapping granularity) can be a resort.

E. Hybrid mapping and Parallelism

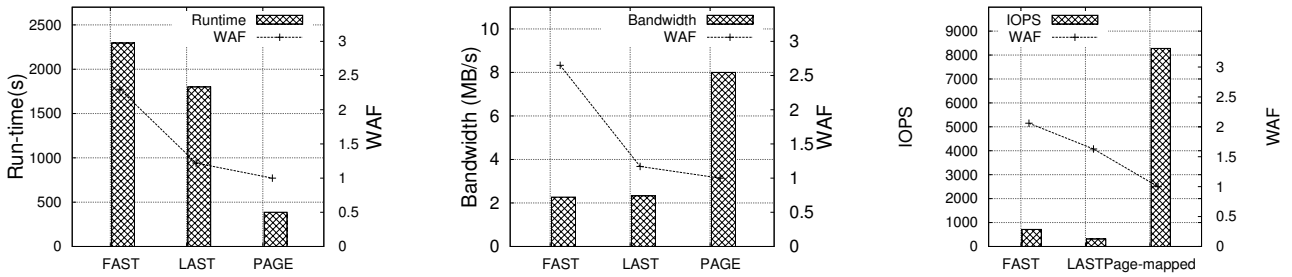
Hybrid mapping breaks in multi-channel/multi-way SSD due to log block fragmentation.

Reducing the size of mapping table is one of the most critical concerns in designing modern flash based storage. While page mapping yields the best random write performance, page mapping suffers from large size of its mapping table. Various hybrid mapping techniques have been proposed [36], [24], [37], [38] to achieve random write performance similar to page mapping while reducing the mapping table size by orders of magnitude. There is ongoing debate among the practitioners on whether to adopt hybrid mapping or not. In reality, however, few commercial SSDs adopt hybrid mapping. While the hybrid mapping brings clear benefit of reduced mapping table, the performance of the hybrid mapping is yet-to-be known under multi-channel/multi-way settings.

We examine the performance of page mapping and two hybrid mapping(FAST and LAST) FTLs. We run three workloads: Windows install, file copy and random write. The SSD is configured to 4 channel 4 way. In this study, we conclude that hybrid mapping is not practically feasible FTL in multi-channel and multi-way settings due to log block fragmentation and subsequent excessive log block merge overhead.

The objective of SSD parallelism(multi-channel/multi-way architecture) is to perform multiple NAND operations in concurrent manner. SSD controller achieves this objective via striping a logical block across the channels and the ways in a certain granularity, e.g. a page[32]. Multi-channel/multi-way SSD distributes the write requests across the channels and/or ways and services them in parallel fashion to expedite the IO performance. In this mechanism, a set of "correlated" write requests can be dispersed across the channels. The SSD performance is critically governed by the overhead of block consolidation, i.e. garbage collection in page mapping and log block merge in hybrid mapping. The key ingredient of block consolidation is to minimize the number of valid pages moved to a new block. Therefore, it is desirable to maintain correlated page in the same block since they are likely to be updated, i.e. invalidated, together. Variety of hybrid mapping algorithms put significant effort to maintain correlated pages in the same physical block. We call the phenomenon that a log block in the hybrid mapping is striped across the channels or across the ways as *log block fragmentation*. The fundamental issue with log block fragmentation is that it may dismantle the locality in the log block, which the hybrid mapping scheme aims at exploiting. We study the effect of SSD parallelism and the performance of hybrid mapping. This study consists of two phase. First, we examine the SSD performance under "as is" porting of hybrid mapping over multi-channel SSD. Second, we revise the hybrid mapping for multi-channel/multi-way environment and compare the performance against page mapping.

In "as is" approach, SSD controller blindly stripes all blocks (data block and log block) across the channels in page granularity. We measure the time to install Windows7 operating system. We found that the log block merge overhead becomes extremely high and Windows7 installation takes approximately 20 hours. Second, we revise the hybrid mappings(FAST and LAST) for multi-channel environment. We modify these mappings so that the sequential log block is not striped and is stored in the single physical block. This is to preserve the spatial locality among the pages via storing the pages at the same physical block. There is additional modification to LAST FTL. In multi-channel version of LAST, we do not stripe hot log block. This is to preserve the access



(a) Windows 7 Installation (b) Copying 100 MP3 files (c) Random Write (IOZONE)

Fig. 12. Performance and Write Amplification Factor (WAF)

correlations in the hot log block.

We examine the performance of revised hybrid mappings and page mapping. We use four channel one way SSD(SSD-D in Table III). We install Windows7 OS and measure the installation time. For sequential and random IO performance, we copy one hundred mp3 file and run random write operation of IOZONE. FAST has 16 random log blocks and four sequential log block. LAST has 8 hot and 8 cold random log block and 4 sequential log block. Over-provisioning rate is approximately 2%. Figure 12 illustrates the result. Among the three FTL's, page mapping yields the best performance for all three workloads. Windows 7 installation takes approximately 6.3min in page mapping while it takes about 4.2 and 5 times longer, in FAST and LAST, respectively. For file copy, bandwidth of page mapping is at least three times higher than the bandwidth of hybrid mappings. The performance difference becomes even starker in random write workload. The IOPS of page mapping is six times higher than that of hybrid mappings. SSD performance is very sensitive to its firmware algorithm. With the same hardware, IOPS and bandwidth can differ by as large as x6 subject to FTL (Figure 12).

In our experiment, hybrid mapping suffers from severe log block fragmentation if it interleaves the log block across the channels. When the controller does not stripe the log block across the channel, it fails to fully utilize the channel parallelism. We find that performance penalty is less severe if hybrid mapping does not interleave the log blocks whose pages are strongly correlated, but still the overall performance is 1/6 of page mapping.

The SSD adopts internal parallelism for better performance. To minimize the performance overhead of log block merge operation, the hybrid mapping should not use parallelism and should direct the correlated pages into the same physical block. Resolving the deficiency of hybrid mapping in multi-channel/multi-way SSD is not a trivial issue since the deficiency is originated from the fundamental nature of NAND flash memory: difference between the write unit, "page" and the erase unit, "block". We carefully argue that hybrid mapping as currently it stands, does not fit in multi-channel/multi-way SSD.

F. Write Amplification and Performance

Write Amplification Factor may not be a right performance indicator.

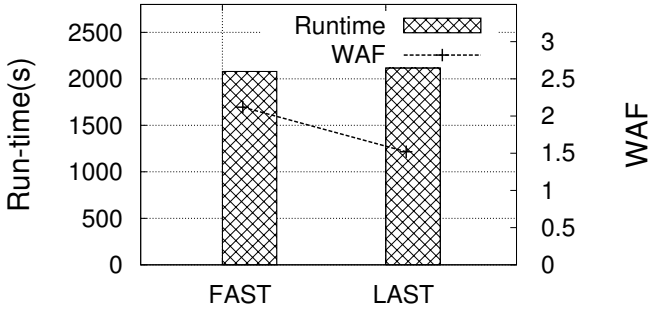


Fig. 13. Write Amplification and Performance (Windows 7 Installation)

Write Amplification Factor (WAF) is the ratio between the number of page writes from the host to the number of page writes which actually happen into flash memory. Write Amplification Factor is being widely perceived as fair metric for SSD performance and endurance. Recently, a number of works have been dedicated to estimate the SSD performance, especially random write performance, using the Write Amplification Factor(WAF)[39], [40], [41], [42]. Despite the efficacy of these analytical models, it is not clear how well the Write Amplification Factor represents the performance of an SSD in practice. With VSSIM which enables us to measure the host performance in real time, we find that the performance becomes better with larger Write Amplification Factor.

We build 4 channel 1 way SSD with VSSIM. We load two hybrid mapping(FAST and LAST) on this SSD and measure the time to install Windows 7, respectively. In this experiment, Write Amplification Factors for FAST and LAST yield 2.1 and 1.5, respectively. LAST exhibits significantly better Write Amplification Factor. Despite its larger, i.e. worse Write Amplification Factor, Windows7 installation takes shorter in FAST than in LAST. Write Amplification does not properly reflect the performance of the underlying storage. Figure 13 illustrates the result.

The performance of an SSD is governed by three factors: (i) the number of flash page writes, (ii) the number of erases and (iii) the degree of parallelism. The Write Amplification represents only the first, *the number of actual page writes*, out of three key constituents of storage performance.

We dissect the behavior of two FTLs, and examine why the WAF does not coincide with the host performance. FAST impose tighter constraints in categorizing the incoming write as

sequential. As a result, larger fraction of writes are categorized as random in FAST than in LAST (97% vs. 3%). Subsequently, FAST loaded SSD has more opportunity to distribute the write operations across the channels exploiting the hardware parallelism. Unexpected consequence is that FAST fails to preserve correlated block accesses at a single physical block. We measure the log block associativity. We find that average log block associativity for FAST and LAST corresponds to 62 and 18, respectively. As a result, FAST yields larger WAF than LAST in Windows7 installation.

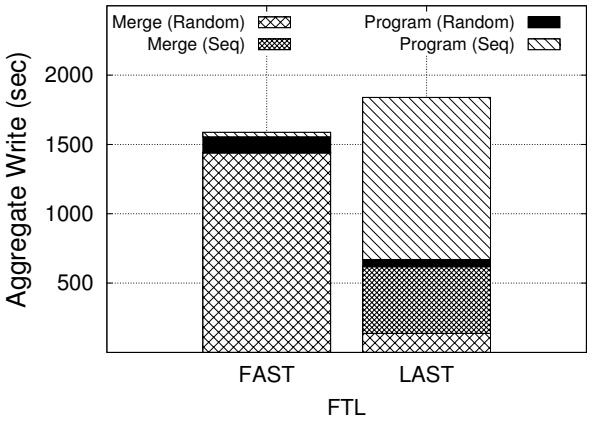


Fig. 14. Dissection of Aggregate Write Latency (Window 7 Installation)

Let us further analyze the details. Increase in the parallelism improves the performance and the increase in the log block associativity degrades the performance due to the increase in WAF and the number of erase operations. We examine how these two factors affect the overall performance. We measure the time spent on writing pages(Figure 14). There are two types of page writes in hybrid FTL: data write and log block merge. The former is caused by the request from the host and the latter is caused by consolidating the log blocks with the data blocks. LAST spends approximately 300 sec more in writing data than FAST does. We observed that FAST generates more writes due to larger log block associativity, i.e. larger Write Amplification. FAST spends most of its time on merging random log blocks and the time to merge sequential log block can be ignored since the sequential writes constitutes only 3% of entire write. In LAST on the other hand, 97% of the writes are sequential. LAST spends most of its time on handling sequential log blocks (write and merge). FAST does suffer from higher block associativity of random log blocks and LAST cannot parallelize the writes to sequential log block. In this experiment, advantage of striping the page writes across the channels outweighs the advantage of maintaining them in a single block to preserve access correlation. As a result, FAST yields the better performance than LAST does even with larger Write Amplification.

G. Importance of Hot/Cold Identification

Hot block identification is critical in making the over-provisioning effective.

SSD reserves a certain fraction of blocks in each NAND device to record updated pages. These blocks are not visible to the host. This is called over-provisioning and the ratio between the amount of invisible blocks in the entire storage is called over-provisioning factor. With larger over-provisioning degree, FTL is given more opportunity to cluster the pages in the same category together, e.g. random or sequential or hot or cold pages together, which makes the block consolidation much more efficient, i.e. low number of valid page copy operation incur. With VSSIM, we examine the time to install Windows7 OS under different over-provisioning degrees and with different ways of categorizing page writes.

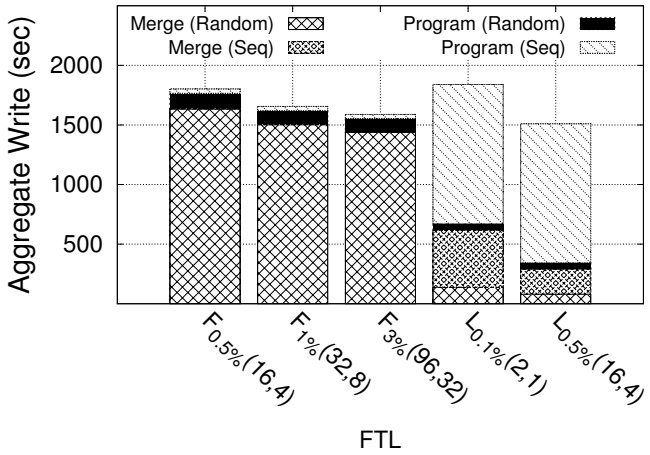


Fig. 15. Aggregate Write time: Windows7 Installation (No. of Random log blocks, No. of Sequential log blocks)

We examine two hybrid mappings, FAST and LAST. We study the relationship between the SSD performance and the over-provisioning degree. For FAST, we vary the number of random and sequential log blocks as (16,4), (32,8) and (96,24), which corresponds to over-provisioning ratio of 0.5%, 1% and 3%. For LAST, we vary the number of random and sequential log blocks as (2,1), and (16,4). Over-provisioning ratios correspond to 0.1% and 0.5%, respectively. Figure 15 illustrates the results.

FAST spends most of the time in merging random log blocks and LAST spends most of the time in writing sequential log blocks. It is surprising to see that LAST yields the same performance as FAST with nearly 1/6 of the log blocks (Figure 15). This clearly suggests that properly incorporating the temporal locality in categorizing the incoming write requests is extremely important to make the over-provisioning effective. In FAST and LAST, average latency of writing a page (random) corresponds to 0.1 msec and 0.7 msec, respectively. FAST well exploits channel parallelism and reduces the effective write latency of random log block. However, the merge overheads of random log blocks in FAST and LAST correspond to around 200 msec and 30 msec, respectively. Due to higher log block associativity, FAST yields excessive log block merge overhead for random log blocks.

H. Related Works

Estimating the behavior of the NAND based storage device is of paramount interests recently. The modeling efforts can be categorized into analytical modeling[6], [39], [40], [41], [42], trace driven simulator[3], [5], [7], [9], virtual device based

simulator[4], [6], [8], and hardware based emulator[10], [11], [12], [13].

DiskSim [43] is one of the foremost and well-known trace driven simulators for hard disk drive. Agarwal et al. [3] developed the first fairly detailed SSD model based upon the DiskSim. Since it has a monolithic architecture, it is difficult to extend and to adopt a new FTL. FlashSim [5] is another trace driven simulator that extends the DiskSim. FlashSim takes an object-oriented approach. Flashsim is pre-loaded with three FTLs and easy to add one. There also introduced clock level simulator to study the bus utilization, channel utilization of NAND based storage[9], [7]. With these trace driven simulators, we can examine the fairly detailed behavior of the device internals, but cannot directly capture the host performance. The fundamental limitation of trace driven simulator is its limited capability to understand the interaction between the host and the device, which above mentioned trace driven SSD simulators are not free from.

Another approach to study the SSD performance is to use virtual device[4], [6], [8]. NANDSim[4] exports very primitive FTL with Read/Write/Erase operations. It is far from modeling various components of the SSD, e.g. FTL, garbage collection, wear-leveling, degree of parallelism, and etc. Kaoutar et. al. [6] proposed a virtual device similar to NANDSim. They developed an analytical model for delay. Even the state of art virtual device provides very simple storage model(single chip, one channel/one way without buffer). The virtual devices are not only incapable of incorporating details of the device hardware internals but also unable to load advanced firmware algorithms.

Hardware based emulator generates the most realistic results among different modeling approaches, but is the least flexible avenue. We cannot adjust hardware design parameters, e.g. the parallelism degree, page size, the number of pages per block, channel clock frequency, and etc. Due to its cost and inflexibility, hardware based emulators[10], [11], [12], [13] are usually built at the final stage of the SSD development process. [13] does not have host interface and it cannot be attached to the host. Hardware based emulator design is dependent upon the NAND device type it is using. [10], [12], [13] are built for legacy NAND, which has been phased out. The emulator boards needs to be re-manufactured when new NAND device emerges[44].

V. Conclusion

In this work, we propose a Virtual SSD Simulator, VSSIM. We managed to incorporate not only essential components of SSD, but also the sophisticated algorithms to exploit multi-channel and multi-way characteristics of an SSD. With VSSIM, it is possible to examine the performance of the host and the detailed internal behavior of the SSD simultaneously in real-time manner. This eventually leads us to acquire the correlational characteristics between the various SSD design parameters and the host performance without physically prototyping it. The case studies in this work are primarily for exhibiting the capability of VSSIM, yet still deliver meaningful answers to critical design issues. First, the benefit of channel parallelism against way parallelism becomes less significant as flash page size increases. The performance advantage of channel parallelism against way parallelism is its ability to parallelize register write operations. However, as flash page becomes larger, NAND program delay constitutes dominant fraction of page write time and the time to transfer data between the controller and page register becomes less significant. Second, for four or more channels SSD, the log block operations of existing hybrid FTL's suffer from severe performance degradation due to log block fragmentation. Hybrid mapping technically breaks when parallelism degree is four or larger in SSD. Third, increasing the number of channels beyond four does bring performance improvement in software installation, massive file copy and random write, but, the performance gain is marginal and may not justify its design overhead, extra pin counts and the increase in peak power consumption. Fourth, Write Amplification may be misleading as a performance metric since it does not capture effect of the degree of parallelism and the number of erase operations on the SSD performance. Finally, it is critical that various types of locality need to be properly exploited to reduce the amount of log block required.

Acknowledgment

This work is sponsored by IT R&D program MKE/KEIT. [No.10035202, Large Scale hyper-MLC SSD Technology Development]. The authors would like to thank Haesung Kim and Joohyun Kim for their contribution to the development of VSSIM at its inception stage.

References

[1] "Flash memory to overtake dram market," in *IC Insights*, 2011.

[2] Gartner, "Forecast: Semiconductor consumption by electronic equipment type, worldwide, 4Q11 update," December 2011.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design trade-offs for SSD performance," in *Proc. of USENIX ATC '08*, 2008, pp. 57–70.

[4] "Memory technology device, www.linux-mtd.infradead.org."

[5] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "Flashsim: A simulator for nand flash-based solid-state drives," in *Proc. of the First International Conference on Advances in System Simulation*, Washington, DC, USA, 2009, pp. 125–131.

[6] K. El Maghraoui, G. Kandiraju, J. Jann, and P. Pattnaik, "Modeling and simulating flash based solid-state disks for operating systems," in *Proc. of WOSP/SIPEW '10*, San Jose, California, USA, January 2010, pp. 15–26.

[7] M. Jung, E. Wilson, D. Donofrio, J. Shalf, and M. Kandemir, "Nandflashsim: Intrinsic latency variation aware nand flash memory system modeling and simulation at microarchitecture level," in *Proc. of IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2012*, pp. 1–12.

[8] P. Jin, X. Su, Z. Li, and L. Yue, "A flexible simulation environment for flash-aware algorithms," in *CIKM '09: Proc. of the 18th ACM conference on Information and knowledge management*, New York, NY, USA, 2009, pp. 2093–2094.

[9] J. Lee, E. Byun, H. Park, J. Choi, D. Lee, and S. H. Noh, "Cps-sim: configurable and accurate clock precision solid state drive simulator," in *Proc. of ACM SAC '09*, Honolulu, Hawaii, USA, 2009, pp. 318–325.

[10] H. eok Kim, E. H. Nam, and K. S. Choi, "Development platforms for flash memory solid state disks," in *Proc. of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008*, pp. 527–528.

[11] "The openssd project," http://www.openssd-project.org.

[12] S. Lee, K. Fleming, J. Park, K. Ha, A. Caulfield, S. Swanson, J. Kim *et al.*, "Bluessd: an open platform for cross-layer experiments for nand flash-based ssds," in *WARP-5th Annual Workshop on Architectural Research Prototyping*, 2010.

[13] Y. Cai, E. Haratsch, M. McCartney, and K. Mai, "Fpga-based solid-state drive prototyping platform," in *Proc. of IEEE International Symposium on Field-Programmable Custom Computing Machines(FCCM)*, Salt Lake City, Utah, US, May 2011.

[14] M. Mesnier, M. Wachs, R. Sambasivan, J. Lopez, J. Hendricks, G. Ganger, and D. O'Hallaron, "`//Trace`: parallel trace replay with approximate causal events," in *Proc. of the 5th USENIX FAST*, 2007, pp. 24–24.

[15] N. Zhu, J. Chen, T. Chiueh, and D. Ellard, "Tbbt: scalable and accurate trace replay for file server evaluation," in *ACM SIGMETRICS Performance Evaluation Review, 2005*, vol. 33, no. 1, pp. 392–393.

[16] N. Joukov, T. Wong, and E. Zadok, "Accurate and efficient replaying of file system traces," in *Proc. of the 4th USENIX FAST, 2005*, vol. 5, pp. 25–25.

[17] E. Anderson, M. Kallahalla, M. Uysal, and R. Swaminathan, "Buttress: A toolkit for flexible and high fidelity i/o benchmarking," in *Proc. of the 3rd USENIX FAST, 2004*, pp. 45–58.

[18] V. Tarasov, S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok, "Extracting flexible, replayable models from large block traces," in *Proc. of the 10th USENIX FAST, 2012*.

[19] SAMSUNG, "Samsung electronics: 2g x 8 bit / 4g x 8 bit nand flash memory (k9xxg08uxm)," June 2006, specification.

[20] I. Corporation, "Understanding the flash translation layer(FTL) specification," Tech. Rep. 297816-002, December 1998.

[21] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proc. of ASPLOS '09*, Washington, DC, USA, 2009.

[22] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *Consumer Electronics, IEEE Transactions on*, vol. 48, no. 2, pp. 366 –375, May 2002.

[23] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, July 2007.

[24] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: locality-aware sector translation for nand flash memory-based storage systems," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 36–42, October 2008.

[25] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable ftl architecture for nand flash-based applications," *ACM Trans. on Embedded Computing Systems*, vol. 7, no. 4, July 2008.

[26] H. Kwon, E. Kim, J. Choi, D. Lee, and S. Noh, "Janus-ftl: finding the optimal point on the spectrum between page and block mapping schemes," in *Proc. of the tenth ACM international conference on Embedded software*, 2010, pp. 169–178.

[27] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. of USENIX ATC '05*, Anaheim, CA, USA, April 2005, pp. 41–46.

[28] A. Kivity, "KVM: the linux virtual machine monitor," in *Proc. of the Ottawa Linux Symposium(OLS), 2007*.

[29] M. Wu and W. Zwaenepoel, "envy: a non-volatile, main memory storage system," in *ACM SigPlan Notices*, vol. 29, no. 11. ACM, 1994, pp. 86–97.

[30] Intel, "Datasheet: Md332b NAND flash memory (js29fxxg08xxxdb)," August 2009, specifiacation.

[31] B. Yoo, Y. Won, S. Cho, S. Kang, J. Choi, and S. Yoon, "SSD characterization: From energy consumption's perspective," in *Proc. of HotStorage 11*, Portland, OR, USA, June 2011.

[32] I. Corporation, "Intel x25-m sata solid-state drive," specification. [Online]. Available: http://download.intel.com/design/flash/nand /mainstream/mainstream-sata-ssd-datasheet.pdf

[33] E. W. Dijkstra, *Stepwise program construction*. Springer, 1982.

[34] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *Computer*, vol. 27, no. 3, pp. 17–28, 1994.

[35] G. Hong, "Analysis of peak current consumption for large-scale, parallel flash memory," Presented at Workshop for Operating System Support for Non-Volatile RAM(NVRAMOS 2011 Spring), Jeju, Korea, April 2011.

[36] D. Ma, J. Feng, and G. Li, "Lazyftl: a page-level flash translation layer optimized for nand flash memory," in *Proc. of the 2011 international conference on Management of data*. ACM, 2011, pp. 1–12.

[37] H. Lee, H. Yun, and D. Lee, "Hftl: hybrid flash translation layer based on hot data identification for flash memory," *Consumer Electronics, IEEE Transactions on*, vol. 55, no. 4, pp. 2005–2011, 2009.

[38] S. Lee, W. Choi, and D. Park, "Fast: An efficient flash translation layer for flash memory," *Emerging Directions in Embedded and Ubiquitous Computing*, pp. 879–887, 2006.

[39] P. Desnoyers, "Analytic modeling of ssd write performance," in *Proc. of SYSTOR, 2012*, June.

[40] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proc. of SYSTOR, 2009*.

[41] R. Agarwal and M. Marrow, "A closed-form expression for write amplification in nand flash," in *Proc. of GLOBECOM Workshops*, 2010, pp. 1846 –1850.

[42] X. Haas, "The fundamental limit of flash random write performance: Understanding, analysis and performance modelling," IBM Research Report, 2010/3/31, Tech. Rep., 2010.

[43] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," Carnegie Mellon University, Tech. Rep., 2008.

[44] "Onfi nand flash specification," http://www.onfi.org/specifications.