Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity

Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity

Yang Hu[†‡], Hong Jiang[§], Dan Feng[†‡✉],
Lei Tian[†‡§],Hao Luo[†], Shuping Zhang[⁑]
[†]School of Computer, Huazhong University of Science and Technology, Wuhan, China, 430074
[‡]Wuhan National Laboratory for Optoelectronics, Wuhan, China, 430074
[§]University of Nebraska-Lincoln, Lincoln, United States, 68588
[⁑]Beijing Institute of Computer Technology and Application, Beijing, China, 100039
✉Corresponding author: dfeng@hust.edu.cn
{yanghu, hluo}@foxmail.com, jiang@cse.unl.edu, ltian@hust.edu.cn, zsp7098@sina.com

ABSTRACT

With the development of the NAND-Flash technology, NAND-Flash based Solid-State Disk (SSD) has been attracting a great deal of attention from both industry and academia. While a range of SSD research topics, from interface techniques to buffer management and Flash Translation Layer (FTL), from performance to endurance and energy efficiency, have been extensively studied in the literature, the SSD being studied was by and large treated as a grey or black box in that many of the internal features such as advanced commands, physical-page allocation schemes and data granularity are hidden or assumed away. We argue that, based on our experimental study, it is these internal features and their interplay that will help provide the missing but significant insights to designing high-performance and high-endurance SSDs.

In this paper, we use our highly accurate and multi-tiered SSD simulator, called SSDsim, to analyze several key internal SSD factors to characterize their performance impacts, interplay and parallelisms for the purpose of performance and endurance enhancement of SSDs. From the results of our experiments, we found that: (1) larger pages tend to have significantly negative impact on SSD performance under many workloads; (2) different physical-page allocation schemes have different deployment environments, where an optimal allocation scheme can be found for each workload; (3) although advanced commands provided by flash manufacturers can improve performance in some cases, they may jeopardize the SSD performance and endurance when used inappropriately; (4) since the parallelisms of SSD can be classified into four levels, namely, channel-level, chip-level, die-level and plane-level, the priority order of SSD parallelism, resulting from the strong interplay among physical-page allocation schemes and advanced commands, can have a very significant impact on SSD performance and endurance.

ICS'11, May 31–June 4, 2011, Tucson, Arizona, USA.
Copyright 2011 ACM 978-1-4503-0102-2/11/05...$10.00.

Categories and Subject Descriptors

B.1.4 [**Hardware**]: Microprogram Design Aids– *Firmware engineering.* B.3.3 [**Memory Structure**]: Performance Analysis and Design Aids– *Simulation.* D.4.2 [**Operating Systems**]: Storage Management– *Secondary storage.*

General Terms

Measurement and Performance

Keywords

NAND-Flash, SSD, simulator, advanced commands, parallelism

1. INTRODUCTION

NAND-Flash based Solid State Drive (SSD) has experienced tremendous development and growth during the last two decades. The enterprise-quality Flash memory storage has dropped in price, increased in per-unit capacity, improved in reliability, and addressed the random write performance penalty, which is traditionally associated with the technology, by various ingenious methods [1-6]. These advantages enable SSDs to be widely used in almost every aspect of modern computing systems, from low-end PCs to high-end servers in supercomputing, thus making the performance and endurance issues of solid-state storage system increasingly attractive to both academia and industry [7-11].

Several topics related to the performance and endurance of SSD, including FTL designs [12-15] and buffer schemes [16-19], have been extensively discussed in the literatures. Other studies in the literature deduce or infer the characteristics of flash or SSD by extended measurement [20-23]. However, some SSD internal behaviors with potentially important impacts on the system performance and endurance have been largely ignored. While very little has been studied and reported in the literature about the performance and endurance impacts and interplay of data granularity of SSD, allocation strategy and advanced commands, our experiences with SSD design and evaluation indicate that judicious use of these features can have significant performance and endurance impacts. For example, we found that diverse allocation schemes can result in different performance level and the way in which parallelism in SSD is exploited can be a key performance-impact factor. Moreover, when these internal SSD behaviors did get discussed in the literature they were studied in isolation without considering their interplay and interaction [14][17][24-27]. In fact, we found that it is this interplay and interaction among them that tend to have the most profound impact on system performance and endurance of SSD.

To thoroughly investigate the aforementioned performance impact and interplay of SSD advanced commands, allocation strategy, and data granularity, in this paper, we carefully examine several internal behaviors rarely discussed in the literature, which may have potentially important impact on the performance and endurance of SSD. Specifically, we carry out in-depth evaluations of these features and their interplay, obtaining the following main insights.

1. Flash page size

With the capacity growth of NAND flash chips, the size of an SSD page has increased significantly over the past few years, for example, from 512B in the 1990s to 16KB in 2010 [28-34]. However, when a page is updated partially (e.g., with small and random I/O requests in such workloads as MSN [35]), bigger pages are more prone to the problem in which some data must be read from the old page and written to a new page after being merged with the new data. This problem leads to degraded performance. In fact, enlarging the flash page size leads to that the average response time is increased to 1.8 times, compared to that of the best performing flash page-size, under the MSN workload. Considering the different access patterns of various workloads, there is no one-size-fits-all page size and an optimal page size must be and can be dynamically determined to optimize performance, adapting to different workloads.

2. Allocation schemes

Static allocation is based on fixed striping while dynamic allocation assigns pages dynamically. Although dynamic allocation is more flexible and adaptive in exploiting parallelisms, thus resulting in better performance in most cases, static allocation is simple in implementation and can be very effective in some workloads. Static allocation is found to perform the best in read operations under all workloads. Dynamic allocation performs the best overall performance and endurance under the most of workloads in aged SSDs.

3. Advanced commands

The SSD manufacturers have provided advanced commands, such as copy-back, multi-plane and interleave operations, with an intention to improve the performance of SSD by handling read, write and erase operations more efficiently. However, we find that some strict restrictions must be adhered to when using these commands, making their appropriate use extremely important. For example, using copy-back blindly leads to that the average response time and erasure count are increase to 7.7 and 17.8 times respectively, compared to that of only using basic commands, under the MSN workload.

4. Priority order of SSD parallelism

Parallelism has been regarded as a key to achieving the peak performance. There are four levels of parallelism in SSD: (1) among channels (channel-level), (2) among chips in a channel (chip-level), (3) among dies in a chip (die-level) and (4) among planes in a die (plane-level). Allocation schemes can effectively utilize the first two levels of parallelisms, while the last two levels of parallelisms can be exploited by advanced commands. We found that the four levels of parallelisms tend to have an optimal priority order. An incorrectly placed order of priority can result in a performance degradation of up to 60%.

The rest of the paper is organized as follows. Section 2 introduces the necessary background and related work to motivate our research. Section 3 presents and validates the evaluation platform – SSDsim, a highly accurate and multi-tiered simulator for the evaluation of various SSD internal behaviors. Then we present our extensive trace-driven evaluations of the internal behaviors of SSD on SSDsim in Section 4. Section 5 summarizes the key observations and insights obtained from our evaluations.

2. BACKGROUND AND MOTIVATION

2.1 Flash Memory Basics

In general, Flash memory can be classified into two categories: NOR and NAND [3]. NOR-Flash memory supports byte-level random accesses and is typically used in read-only applications such as storing firmware codes. NAND-Flash memory, in contrast, has higher density, larger capacity and lower cost than NOR-Flash, but only supports block-level random accesses. It is thus typically used for more general-purpose applications.

There are two NAND-Flash technologies, Single-Level Cell (SLC) and Multi-Level Cell (MLC) [4][5]. While the former stores only one bit per cell, the latter stores two or even more bits per cell. Throughout this paper, we will use the term Flash to refer specifically to NAND-Flash memory.

To increase storage density, flash manufactories package several flash chips together, a model called package [28-34]. All chips in a package share the same 8/16-bit I/O bus of the package but have separate chip enable and ready/busy control signals. Each chip is composed of two or more dies. Each die has one internal ready/busy signal that is different from the external ready/busy signal of a chip. The internal ready/busy signal is invisible to user. It will only be used in *advanced commands*. Each die is composed of multiple planes. Each plane contains thousands of flash blocks and one or two data/cache registers used as an I/O buffer. A flash block typically consists of 64 or 128 pages, where a page is further divided into many 512B sub-pages. Each sub-page has a 16B spare space used to store a variety of information, such as error correction code (ECC), logical page number and sub-page state. The size of a page has been steadily increasing due to the technology development as well as the growing size of a single chip. While chip and die are not clearly distinguished and often confused with each other in many previous studies in the literature, chip enable and read/busy signals make them clearly distinct from each other. A chip is the basic functional unit that has its independent chip enable and read/busy signals. A die is a component of a chip, which only has an internal read/busy signal.

There are two key and unique flash characteristics, namely, *write-after-erase* and *erase cycle*. A write operation can only change the value of each target bit from `1` to `0`. Once a page is written, it must be erased, where all bits are reset to `1`, before the next write operation can be performed on the same page. Each flash block has a limited number of erase cycles before it is worn out. After wearing out, a block can no longer store any data. A typical MLC Flash has an erase-cycle limit of about 10K, while a typical SLC Flash has an erase-cycle limit of about 100K with a 1bit/512byte ECC [4].

The page size of early NAND-Flash products is typically 512 bytes, consistent with a hard disk drive (HDD) sector [36]. With the development of fabrication technology, the storage density has been steadily increasing, resulting in a diverse set of page sizes among NAND-Flash products, including 2KB, 4KB, 8KB and 16KB [28-34]. In the pursuit of higher capacity, SSD products are employing increasingly large page size.

2.2 Flash Commands: Basic Commands and Advanced Commands

There are three basic operations in Flash: read, program (write) and erase. A read operation fetches data from a target page. A write operation writes data to a target page. An erase operation resets all bits of a target block to `1`. All operations are initiated by writing the command code to the command register and the address of the request to the address register. The address points to the target data of the request inside the package. An address is separated into six segments: chip address, die address, plane address, block address, page address and in-page address, as illustrated in Figure 1. Within a block, the pages must be programmed consecutively in the increasing order of page address. Random-page-address programming is prohibited. We call this Restriction (a).

↓Most significant bit — Least significant bit↓

chip	die	plane	block	page	In-page

|← In flash package address →|

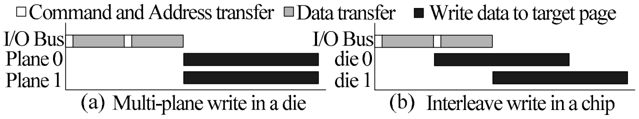
Figure 1. Flash package address format.

Most flash manufacturers provide advanced commands, such as *copy-back*, *multi-plane* and *interleave*, to further improve the performance of SSD. Advanced commands are extensions of the basic read, program and erase commands but with some usage restrictions [28-34][37].

Copy-back (internal data move) command moves one page of data from one page to another in the same plane, without occupying the I/O bus. Some manufacturers also call this command internal data move [38-39]. We will call it copy-back in the remainder of the paper. The source page and the target page must have the same chip, die and plane addresses. The addresses of the source page and destination page must be both odd or both even. As shown in Figure 4, a copy-back operation can only move data from page 0 to page 2, or from page 1 to page 3, etc. Moving data from page 0 to page 1 or page 3 is prohibited. We call this Restriction (b).

Multi-plane command activates multiple read, program or erase operations in all planes of the same die. It only costs the time of one read, write or erase operation, while executing multiple such operations, as illustrated in Figure 2 (a). The pages executing a multi-plane read/write operation must have the same chip, die, block and page addresses. And the blocks executing a multi-plane erase operation must have the same chip, die and block addresses. As shown in Figure 4, only page 1 from plane 0 and page 1 from plane 1 of the same die can be read/written simultaneously by using a multi-plane read/write operation. Reading/writing page 1 from plane 0 and page 3 from plane 1 using a multi-plane read/write operation is prohibited. We call this Restriction (c).

Interleave command executes several page read, page write, block erase and multi-plane read/write/erase operations in different dies of the same chip simultaneously. The interleave command is different from the interleave operation mentioned in [17][25-27]. The former is a flash command that is executed among different dies in the same chip, while the latter is executed among different chips in the same channel. An interleave write command is illustrated in Figure 2 (b). Other than the restriction that pages operated simultaneously must belong to different dies on the same chip, there are no other restrictions when using the interleave command.



(a) Multi-plane write in a die (b) Interleave write in a chip

Figure 2. The multi-plane write and interleave write command process

2.3 Allocation Schemes

An allocation scheme determines how to choose free physical page(s) to accommodate logical page(s) being written to the SSD. To locate a particular physical page, one must know the channel address and package address, in addition to the chip address, die address, plane address, block address and page address, as shown in Figure 1. The format of a full address of SSD is shown in Figure 3.

Allocation schemes are classified into two categories: dynamic and static.

Static allocation first assigns a logical page to a *pre-determined* channel, package, chip, die and plane, before allocating it to any free physical page of the plane. The channel, package, chip, die and plane addresses assigned to each logical page are typically calculated by some formulas that define a special allocation scheme.

Dynamic allocation assigns a logical page to any free physical page of the entire SSD. When a write request arrives, a dynamic allocation scheme chooses a free physical page by considering several factors, such as the idle/busy state of channels, the idle/busy state of chips, the erasure count of blocks, the priority order of parallelism and so on. The scheme that assigns a logical page to any free physical page of the pre-determined channel is also classified in the dynamic allocation category.

There are many existing static allocation schemes, of which the scheme shown in Figure 11(b) has been shown to perform the best by an extensive comparative study in [14].

There has not been any direct comparison between the static and dynamic allocation schemes in the literature, to the best of our knowledge. In this paper, we will evaluate and directly compare the static and dynamic allocation schemes in terms of performance and wear-leveling.

channel	package	chip	die	plane	block	page	In-page

|← In flash package address →|
|← Full SSD address →|

Figure 3. The format of a full SSD address.

2.4 Parallelism inside SSDs

There are four levels of parallelism in SSD: (1) among channels (*channel-level*), (2) among chips in a channel (*chip-level*), (3) among dies in a chip (*die-level*), and (4) among planes in a die (*plane-level*). For example, in Figure 4, if a request is served by channel 0 and channel 1 simultaneously, it exploits the channel-level parallelism; if it is served by chip 0 of package 0, chip 1 of package 0 and chip 0 of package 1 in channel 0 simultaneously, it leverages the chip-level parallelism; if it is served by plane 0 of die 0 and plane 1 of die 1 on the same chip, it utilizes the die-level parallelism; if it is served by plane 0 and plane 1 of the same die, it makes use of the plane-level parallelism.

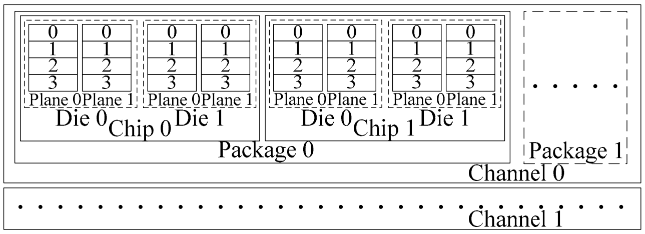


Figure 4. An SSD internals.

[25] and [27] exploit the channel-level parallelism; [17] makes use of the chip-level parallelism; and [26] employs the plane-level parallelism. Previous studies in the literature mainly focus on the first two levels of parallelism. In this paper, we will evaluate the performance impact of exploiting all four levels of SSD parallelisms through the interplay of the aforementioned internal behaviors of SSD.

2.5 SSD Simulator

At present, there are only two open-source SSD simulators [40-41] available in the public domain. They provide basic research platforms for researchers to evaluate their designs of FTL. They provide the first two levels of parallelism, including channel-level and chip-level. They only support one of the three advanced commands, copy-back. However, they both fail to adhere to Restriction (b) (Section 2.2) when using the copy-back command. Furthermore, they did not validate their measurement accuracy against a real SSD system by directly comparing the simulation measurements with the real SSD system measurements.

To address the drawbacks of the existing SSD simulators, we designed and implemented a new SSD simulator, called SSDsim, which provides the detailed and accurate simulation of each level of SSD, including hardware, FTL and buffer layer. It provides four levels of parallelism, supports all the advanced commands that adhere to all the aforementioned restrictions. It is directly validated against a real SSD prototype. The aim of design and implementation of SSDsim is to provide an open-source and high-accuracy SSD research tool for all researchers.

2.6 Research Motivation

To design a high-performance and high-endurance SSD, we must comprehensively consider many factors that have been discussed so far. More specifically, we must answer the following research questions that have not been fully addressed, if at all, in the literature, to the best of our knowledge.

Question 1: Is the flash page size a factor impacting the SSD performance? And if so, to what extent? Enlarging the flash page size can increase the capacity of SSD. But does it also help performance?

Question 2: How to choose allocation schemes?

The question has been partially answered by previous studies in the literature. However, a comprehensive answer is still elusive. Since different workloads have diverse characteristics, no one allocation scheme can possibly fit all workloads. On the other hand, are there certain workloads that will be best suitable for a particular type of scheme? For example, we found that in all cases, the read performance of the static allocation scheme is consistently superior to that of the dynamic allocation scheme. Conversely, in an aged SSD, the dynamic allocation scheme significantly outperforms the static allocation scheme.

Question 3: Do advanced commands always improve performance? If not, how should they be appropriately used to promote performance?

Question 4: Given the four levels of parallelism in SSD, what is their priority order that optimizes the performance and endurance of SSD?

To comprehensively answer these questions, we conduct a series of trace-driven experiments and evaluations detailed in Section 4, on the SSDsim simulator to be described next.

3. EVALUATION PLATFORM

In this section we present the evaluation platform on which the in-depth investigation into the internal SSD behaviors and their interplay (to be detailed in Section 4) are conducted by first introducing and validating the core of this platform, the SSDsim simulator. This is followed by a description of the real-world workloads chosen for this investigation and the configuration of the evaluation platform.

3.1 SSDsim Simulator and Its Validation against a Hardware SSD Prototype

We design and implement an SSD simulator, called SSDsim, which is event-driven, modularly structured, and multi-tiered. SSDsim is a single-threaded program written in C, which has about 15 thousand lines of C code. SSDsim is capable of simulating most SSD hardware platforms, mainstream FTL schemes, allocation schemes, buffer management algorithms and request scheduling algorithms. The three-tiered SSDsim design consists of the buffer and request-scheduling module at the top, the FTL and allocation module in the middle, and the low-level hardware platform module at the bottom. The top module is responsible for buffer organization and scheduling requests; In the middle module, the FTL sub-module simulates many state-of-the-art FTL schemes including pure page-FTL [43][44], pure block-FTL [43][44], DFTL [12] and FAST [15], and the allocation sub-module provides the choice of allocation schemes including the dynamic allocation and the static allocation. The bottom module simulates the behaviors of all the Flash operations based on the Open NAND Flash Interface Specification (ONFI) 2.2 [37]. This module supports four levels of parallelism and all advanced commands that are adhered to all aforementioned restrictions. By feeding block-level trace files and configuring with the parameter files, we can obtain the waiting time, processing time, response time of each request, total erasure count, buffer hit count and other detailed information.

In the design of SSDsim, we take into explicit account the time consumed by the necessary internal SSD software cost so as to achieve a high fidelity of the simulator. This, we argue, is one of the key features distinguishing SSDsim from the existing open-source SSD simulators. As a known fact in HDD, a typical request's response time is in the millisecond-scale, with negligible amount of program code being executed while processing the request. In SSD, on the other hand, a request's response time is in the microsecond-scale and there is significantly more program code being executed to service the request than in HDD, including address mapping, data merge and migration, among other things. Assuming a frequency of 100MHz for the SSD controller, one instruction executed by the controller will cost 10 nanoseconds. The time cost of 100 lines of assembly code will be about 1 microsecond, which is no longer negligible as part of SSD's response time. In fact, we found that the software processing cost as part of the response time of SSD is not only non-negligible, but actually a significant part of the response time. For example, in the response time of one read request, the software processing cost accounted for up to 18.9%.

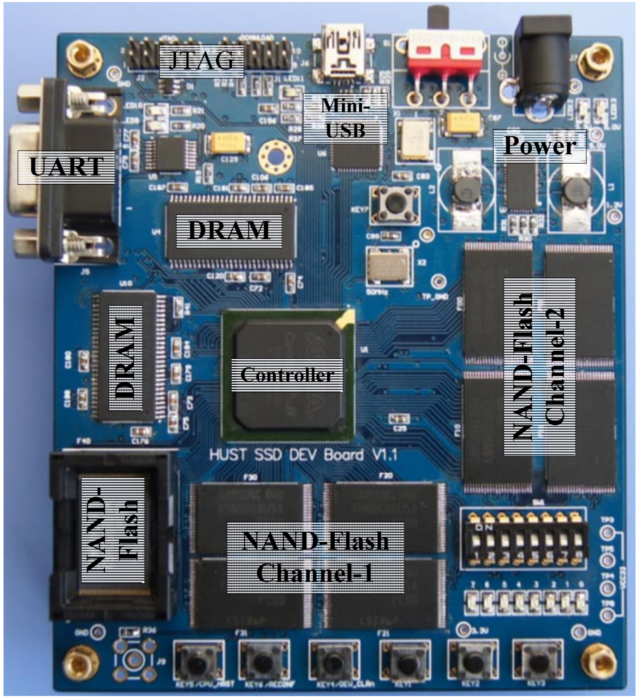


Figure 5. Real SSD hardware prototype.

To validate the accuracy of SSDsim, we have implemented a real SSD hardware prototype, as shown in Figure 5. In this prototype, an FPGA chip acts as the controller; eight SUMSANG flash chips [28] are organized into two independent channels, and four 16MB DRAM chips are used to store the mapping table and data buffer. For validation purposes, the same buffer management schemes, FTL and allocation schemes are implemented in the hardware prototype and SSDsim, the configuration parameter file based on the hardware prototype is fed to SSDsim, and the same request streams are fed to both the hardware prototype and SSDsim. Four workloads, which are detailed in Section 3.2 and reflect the high-performance computing environment with diverse write/read request ratios, request sizes, request characteristics, are used in our SSDsim validation.

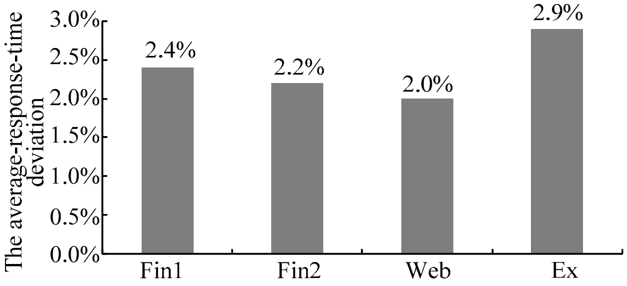


Figure 6. The average-response-time deviation of SSDsim from the prototype.

The main evaluation results from SSDsim and the hardware prototype are presented in Figures 6 and 7. In Figure 6, the average-response-time deviation of SSDsim from the prototype is plotted as a function of the four workloads. With a deviation of only 2%~2.9% shown in this figure, it is clear that the average response time obtained from SSDsim is very close to that obtained from the prototype, indicating the high accuracy of SSDsim.

Figure 7 plots the simulation accuracy as a Cumulative Distribution Function (CDF) of the response time. The sub-figure in each of the four parts of Figure 7 is a microscopic illustration of inflexion of each part. In Figure 7, the blue lines represent the prototype and the red lines represent SSDsim. It is evident that the two curves in each part almost completely overlap, suggesting that SSDsim matches extremely well with the prototype in the response-time measurement.

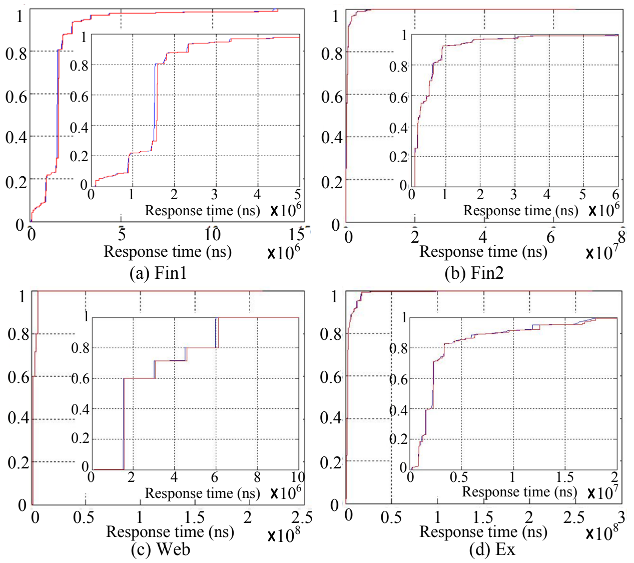


Figure 7. The Cumulative Distribution Function (CDF) of the response time.

3.2 Workloads

We use a set of real-world workloads, shown in Table 1 and reflecting the high-performance computing environments, to study the performance and endurance impacts of the internal behaviors/features of SSD, including the flash page size, allocation schemes, advanced commands, and their interplay. Financial1, Financial2 and Websearch were collected at a large financial institution and a popular Internet web search machine respectively [45]. Exchange [35] was collected at the Microsoft Exchange 2007 SP1 server, which is a mail server for 5000 corporate users. MSN [35] was collected at the Microsoft's several Live file servers. Develop [35] was obtained from a file server accessed by more than 3000 users to download various daily builds of Microsoft Visual Studio. Radius [35] was obtained from a RADIUS authentication server that is responsible for worldwide corporate remote access and wireless authentication. Table 1 summarizes the basic characteristics of these traces, including the average request size for reads and writes percentage of read requests, and request inter-arrival time.

Table 1. Workload characteristics of the traces

Workloads	Abb.	Avg. req. size read/write(KB)	Read(%)	Int. arrv. Time(ms)
Financial1	Fin1	2.25/3.75	23.2	8.19
Financial2	Fin2	2.3/2.9	82.3	11.08
Websearch	Web	15.15/8.6	99.9	2.99
Exchange	Ex	15.15/14.5	30.8	1179
MSN	MSN	9.6/11.1	67.2	513
Develop	Dev	18.45/10.95	88.6	1985
Radius	Rad	124.25/12.45	17.1	9475

3.3 Configuration of the Evaluation Platform

In our evaluation experiments, we assume a multiple-channel, multiple-package, multiple-chip, multiple-die and multiple-plane SSD organization. There are many ways to organize channels and packages based on the sharing methods of I/O bus, chip enable signal and ready/busy signal. Since our focus is on the use of SSD in the high-performance computing environment, we will concentrate on an organization that offers potentially the best performance. As shown in Figure 8, each channel has its independent I/O bus, and each chip has its independent chip enable signal and ready/busy signal. Both of them constitute the independent service units in the SSD. Since a package simply overlaps several chips, we only give sketches of chips without packages. The timing and organization characteristics of the configuration are based on a real NAND-Flash product [28], as summarized in Table 2. In the evaluations we also assume two types of SSD, the aged and the non-aged. While the former is used to show the case where a great number of physical pages have been previously written and thus garbage collection and erase operations are far more likely to be triggered by new requests, the latter is used to show the opposite situation. In particular, the former will allow us to examine the performance and endurance impact of garbage collection and erase operations that cause channels and chips to be in the busy state.

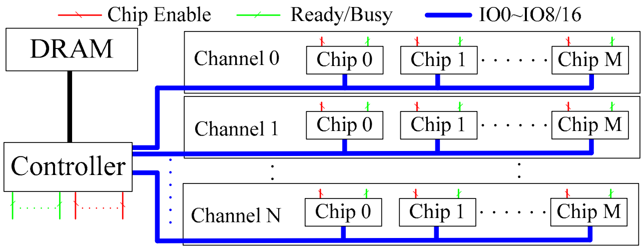


Figure 8. The evaluation platform of high-performance SSD.

Table 2. Configuration parameters used in SSDsim (Channel-Chip-Die-Plane-Block-Page indicates the number of channels in the SSD, chips in a channel, dies in a chip, planes in a die, blocks in a plane and pages in a block, respectively. Unless otherwise noted, they are default experiment parameters)

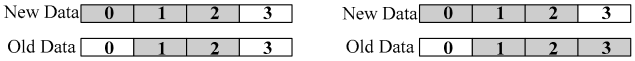
Parameters	Values
Page read to register	20us
Page write from register	200us
Block erase	1.5ms
Read one byte data from register	25ns
Write one byte data to register	25ns
Channel-Chip-Die-Plane-Block-Page	4-4-2-2-2048-64
Page size	2KB

4. EXPERIMENTAL EVALUATIONS

In this section we evaluate three SSD internal behaviors, or characteristics, which have notable impact on SSD performance and endurance, namely, (1) flash page size, (2) allocation schemes and (3) advanced commands. This is followed by a study on the interplay of these characteristics and the priority order of parallelism in SSD, presented in Section 4.4. We obtain our experimental results from SSDsim rather than the hardware SSD prototype. Since it is not easy to reconfigure some parameters of the hardware SSD prototype, such as the flash page size and the numbers of channels and chips, conducting experiments on the hardware SSD prototype is both costly and time consuming.

4.1 Flash Page Size

There are two scenarios in which a logical page is written to the flash memory, the logical page is written for the very first time and the logical page is rewritten or *updated* in the flash memory. The update operation can lead to two types of SSD internal data movement depending on whether the new data of the page fully overlaps, called "covered", or partially overlaps, called "un-covered", the old data of the page. Figure 9 shows the covered (Figure 9(a)) and un-covered (Figure 9(b)) cases of a page update assuming a page size of 2KB (i.e., the equivalent of 4 sectors), where the shaded sectors of the page represent valid data while the un-shaded sectors represent invalid data. In the case of a covered update, shown in Figure 9(a) the new data (sectors 0-2) is written to a new physical page, invalidating the old physical page containing the old data (sectors 1-2) of the logical page and modifying the mapping information of the logical page. In the case of an un-covered update, shown in Figure 9(b), the old data (sector 3) is read out to be combined with the new data (sectors 0-2) of the logical page before the combined data (sectors 0-3) is written to a new physical page, invaliding the old physical page containing the old data and modifying the mapping information of this logical page.



(a) Covered (b) Un-covered

Figure 9. The covered and un-covered update operations.

In other words, an un-covered operation requires one more flash read operation than a covered update operation, which can have a negative impact on the request's response time. Given the same average request size, the larger the page size is, the more likely it is for un-covered update operations to be induced. This is because, in a large-size page, a small write request will more likely find itself updating a subset of the sectors in a page with at least one valid sector to be combined, resulting in an un-covered update operation. On the other hand, a large page size also has its advantages, since a read/write operation in an SSD with a large page-size can fetch/send more data to/from the register, which allows a large read/write request to be more efficiently executed.

In Figure 10, we plot the percentage of un-covered update operations among all write operations, shown as the line plots and labeled on the Y-axis on the right side of the figure, as a function of the flash page-size under five different workloads. The performance impact of these un-covered update operations, measured in the average response time normalized to that of the best performing flash page-size, shown as the bars and labeled on the Y-axis on the left side of the figure, is plotted as a function of the flash page sizes under five different workloads.

From Figure 10, it is clear that, under the Dev, MSN, Ex and Rad workloads, the page size of 4KB results in the best average response time. This is because 78.6%, 74.3%, 99.2% and 33.4% of write requests in MSN, Ex, Dev and Rad, respectively, are of size 4KB or the multiples of 4KB, which induce fewer un-covered update operations than the 8KB-page and 16KB-page SSDs. The 4KB page-size also results in the least average response time under these four workloads. Although the un-covered update operation count in the 2KB-page SSD is smaller than that in the 4KB-page SSD, a 4KB write request in the 2KB-page SSD requires two or three write operations, in contrast to the one or two write operations required in the 4KB-page SSD, giving rise to a higher average response time in the former. Under the Fin1 workload, the average response time is the best in the 16KB-page SSD, because the un-covered update-operation count changes very little with an increase in the page size while the larger page size favors the large requests in this workload.

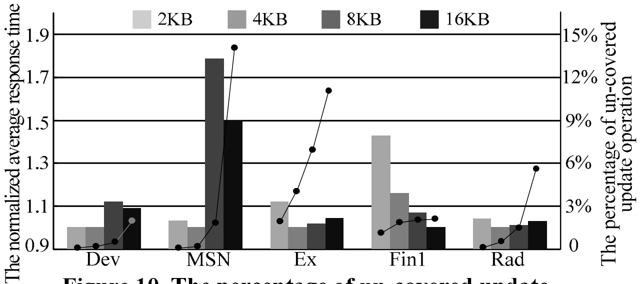


Figure 10. The percentage of un-covered update operations and normalized average response time as functions of flash page-size under different workloads.

Insight 1: Enlarging the storage capacity of SSD by means of increasing the page size may not be a wise choice under some workloads. Instead, we argue that a better choice for large capacity and stable performance is to use more packages with an appropriate page size in the same channel, or overlap more chips of an appropriate page size in a flash package. Since the controller provides a chip enable signal and a busy/ready signal to each chip, the storage capacity of SSD can be enlarged without decreasing the I/O performance by increasing some control signals of the controller in this way. Further, to design high-performance and large-capacity SSDs, the request size and the percentage of the un-covered update operations of the workload must be taken into account to choose flash chip with an appropriate page size.

4.2 Allocation Schemes

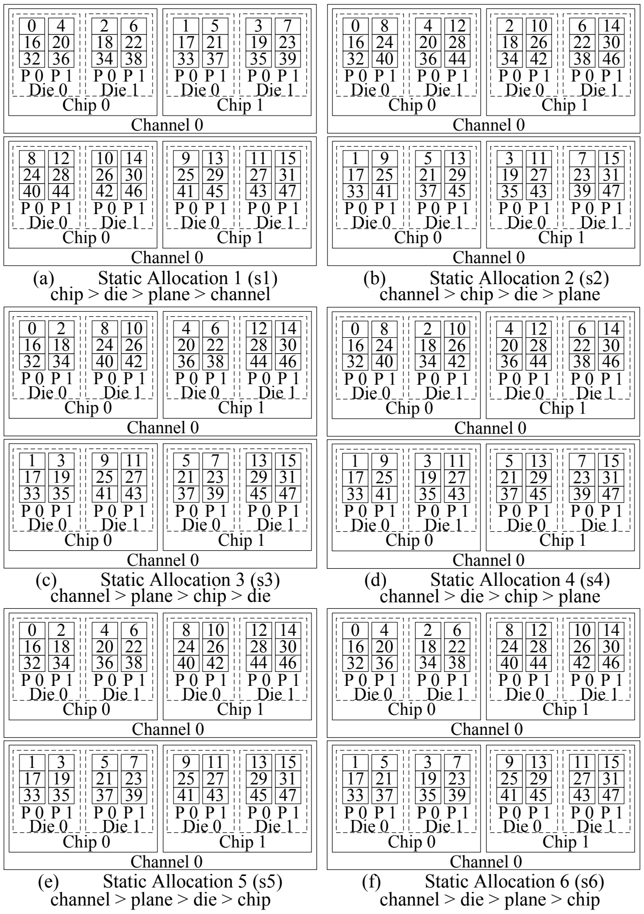
In this subsection, we compare the performances of the static allocation and dynamic allocation schemes, and evaluate the related wear-leveling issues.

Figure 11 illustrates six different static allocation schemes (including static allocation 2 proposed by J. Shin et al. [14]), which are referred to as s1, s2, s3, s4, s5 and s6.

As mentioned in Section 2.3, dynamic allocation schemes assign a logical page to any free physical page of the entire SSD or the pre-determined channel, according to the idle/busy state of channels, the idle/busy state of chips, the erasure count of blocks, and the priority order of parallelism. Different combinations of these factors will derive many different dynamic allocation schemes. For example, when a 4KB write request arrives in an SSD of 2KB-page-size Flash, assuming that 2 channels as well as 2 chips in each channel are idle, the request can be served by 2 channels or 2 chips in a channel, when applying different priority order of parallelism. The former is channel-level parallelism first, and the latter is chip-level parallelism first. Since we discuss advanced commands and priority order of parallelism in SSD in Sections 4.3 and 4.4, respectively, die-level parallelism and plane-level parallelism will not be explored in this section. In other words, the multi-plane and the interleave advanced commands will not be used, and the priority order of parallelisms will be channel-level parallelism first, followed by the chip-level parallelism.

We compare the average response time of the dynamic allocation scheme with that of s2, because s2 is shown to achieve the best performance when advanced commands are not employed [14].

The read/write/overall performance impact of different allocation schemes, measured in the average response time normalized to that of the s2 is plotted in Figure 12, under six different workloads.



(a) Static Allocation 1 (s1) chip > die > plane > channel

(b) Static Allocation 2 (s2) channel > chip > die > plane

(c) Static Allocation 3 (s3) channel > plane > chip > die

(d) Static Allocation 4 (s4) channel > die > chip > plane

(e) Static Allocation 5 (s5) channel > plane > die > chip

(f) Static Allocation 6 (s6) channel > die > plane > chip

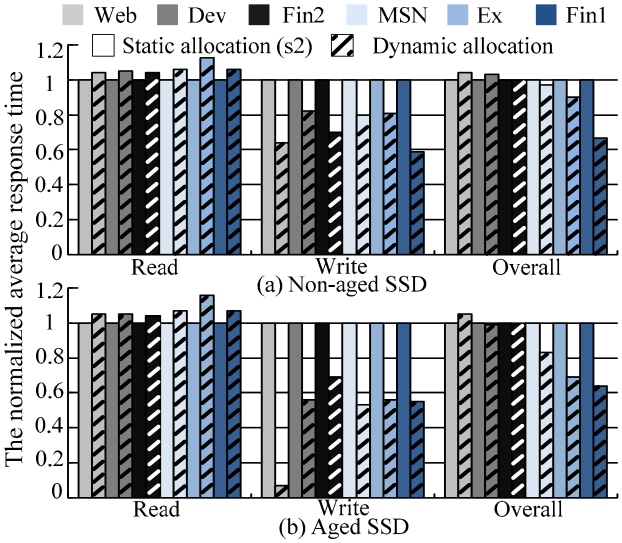
Figure 11. Six kinds of typical static allocation schemes. (A > B > C > D means the priority order of allocating logical page. In other word, it is striping address to A first, then to B, then to C, and finally to D.)

From Figure 12, it is clear that the static allocation scheme performs the best for read requests in both the non-aged SSD and the aged SSD, under all workloads. For a given read request whose size is a multiple of a logical page size, the striping nature of the static allocation is likely to distribute the sequential logical pages of the request to different channels and chips, which tends to exploit more parallelisms of the multi-channel and multi-chip structured SSD, thus decreasing the response time of this request. In the dynamic allocation, on the other hand, it is entirely possible that the sequential logical pages are stored in the same channel, or even the same chip, so that these sequential logical pages will be operated in the same channel or chip one by one, failing to exploit the parallelism of SSD. Since 99.99% of the requests are read requests in the Web workload, the overall performance of the static allocation is better than dynamic allocation, in both the non-aged SSD and the aged SSD.

Insight 2: The static allocation scheme consistently outperforms the dynamic allocation scheme in serving read requests. Thus, in the application environments that demand fast reads, or are read-dominant in their workloads, the static allocation scheme should be employed.

For the non-aged SSD, Figure 12 (a) shows that the dynamic allocation scheme outperforms the static allocation scheme under all workloads for write requests and overall under the Fin2, MSN, Ex, and Fin1 workloads. This is because the sequential logical pages of a multi-page write request are likely to be serviced by multiple chips in several channels in the static allocation, while the response time of the request is determined by the logical page completing the last. If any one of the logical pages happens to be on a busy chip, which is very likely, the response time of the request will be severely delayed. This, however, does not happen in the dynamic allocation, since write requests can be adaptively distributed to idle chips.

In the aged SSD, the write-performance advantage of the dynamic allocation scheme becomes more pronounced, as shown in Figure 12 (b). This is because there are more garbage collection and erase operations in an aged SSD than in a non-aged SSD, which can cause more chips to be in the busy state and further decrease the write performance of the static allocation scheme.



(a) Non-aged SSD

(b) Aged SSD

Figure 12. The normalized average response time when using static allocation and dynamic allocation schemes, in non-aged and aged SSDs, under the six workloads.

Insight 3: In a non-aged SSD, the static allocation scheme is preferable when the workload is read-dominant. Otherwise, the dynamic allocation scheme should be employed. In an aged SSD, the dynamic allocation scheme consistently outperforms of the static allocation scheme, with the only exception being the read-only workloads.

Wear-leveling algorithms are used to distribute the erase operations evenly to the entire SSD for the purpose of enhancing flash endurance. To balance erasure count, wear-leveling usually writes hot data to the least frequently erased blocks and migrates cool data to blocks with higher erasure counts [46]. Obviously, such data migrations will lead to extra read write and erase operations that have negative impact on performance and endurance. In Table 3, we list the standard deviation of the total erasure counts of blocks in each plane for the static and dynamic allocation schemes under the five workloads, where a low standard deviation indicates a more evenly distributed erase operations. It is clear from the table that the dynamic scheme has a much better wear-leveling performance than the static scheme.

Insight 4: The dynamic allocation scheme consistently outperforms the static scheme on the wear-leveling performance.

Table 3. The standard deviations of total erasure count of blocks in each plane when employing either static allocation scheme or dynamic allocation scheme.

Workloads	Static allocation	Dynamic allocation
Dev	284.9	2.5
Ex	409.1	39.5
Fin1	207.3	3.9
MSN	3534.4	112.6
Rad	7.5	2.6

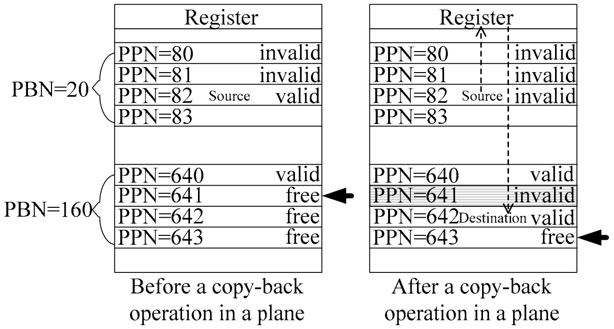
4.3 Advanced Commands

In this subsection, we evaluate the impact of the advanced commands provided by Flash manufacturers, and how Restrictions (a)-(c) make these advanced commands a double-edged sword.

To better examine the performance impact of the multi-plane read/write/erase commands that exploit the plane-level parallelism and the interleave read/write/erase commands that exploit the die-level parallelism, we exclude the interference of the channel-level parallelism by employing a single-channel SSD in the experiments of this section.

4.3.1 Copy-back

When using the copy-back command, Restriction (a) and Restriction (b) must be adhered to. Figure 13 illustrates the process of executing a copy-back command, where the data stored in PPN =82 needs to be migrated to a free physical page. Since the pages in a block must be programmed sequentially, the next available page is PPN =641. However, Restriction (b) forbids us to write the data to PPN =641, forcing the invalidation of PPN =641 and migration of the data into PPN =642. It is obvious that using the copy-back command blindly will lead to a waste of flash pages. In fact, our experiments reveal that using the copy-back command blindly can cause almost half of the copy-back operations to each invalidate one extra page under all workloads.



Before a copy-back operation in a plane

After a copy-back operation in a plane

Figure 13. The exemplar process of executing a copy-back command.

To use the copy-back command wisely to minimize the number of invalidated pages, we recommend its use only when the addresses of the source page and the destination page have the same parity.

The performance impact of the way in which the copy-back command is used, measured in the average response time normalized to that of only using the basic commands, is plotted as a function of the workloads and labeled on the Y-axis on the left side of Figure 14. The erasure count of using the copy-back command blindly, normalized to that of only using basic commands, is shown by small triangles and labeled on the Y-axis on the right side of the figure. In the experiments presented in this subsection, the dynamic allocation scheme employs the same priority order of parallelisms as that used in Section 4.2.

In Figure 14, it is clear that using the copy-back command blindly has a notable negative impact on the average response time and the erasure count measures under the Dev, MSN and Ex workloads. This is because a large number of the copy-back commands lead to many pages being invalidated under these workloads, which in turn trigger frequent garbage collections. During a garbage collection, more copy-back operations and erase operations will be performed, which further decreases the overall performance and increases erasure count. On the other hand, using the copy-back command wisely does improve performance without increasing the erasure count. This is because there are no extra pages invalidated and no extra erasure operation induced.

In Figure 14, we only present the results of an aged SSD, since garbage collections are rarely triggered in a non-aged SSD.

Insight 5: The copy-back command should only be used wisely when the addresses of the source page and the destination page have the same parity, otherwise the I/O performance and endurance of SSD can be significantly reduced.

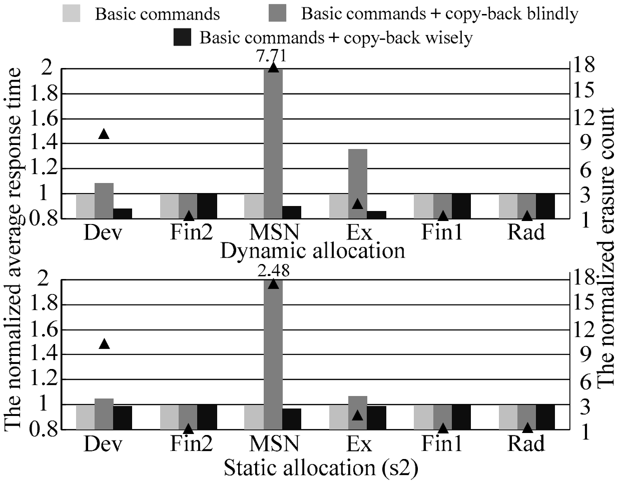


Figure 14. The performance comparison of different the methods of using copy-back command.

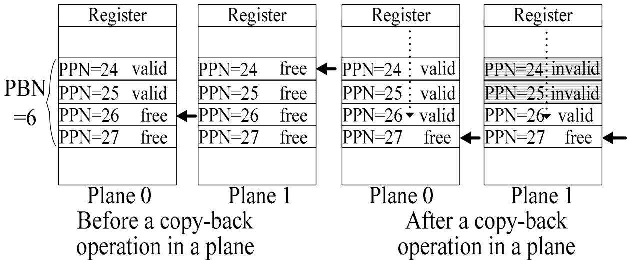
4.3.2 Multi-Plane

In this section, we analyze the multi-plane command for reads and writes, which we call MPW (Multi-Plane Write) and MPR (Multi-Plane Read) for short in the remainder of the paper.

As mentioned in Section 2.2, a multi-plane command can execute the same basic command in all planes on the same die. Therefore, it exploits the parallelism among the planes of the same die. When using multi-plane write command, Restriction (a) and Restriction (c) must be adhered to.

Figure 15 illustrates an MPW operation. Two different planes of the same die, plane 0 and plane 1, are shown in the figure. The page address of the next available page in plane 0 is 26 while that in plane 1 is 24. When using the MPW command in these two planes, PPN =24 and PPN =25 in plane 1 will be invalidated. Therefore, in this case, executing the MPW command invalidates (and wastes) two free pages.

Similar to the copy-back command, the MPW command can be used blindly or wisely. In Figure 16, we plot the average response times and erasure count of the MPW command in the same way as in Figure 14. In the experiments of this sub-section, the dynamic allocation scheme exploits MPW/MPR and adheres to the parallelism priority order of channel-level first, plane-level second and chip-level last. The static scheme s3 performs the best I/O performance when employing MPW/MPR among all the static allocation schemes.



Before a copy-back operation in a plane

After a copy-back operation in a plane

Figure 15. The exemplar process of executing a MPW.

As shown in the upper sub-figure of Figure 16, when employing the dynamic allocation, using MPW blindly improves average response time over basic commands. However, a large number of free pages are invalidated, which leads to more extra erase operations. Note that the benefit of plane-level parallelism outweighs the loss caused by the extra erase operations. Therefore using MPW blindly can improve response time under all workloads. On the other hand, since the condition required for the wise MPW, i.e., the target pages executing an MPW must have the same chip, die, block and page addresses, can rarely be met, the improvement by the wise MPW is insignificant.

As shown in the bottom sub-figure of Figure 16, when employing the static allocation, MPW has a similar performance to that in the dynamic allocation.

We only show the experiment results in an aged SSD because using MPW blindly will not likely trigger garbage collection and resulting erase operations.

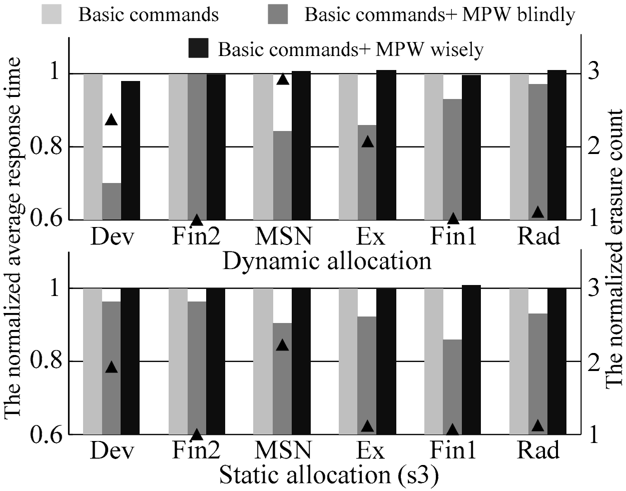


Figure 16. Performance comparison of different the methods of using MPW.

Insight 6: Using MPW blindly improves the I/O performance but reduces the endurance under most workloads. The impact of the wise MPW is negligible because the condition required for its application can rarely be met.

MPR performs multiple-page read operations in different planes of the same die simultaneously. When using MPR, Restriction (c) must be adhered to. In Table 4, we list the performance gains due to MPR under the dynamic and static allocation schemes, respectively. We found that the performance gains are negligible under a majority of the workloads. A speedup of only 1.16 is observed for a two-page MPR command. Moreover, since Ex and Fin1 are write-dominant workloads, the performance gain due to MPR is negligible. Since the request size of Fin2 is too small to be striped onto multiple pages, MPR is not applicable there. Under Web, Dev and MSN, the performance gains will be higher compared to other workloads, since these three workloads are read-dominant, whose request sizes are multiples of a flash page size.

Table 4. The performance gain due to MPR (RS is short for response time speedup. Baseline is based on basic command alone.**)**

	Web	Dev	Fin2	MSN	Ex	Fin1
Dynamic RS	1.15	1.11	1.02	1.04	1.00	1.00
Static RS	1.09	1.01	1.00	1.01	1.00	1.00

Insight 7 : MPR cannot provide significant performance improvement, under most workloads. But in the application environments whose workloads are read-dominant and comprise of large reads (i.e., Web, Dev and MSN), using MPR can help improve I/O performance.

In addition to MPW and MPR, the multi-plane command can also activate multiple erase operations in all planes of the same die. However, since the extent to which the erase operations are triggered in all planes of the same die at the same time is heavily dependent on the specific garbage collection algorithm and weal-leveling algorithm used, which are beyond the scope of this paper, we will not evaluate the impact due to the multi-plane erase command independently.

4.3.3 Interleave

The interleave command exploits the parallelism among dies on the same chip. Pages and blocks from different dies on a chip can be read/written and erased simultaneously by executing an interleave command. The command is different from other advanced commands in that only Restriction (a) must be adhered to. Therefore, there is no endurance loss when using the interleave command, unlike using other advanced commands.

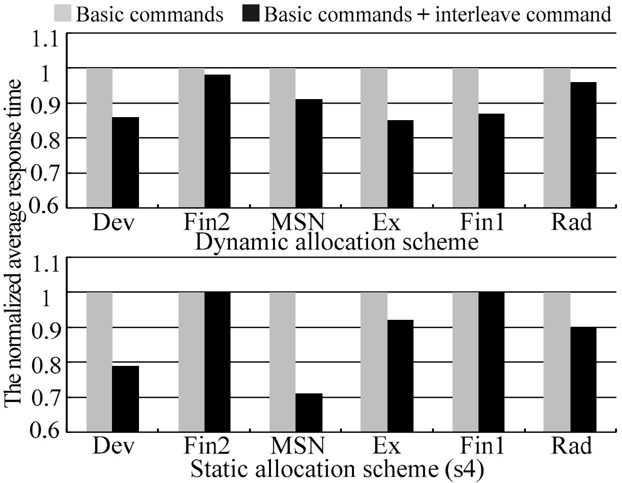


Figure 17. The performance comparisons of employing interleave command or not.

We plot the performance gain due to the interleave read/write/erase command as a function of the workloads in Figure 17, measured in the average response time normalized to that based on basic commands. In the experiments of this sub-section, the dynamic allocation scheme exploits the interleave command and adheres to the parallelism priority order of channel-level first, die-level second and chip-level last. The static scheme s4 performs the best when employing the interleave command among all the static allocation schemes.

The results from the figure show that, while the I/O performance is improved, SSD endurance is not notably impacted. The only exceptions are Fin1 and Fin2, where no significant performance gains are observed when using the static allocation, since the request sizes of these two workloads are small, thus depriving the interleave command the opportunity to be applicable.

Insight 8: The interleave command can help improve the I/O performance without any endurance degradation. Therefore the interleave command should be applied under all circumstances.

For the same reason given to the case of the multi-plane erase command at the end of Section 4.3.2, we will not evaluate the impact of using the interleave erase command independently. The interleave command can be combined with MPW and MPR, which we will discuss next.

4.3.4 The combinations of the three advanced commands

In this sub-section, we employ the three advanced commands simultaneously, and evaluate their combined impacts on the performance and endurance of SSDs. Based on Insights 5-8, there are two recommended approaches to using the advanced commands, namely, (1) use the copy-back command, the MPW command, and the interleave command wisely (i.e., with matching parity in addresses) under all circumstances; and (2) use the copy-back command wisely, the MPW command blindly, and the interleave command ubiquitously.

In Figure 18, we plot the average response time and erasure count of using advanced commands in the two recommended ways as a function of workloads, in the same way in which Figures 14, 16 are plotted. We only display the results of the static scheme s6 that is shown to achieve the best performance when the advanced commands are employed. We also only list the experimental results in an aged SSD for the reason discussed earlier.

From the figure, we found that the combined use of advanced commands based on Approach (2) achieves the best performance but leads to SSD endurance degradation, while Approach (1) achieves the less performance gain but without any endurance loss.

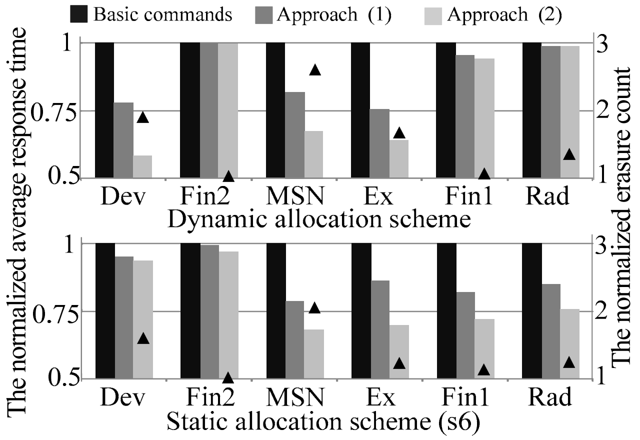


Figure 18. The performance comparison of two approaches to using the advanced commands.

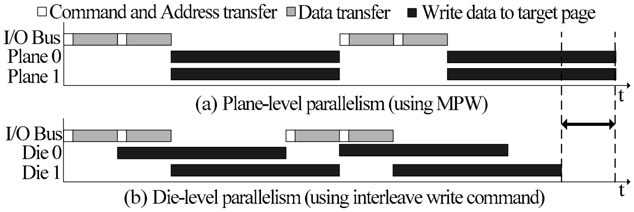
4.4 Priority Order of Parallelism in SSD

As discussed in Section 2.4, there are four levels of parallelism in SSD, namely, channel-level, chip-level, die-level and plane-level. To determine the priority order of these levels that optimizes the performance and endurance of SSD, we first infer the optimal priority order qualitatively, and then confirm the optimality quantitatively by a series of experiments with different allocation schemes.

Strictly speaking, each read/write operation consists of two steps, (1) data transfer and (2) reading/writing data from/to the target page to/from the data register of the plane. The aim of parallelism is to overlap or pipeline these two steps. Chip-level parallelism, die-level parallelism and plane-level parallelism are executed on the same channel, which share the same channel bus. As a result, these three levels of parallelism can only overlap or pipeline step (2) of an operation. On the other hand, the channel-level parallelism overlaps not only step (2), but also step (1) of an operation. Therefore channel-level parallelism should be given the highest priority among the four levels of parallelism.

Chip-level parallelism renders multiple chips busy. When the chips on the channel are servicing requests, the subsequent requests cannot be serviced until these chips return to the idle state. On the other hand, die-level parallelism and plane-level parallelism only involve a single chip, thus making them a higher priority than the chip-level parallelism.

As shown in Figure 19, to serve a four-page-write request, two MPW operations are executed when exploiting the plane-level parallelism. To exploit the die-level parallelism, however, two interleave write commands are executed. From the figure, we find that the latter to be superior to the former. Moreover, exploiting the plane-level parallelism requires the execution of the MPW/MPR command, which often invalidates free pages. On the contrary, the interleave command required for exploiting the die-level parallelism has no such disadvantages. Therefore, die-level parallelism should be given a higher priority than plane-level parallelism.



(a) Plane-level parallelism (using MPW)

(b) Die-level parallelism (using interleave write command)

Figure 19. The plane-level parallelism compares to die-level parallelism

4.4.1 Evaluation of priority order of SSD parallelism under the dynamic allocation

In this sub-section, we use six different SSDs to conduct a set of experiments to evaluate the priority order of SSD parallelism. The configuration parameters of the six SSDs are shown in Table 5.

Table 5. Six kinds of configured SSDs. (A>B in the "Priority" field signifies that choosing a free page from A is preferred to choosing one from B. Cl.-Cp.-D.-P. indicates the numbers of channels in the SSD, chips in a channel, dies in a chip, and planes in a die, respectively. The "AC" row indicates whether advanced commands are used (Yes) or not (No))

SSD	Cl.-Cp.-D.-P.	AC	Page	Priority
SSD1	8-4-2-2	Yes	2KB	chip >die>plane>channel
SSD2	8-4-2-2	Yes	2KB	channel> chip>die>plane
SSD3	1-4-2-2	Yes	2KB	channel>chip>die>plane
SSD4	1-4-2-2	Yes	2KB	channel>die>chip>plane
SSD5	1-4-2-2	Yes	2KB	channel>die>plane>chip
SSD6	1-4-2-2	Yes	2KB	channel>plane>die>chip

The hardware organizations of SSD1 and SSD2 (see Table 5) are different to those of SSD3, SSD4, SSD5 and SSD6, thus we compare their performance in two separated sub-figures (Figure 20 (a) and Figure 20 (b)). Figure 20 (a) shows that SSD2 outperforms SSD1 consistently. In SSD2, we distribute the requests to different channels. When 8 channels are deployed, steps (1) and (2) of an operation can be perfectly overlapped under all workloads. In SSD1, several pages of a request are distributed to some chips of the same channel, which results in multiple data transfers (i.e., step (1)) and one reading/writing flash media (i.e., step (2)). This explains SSD2's superiority to SSD1 and confirms quantitatively that channel-level parallelism should be given the first priority.

Figure 20 (b) shows that SSD4 consistently outperforms SSD3. While SSD3 prefers the chip-level parallelism to the die-level parallelism, the reverse is true for SSD4. Thus, when a request involving two pages is served by SSD3, two chips become busy. On the contrary, only one chip becomes busy in SSD4, allowing SSD4 to serve more subsequent requests than SSD3. This confirms that the die-level parallelism should be given a higher priority than the chip-level parallelism.

SSD5 outperforms SSD6 because the former uses an interleave write operation while the latter employs an MPW operation when a request that needs to write two pages arrives. Since using MPW blindly leads to render free pages invalidated, as discussed in Section 4.3.2, more erase operations will be triggered in SSD6 than in SSD5. Therefore, the die-level parallelism must be given a higher priority than the plane-level parallelism.

SSD5 is superior to SSD4, because a request that needs to read/write four pages can render two chips busy in SSD4 by executing two consecutive interleave read/write operations. On the other hand, for the same request only one chip is rendered busy in

SSD5 with the execution of a single interleave multi-plane read/write operation. Therefore, the priority of the chip-level parallelism should be the lowest.

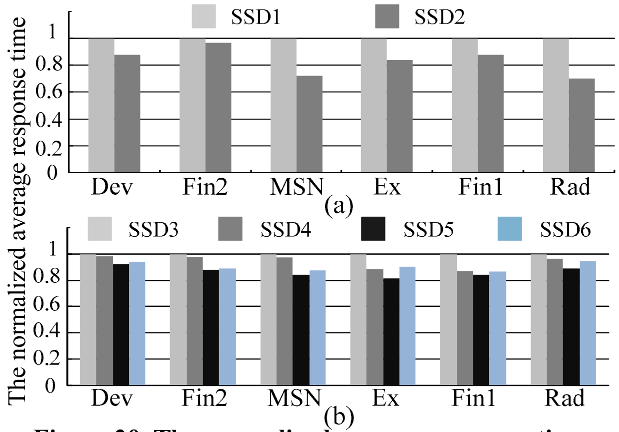


Figure 20. The normalized average response time of SSD1, SSD2, SSD3, SSD4, SSD5 and SSD6.

4.4.2 Evaluation of priority order under the static allocation

In the experiments of this subsection, we use six SSDs, SSD-s1, SSD-s2, SSD-s3, SSD-s4, SSD-s5 and SSD-s6, that employ six different static allocation schemes, s1, s2, s3, s4, s5, s6, as shown in Figure 11. These six SSDs share the following common configuration parameters: 8 channels in the SSDs, 4 chips in each channel, 2 dies on each chip, 2 plane on each die, 2048-block planes on each die, and each block contains 64 2KB pages. All advanced commands are used. In addition to the six real-world workloads listed in Table 1, we use a set of synthetic workloads in our experiments, whose key characteristics are shown in Table 6.

Table 6. The characteristics of synthetic workloads

Workload	Write ratio	Req. size	Interval time
Syn1	100%	16KB	30us (75%)
Syn2	25%	16KB	30us (75%)
Syn3	100%	20KB	200us (75%)
Syn4	25%	20KB	200us(75%)

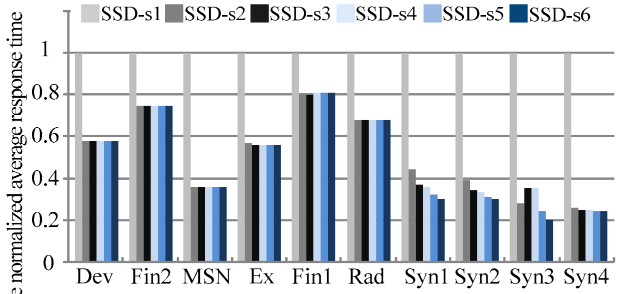


Figure 21. The normalized average response time of SSD-s1, SSD-s2, SSD-s3, SSD-s4, SSD-s5, SSD-s6.

The performance comparisons of SSD-s1, SSD-s2, SSD-s3, SSD-s4, SSD-s5 and SSD-s6 are shown in Figure 21. We found that SSD-s1 performs the worst among all the SSDs. It further confirms that the channel-level parallelism should be given the highest priority. With the exception of SSD-s1, all SSDs perform almost the equal under the real-world workloads. This is because the request intensities of these workloads are relatively low, which can be fully served by the channel-level parallelism. Therefore we use a set of higher-intensity synthetic workloads in our experiments. Under these synthetic workloads, SSD-s6 performs the best, since the allocation scheme of SSD-s6 is adhered to the priority order of parallelism inferred at the beginning of Section 4.4.

Insight 9: The optimal priority order of parallelisms in SSD should be (1) the channel-level parallelism, (2) the die-level parallelism, (3) the plane-level parallelism, and (4) the chip-level parallelism.

5. CONCLUSION

We presented and validated an event-driven, modularly structured, multi-tiered and high accuracy SSD simulator, called SSDsim. Through extensive performance analysis conducted on SSDsim, we obtained important insights into the design and use of SSDs. Based on these insights, we argue that Flash page sizes, allocation schemes, advanced commands and the priority order of SSD parallelisms have significantly important impacts on the performance and endurance of SSD. More specifically, from the in-depth evaluations of these features and their interplay, our work provides the following important insights: (1) to design high-performance and large-capacity SSDs, the request size and the percentage of the un-covered update operations of the workload must be taken into considerations to choose an appropriate page size; (2) the static allocation is found to perform the best on read performance under all workloads. The dynamic allocation performs the best on overall performance and endurance under the most of workloads in aged SSDs; (3) there are two recommended approaches to using the advanced commands, namely, use the copy-back command, the MPW command, and the interleave command wisely as well as use the copy-back command wisely, the MPW command blindly, and the interleave command ubiquitously; (4) the optimal priority order of parallelisms in SSD should be the channel-level parallelism first, the die-level parallelism second, the plane-level parallelism third, and the chip-level parallelism last.

6. AVAILABILITY

We intend to release SSDsim source code for public use in the near future. Please check http://storage.hust.edu.cn/SSDsim to obtain a copy.

7. ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments. This research was partially supported by the National Basic Research 973 Program of China under Grant N o. 2011CB302301, 863 project 2009AA01A402, NSFC No.61025 008, 60933002,60873028,60703046, Changjiang innovative group of Education of China No. IRT0725, US NSF under Grants IIS-0916859, CCF-0937993, and CNS-1016609.

8. REFERENCES

[1] A. R. Olson and D. J. Langlois. 2008. Solid State Drives Data Reliability and Lifetime. White Paper. Imation Corp. http://www.imation.com/PageFiles/1189/SSD_Gov_DataReliability_WP.pdf

[2] W. Hutsell, J. Bowen and N. Ekker. 2008. Flash Solid State Disk Reliability. White Paper. Texas Memory Systems. http://www.ramsan.com/files/f000252.pdf

[3] M-System. Two Technologies Compared: NOR vs NAND. In white paper, 2003. http://maltiel-consulting.com/Nonvolatile_Memory_NOR_vs_NAND.pdf

[4] SLV vs. MLC: An Analysis of Flash Memory. In white paper. Super Talent Technology, Inc. http://www.supertalent.com/datasheets/SLC_vs_MLC%20whitepaper.pdf

[5] J. Cooke. Introduction to Flash Memory (T1A). Slides. 2008. http://www.slideshare.net/Flashdomain/introduction-to-flash-memory-t1a

[6] K. M. Greenan, D. D. E. Long, E. L. Miller, T. Schwarz and A. Wildani. Building Flexible, Fault-Tolerant Flash-based Storage Systems. In *Proc. of* HotDep’09, June 2009.

[7] M. Moshayedi and P Wilkison. Enterprise SSDs. ACM QUEUE. July/August 2008

[8] C. Dirik and B. Jacob. The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization. In *Proc. of* ISCA’09, June 2009.

[9] A. M. Caulfield, J. Coburn, T. I. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. K. Gupta, A. Snavely and S. Swanson. Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing. In *Proc. of* SC’10, November 2010.

[10] A. Leventhal. Flash Storage Today. ACM QUEUE. July/August 2008.

[11] G. Graefe. The Five-minute Rule Twenty Years Later, and How Flash Memory Changes the Rules. In *Proc. of* DaMoN’07. June 15, 2007

[12] A. Gupta, Y. Kim and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective of Page-level Address Mapping. In *Proc. of* ASPLOS’09. March 7-11, 2009.

[13] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin and L. Wang. Achieving Page-Mapping FTL Performance at Block-Mapping FTL Cost by Hiding Address Translation. In *Proc. of* MSST’10. May 3-7, 2010.

[14] J. Shin, Z. Xia, N. Xu, R. Gao, X. Cai, S. Maeng and E. Hsu. FTL Design Exploration in Reconfigurable High-Performance SSD for Server Applications. In *Proc. of* ICS’09, June 2009.

[15] S. Lee, D. Park, T. Chung, D. Lee, S. Park and H. Song. A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation. ACM Transactions on Embedded Computing Systems, Vol.6, No.3, Article 18, July 2007.

[16] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proc. of* FAST’08. February 26-29, 2008.

[17] J. Seol, H. Shim, J. Kim and S. Maeng. A buffer replacement algorithm exploiting multi-chip parallelism in solid state disks. In *Proc. of* CASES’09, October 2009.

[18] S. Park, D. Jung, J. Kang, J. Kim and J. Lee. CFLRU: A Replacement Algorithm for Flash Memory. In *Proc. of* CASES’06, October 2006.

[19] H. Jo, J. Kang, S Park, J. Kim and J. Lee. FAB: Flash-Aware Buffer Management Policy for Portable Media Players. IEEE Transaction on Consumer Electronics, Vol.52, No.2, M ay 2006.

[20] F. Chen, D. A. Koufaty and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *Proc. of* SIGMETRICS /performance’09. June 15-19, 2009

[21] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel and J. K. Wolf. Characterizing Flash Memory: Anomalies, Observations, and Applications. In *Proc. of* MICRO’09. December 12-16, 2009.

[22] S. Boboila and P. Desnoyers. Write Endurance in Flash Drives: Measurements and Analysis. In *Proc. of* FAST’10. February 23-26, 2010.

[23] P. Desnoyers. Empirical Evaluation of NAND Flash Memory Performance. In *Proc. of* HotStorage’09, October 2009.

[24] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proc. of* USENIX’08, June 2008

[25] J. Kang, J. Kim, C. Park, H. Park and J. Lee. A multi-channel architecture for high-performance and flash-based storage system. Joumal of Systems Architecture. 53: 644-658, 2007.

[26] S. Park, E. Seo, J. Shin, S. Maeng and J. Lee. Exploiting internal parallelism of flash-based SSDs. IEEE Computer Architecture Letters. 03-Feb-2010.

[27] S. Park, S. Ha, K. Bang and E. Chuang. Design and analysis of flash translation layers for multi-channel NAND flash based storage devices. IEEE Transaction on Consumer Electronics, Vol.55, No.3, August 2009.

[28] K9XXG08UXA datasheet. http://www.samsung.com/products/semiconductor/flash/technicallinfo/datasheets.htm.

[29] K9NCG08U5M datasheet. http://www.samsung.com/products/semiconductor/flash/technicallinfo/datasheets.htm.

[30] Micro MT29F16G08FAA NAND Flash Memory datasheet. http://www.micron.com//document_download/?documentId=4308

[31] Micro MT29F256G08CUCBB NAND Flash Memory datasheet. http://www.micron.com//document_download/?documentId=4368

[32] Intel JS29F64G08CAMD1 MD332 NAND Flash Memory datasheet. http://www.intel.com/design

[33] Toshiba TH58TVG7S2F NAND Flash Memory datasheet. http://www.toshiba.com/

[34] Hynix H27UCG8U5(D)A Series 64Gb NAND Flash datasheet. http://www.hynix.com/datasheet/

[35] Microsoft Enterprise Traces. http://iotta.snia.org/traces/list/BlockIO

[36] Application note for nand flash memory (revision 2.0) http://www.samsung.com/global/business/semiconductor/products/flash/downloads/applicationnote/app_nand.pdf

[37] Open NAND Flash Interface SpecificaRion. revision2.2.http://onfi.org/wp-content/uploads/2009/02/ ONFI%202_2%20 Gold.pdf

[38] NAND Flash Performance Improvement Using Internal Data Move. Technical Note TN-29-15. http://download.micron.com/pdf/technotes/nand/tn2915.pdf

[39] Using COPYBACK Operations to Maintain Data Integrity in NAND Devices. Technical Note TN-29-41. http://www.eetasia.com/STATIC/PDF/200903/EEOL_2009MAR02_STOR_AN_01.pdf?SOURCES=DOWNLOAD

[40] SSD Extension for DiskSim Simulation Environment. http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4/

[41] Y. Kim, B.Tauras, A. Gupta, D. M. Nistor and B. Urgaonkar. FlashSim: A Simulator for NAND Flash-based Solid-State Drives. Technical Report CSE-09-008

[42] J. Bucy, J. Schindler, S. W. Schlosser and G. R. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual. May 2008.

[43] E. Fal and S. Toledo. Algorithms and Data Structures for Flash Memories. ACM Computing Surveys, Vol.37, No.2, June 2005, pp.138-163.

[44] T. Chung, D. Park, S. Park, D. Lee, S. Lee and H. Song. System Software for Flash Memory: A Survey. International Federation for Information Processing 2006. EUC 2006, LNCS 4096, pp. 394-404. 2006.

[45] UMass Trace Repository. http://traces.cs.umass.edu

[46] Weal-Leveling Techniques in NAND Flash Devices. Technical Note TN-29-42. http://download.micron.com/pdf/technotes/nand/tn2942_nand_wear_leveling.pdf