Exploring and Exploiting the Multilevel Parallelism Inside SSDs for Improved Performance and Endurance

Yang Hu, Hong Jiang, *Senior Member*, *IEEE*, Dan Feng, *Member*, *IEEE*, Lei Tian, *Member*, *IEEE*, Hao Luo, and Chao Ren

Abstract—Given the multilevel internal SSD parallelism at the different four levels: channel-level, chip-level, die-level, and plane-level, how to exploit these levels of parallelism will directly and significantly impact the performance and endurance of SSDs, which is in turn primarily determined by three internal factors, namely, advanced commands, allocation schemes, and the priority order of exploiting the four levels of parallelism. In this paper, we analyze these internal factors to characterize their impacts, interplay, and parallelism for the purpose of performance and endurance enhancement of SSDs through an in-depth experimental study. We come to the following key conclusions: 1) Different advanced commands provided by Flash manufacturers exploit different levels of parallelism inside SSDs, where they can either improve or degrade the SSD performance and endurance depending on how they are used; 2) Different physical-page allocation schemes employ different advanced commands and exploit different levels of parallelism inside SSDs, giving rise to different performance and endurance impacts; 3) The priority order of using the four levels of parallelism has the most significant performance and endurance impact among the three internal factors. The optimal priority order of using the four levels of parallelism in SSDs is found to be: 1) the channel-level parallelism; 2) the die-level parallelism; 3) the plane-level parallelism; and 4) the chip-level parallelism.

Index Terms—NAND Flash-based SSD, advanced commands, allocation schemes, internal parallelism, performance, endurance

1 Introduction

During the last two decades, tremendous development and growth have happened in the NAND-Flash-based Solid State Drive (SSD). Various ingenious methods [1], [2], [3], [4], [5], [6] have been presented to decrease the price, increase the per-unit capacity, improve the reliability, and address the random-write performance penalty for enterprise quality Flash memory storage. SSDs have been widely employed in modern computing systems from low-end personal computers to high-end high-performance supercomputers. Thus, academia and industry pay a tremendous deal of attention to the performance and endurance issues of the solid-state storage system [7], [8], [9], [10], [11].

Such performance and endurance issues of SSDs, from Flash Translation Layer designs [12], [13], [14], [15] and buffer schemes [16], [17], [18], [19] to the characteristics of Flash or SSD [20], [21], [22], [23], have been extensively discussed in the literature. However, some SSD internal behaviors, including those of advanced commands, allocation schemes, and parallelism inside SSDs that have potentially significant impacts on the SSD performance and endurance have been ignored. While there are some studies [14], [17], [24], [25], [26], [27] touching on the performance and endurance impacts of some of these internal behaviors, they focused on one or two individual factors in isolation and ignored the interplay among these factors, to the best of our knowledge. Our study reveals that advanced commands and allocation schemes are closely interrelated with the multiple levels of parallelism inside SSDs, and judicious and coordinated exploitation of these factors has the most profound impact on performance and endurance of SSDs.

In this paper, we thoroughly examine advanced commands, allocation schemes, parallelism inside SSDs and their relationship and interplay that have been largely ignored in the literature. Our in-depth experimental study has helped us obtain the following important and useful insights into the design of high-performance SSDs.

Advanced commands. To improve the performance of SSDs through efficient read, write, and erase operations, manufacturers have provided three kinds of advanced commands, including *copyback*, *multiplane*, and *interleave*. The latter two utilize two levels of parallelism inside SSDs. The use of these three advanced commands must adhere to some strict restrictions. We find that the appropriate application of these commands is extremely important. For instance, using the copyback command blindly results in a significant increase in the erasure count and average response time, by as much as 17.8 times (17.8×) and 7.7 times (7.7×), respectively, under the MSN workload (See Section 4.1).

- *Y. Hu, D. Feng, and C. Ren are with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, #1037, Luoyu Road, Wuhan 430074, Hubei, China. E-mail: yanghu@foxmail.com, dfeng@hust.edu.cn, chaoren.hust@gmail.com.*
- *H. Jiang, L. Tian, and H. Luo are with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, 217 Schorr Center, 1101 T Street, Lincoln, NE 68588-0150. E-mail: {jiang, tian, hluo}@cse.unl.edu.*

Manuscript received 12 Aug. 2011; revised 8 Feb. 2012; accepted 21 Feb. 2012; published online 28 Feb. 2013.
Recommended for acceptance by D.K. Panda.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-08-0541.
Digital Object Identifier no. 10.1109/TC.2012.60.

0018-9340/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

SAMSUNG EX1014, Page 1 of 15

Allocation schemes. Dynamic allocation assigns physical pages dynamically, while static allocation is based on fixed striping. The latter is easy in implementation and can be very effective in some application environments, while the former is more flexible and adaptive in exploiting parallelism and, thus, conducive to better performance in most cases. The physical page assigned to a logical page can be calculated by a series of formulas in static allocation. Static allocation performs consistently the best in serving read requests under all workloads, while dynamic allocation is superior in write and overall performance and endurance under most workloads (See Section 4.2).

Priority order of SSD parallelism levels. Parallelism has been regarded as a key factor in achieving the peak SSD performance [24]. There are four levels of parallelism inside an SSD:

1. *channel-level*,
2. *chip-level*,
3. *die-level*, and
4. *plane-level* [47], [48].

Advanced commands utilize the last two levels of parallelism; allocation schemes can effectively utilize multiple levels of parallelism and determine the priority order of them. Our analysis and evaluation show that there tends to exist an optimal priority order of the four levels of parallelism. Improper priority orders can result in a performance degradation of up to 60 percent (See Section 4.3).

The rest of the paper is organized as follows: The background necessary for motivating this research is presented in Section 2. Section 3 introduces the evaluation platform and workloads. Then, we present our extensive trace-driven evaluations of the parallelism exploitation inside SSDs in Section 4. In Section 5, we present the design guideline and an example of high-performance SSDs. Related work is introduced in Section 6. Section 7 summarizes the key observations and insights obtained from our evaluations.

2 Background and Motivation

2.1 Flash Memory Basics

Generally speaking, there are two kinds of Flash memory: NOR and NAND [3]. NOR-Flash memory provides byte-level (8 bits) random access, thus it is typically used in read-only applications such as storing firmware codes. In contrast, NAND-Flash memory provides higher storage density, larger capacity, and lower cost than NOR-Flash memory, but only supports page-level (2, 4 KB, etc.) random access. Thus, it is typically used for more general-purpose applications such as storing user data. Throughout this paper, we will use the term "Flash" to refer to NAND-Flash memory specifically.

To increase storage density, Flash manufactories aggregate several Flash chips into a module called package [28], [29], [30], [31], [32], [33], [34]. All chips in a package share the same 8/16-bit-I/O bus but have separated chip enable (CE) and ready/busy (R/B) control signals. At present, major Flash packages use 8-bit wide I/O bus. Without special explanation, I/O bus is by default 8-bit wide in the following paper. Each chip is composed of multiple dies.

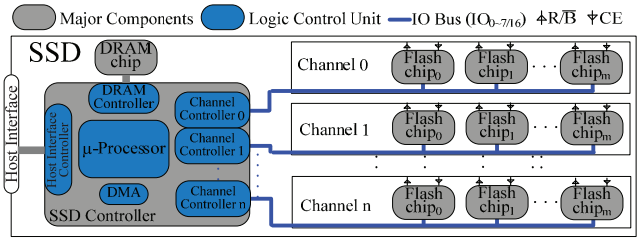


Fig. 1. The entire logical architecture of one general SSD controller.

Each die has one internal R/B signal invisible to users and valid only in advanced commands. Each die is composed of multiple planes. Each plane contains thousands of Flash blocks and one data register (Some products' plane contains an extra cache register). A flash block typically consists of 64 or 128 pages, where a page is further divided into many 512-byte subpages, which equals a typical sector size. Each subpage has 16-byte spare space used to store metadata. While chips and dies are often confused with each other in many previous studies in the literature, CE and R/B signals make them clearly distinct from each other. A chip is a basic service unit that has its independent CE and R/B signals. A die is a component of a chip, which has an internal R/B signal only.

There are two well-known and unique Flash physical characteristics, namely, write-after-erase and limited erase cycle. A write operation can only change the value of each target bit from "1" to "0." After being written a page must be erased (i.e., all the bits are reset to "1") before the next write operation. Each Flash block has an upper limit of erase cycles before it is worn out. After worn out, a block can no longer store any data. A typical erase-cycle limit is about 10K-100K [4].

2.2 SSD Basics

Flash memory SSD is an embedded system, which is mainly composed of storage component (Flash memory chips) and control component (embedded CPU), called an SSD controller, as shown in gray blocks of Fig. 1. Fig. 1 illustrates the traditional logical architecture of an SSD. SSD controller must provide connections between host interface and Flash channel interface. Generally speaking, for one SSD, there is only one host interface to connect with the host, such as PATA, SATA, SAS, PCI-E, and so on; in contrast, there are multiple Flash channel interfaces that can connect with multiple independent channels, inside an SSD controller, for increasing SSD's capacity and improving its I/O performance [24].

Based on the different sharing modes of Flash chip control signals and data signals [24], there are multiple ways to implement Flash memory channel interface. Because we focus on the design of high-performance SSD, we only consider one special implementation of channel, which provides the best performance, called independent channel as shown in Fig. 2.

The SSD controllers provide two kinds of on-chip resources, including logic cells and user I/O pins. Except for the logic cells used for implementing host interface, DMA, buffer interface and other system components, building Flash channel controllers will consume the rest of

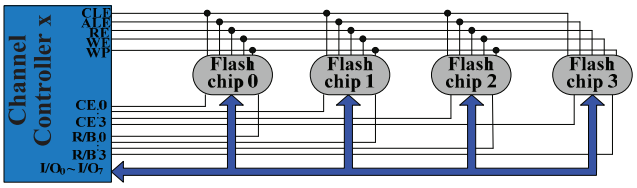


Fig. 2. The connection of an independent channel.

TABLE 1
The Signals Provided by an Independent SSD Channel Controller (N Indicates the Number of Signals; x Indicates the Number of Chips in a Channel)

Name	Function	N	Name	Function	N
I/O	Data input/output	8	CE	Chip Enable	x
R/B	Read/Busy state	x	RE	Read Enable	1
CLE	Command Latch Enable	1	WE	Write Enable	1
ALE	Address Latch Enable	1	WP	Write Protect	1

logic cells. A Flash channel controller consumes not only logic cell resource but also user I/O pin resource. We list all signals needed by a Flash channel in Table 1. If we employ four chips in a channel, each channel will consume 21 user I/O pins. The independent Flash channel number is restricted by the limited logic cell and user I/O pin resources.

2.3 Parallelism Inside SSDs

There are four levels of parallelism inside SSDs:

1. *channel level* (among channels);
2. *chip level* (among chips in a channel);
3. *die level* (among dies in a chip); and
4. *plane level* (among planes in a die).

For instance, as shown in Fig. 3, if a request is serviced 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 die 0 and 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.

In fact, except for the channel-level parallelism, other three levels of parallelism can improve the utilization of an independent channel. Meanwhile, limited by the technical constraints of current asynchronous Flash I/O bus, the maximum bandwidth is 40 MB/sec [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [37]. Therefore, by utilizing chip-level, die-level, and plane-level parallelism [8], [24] it is possible to achieve this maximum bandwidth.

2.4 Basic Command and Advanced Command

There are three common basic commands in Flash: read, program (write), and erase. A read command fetches data from a target page. A write command writes data to a target page. An erase command resets all bits of a target block to "1". Each read/write command operation consists of two steps: 1) data transfer; and 2) reading/writing data from/to the target page to/from the plane data register. All commands are initiated by writing the command code to

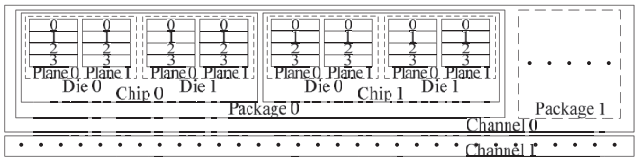


Fig. 3. An SSD internals.

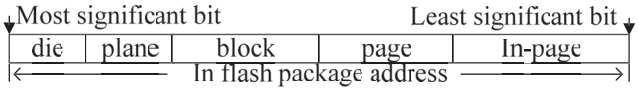


Fig. 4. A format of Flash chip address.

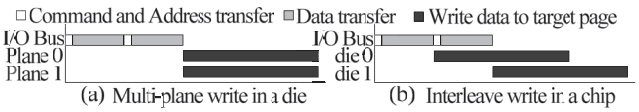


Fig. 5. The multiplane write and interleave write command process.

the command register and the address of the request to the address register. The address refers to the location of the target data of the request inside the chip. A Flash chip address is separated into five segments: die, plane, block, page, and in-page, as illustrated in Fig. 4. Within a block, the pages must be written consecutively in the ascending order of page address. Random-page-address writing is prohibited. We call this restriction R_a.

Most Flash manufacturers provide advanced commands, such as *copyback*, *multiplane*, 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], [29], [30], [31], [32], [33], [34], [37].

2.4.1 Copyback (Internal Data Move)

This command moves data from one page to another in the *same* plane without occupying the I/O bus. The command saves twice data transfer time. Some manufacturers also call it "internal data move" [38], [39]. We will call it copyback in the rest of the paper. The source page and the target page must have the same die and plane addresses, and their addresses must be both odd or even. As shown in Fig. 3, a copyback command operation can only move data from page 0 to page 2, or from page 1 to page 3, and so on. Moving data from page 0 to page 1 or page 3 are prohibited. We call this restriction R_b.

2.4.2 Multiplane

This command activates multiple read, program or erase operations in all planes of the same die. The command utilizes the plane-level parallelism. It only costs the time of multiple data transfer and one read, write or erase operation to complete multiple read, write or erase operations as illustrated in Fig. 5a. The pages executing a multiplane read/write operation must have the same die, block, and page addresses. And the blocks executing a multiplane erase operation must have the same die and block addresses. As shown in Fig. 3, only page 1 from plane 0 and page 1 from plane 1 of the same die can be read/written simultaneously by using a multiplane read/write operation. Reading/writing page 1 from plane 0 and page 3 from plane 1 using a multiplane read/write operation is prohibited. We call this restriction R_c.

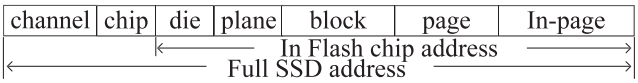


Fig. 6. The format of a full address of SSD.

2.4.3 Interleave

This command executes several read, write, erase, and multiplane read/write/erase operations in different dies of the same chip simultaneously. It utilizes the die-level parallelism. The interleave command is different from the interleave operation [17], [25], [26], [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 Fig. 5b.

2.5 Allocation Schemes

An allocation scheme determines how to choose free physical page to accommodate logical page to be written to the SSD. To locate a particular physical page, one must know the channel address and chip address in addition to the die address, plane address, block address, and page address as shown in Fig. 4. The format of a full address of SSD is shown in Fig. 6.

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

Static allocation first assigns a logical page to a *predetermined* channel, chip, die, and plane before allocating it to any free physical pages of the plane [14]. The channel, chip, die, and plane addresses assigned to each logical page are typically calculated by certain formulas that define a special allocation scheme, which we will discuss in Section 4.2.

Dynamic allocation assigns a logical page to any free physical page of an 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 and chips, the erasure count of blocks, the priority order of parallelism levels, and so on.

In this paper, we will evaluate and directly compare static and dynamic allocation schemes in terms of performance and wear leveling, and consider the utilization of multiple levels of parallelism.

2.6 Research Motivation

To design a high-performance and high-endurance SSD, appropriately utilizing multiple levels of parallelism is a basic and important factor. Utilizing these multiple levels of parallelism relates to several other factors, such as advanced commands, allocation schemes, and the priority order of using different parallelism levels inside SSD. The previous literature discuss part of these factors independently, thus we must answer the following research questions that have not been fully and comprehensively addressed, if at all, in the literature, to the best of our knowledge.

Motivation 1: Advanced commands make use of two kinds of parallelism: plane-level parallelism and die-level parallelism. When we utilize these two parallelism by employing advanced commands, what issues should be considered by us?

TABLE 2
Configuration Parameters Used in SSDsim

Parameters	Values
Page read to register (t_R)	20us
Page write from register (t_{PROG})	200us
Block erase (t_{ER})	1.5ms
Read one byte data from register (t_{RC})	25ns
Write one byte data to register (t_{WC})	25ns
Channel-Chip-Die-Plane-Block-Page	4-4-2-2-2048-64
Page size	2KB

(The Channel-Chip-Die-Plane-Block-Page parameter indicates the numbers of channels in 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 the default configuration parameters for the evaluation.)

Motivation 2: Taking advanced commands and multiple levels of parallelism into consideration, how can we design static allocation schemes and dynamic allocation schemes? What is the difference on performance and endurance when using these two allocation schemes? And, how to choose an appropriate allocation scheme?

Motivation 3: Given the four levels of parallelism inside an SSD, what is the optimal priority order to optimize the performance and endurance of an SSD?

To comprehensively answer these questions, we conduct a series of trace-driven experiments and evaluations detailed in Section 4.

3 Evaluation Platform

In this section, we will introduce the evaluation environment we use to conduct the in-depth investigation.

3.1 Evaluation Platform

We employ an event-driven, modularly designed and configurable SSD simulator, called SSDsim [47], as our evaluation platform.

The evaluation platform is assumed to be an SSD with multiple channels, multiple chips, multiple dies, and multiple planes. There are many ways to organize channels and chips based on the sharing methods of I/O bus, CE signals, and R/B signals. Since our research focuses on the SSD used in the high-performance computing environment, we will concentrate on independent channel organization, as shown in Fig. 2. The timing and organization characteristics of the configuration are based on a real NAND-Flash product [28], as summarized in Table 2.

3.2 Workloads

We use a set of real-world traces listed in Table 3 to study the performance and endurance impacts of advanced commands, allocation schemes, and multiple levels of parallelism inside SSDs. *Develop* [35] was obtained from a file server accessed by more than 3,000 users to download various daily builds of Microsoft Visual Studio. *Exchange* [35] was collected at the Microsoft Exchange 2007 SP1 server, which is a mail server for 5,000 corporate users. *Financial1* and *Financial2* were collected at a large financial institution [45]. *MSN* [35] was collected at the Microsoft's

SAMSUNG EX1014, Page 4 of 15

TABLE 3
Characteristics of the Workloads

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

several live file servers. *Radius* [35] was obtained from a RADIUS authentication server that is responsible for worldwide corporate remote access and wireless authentication. *Websearch* was collected at a popular Internet web search machine [45].

4 Experimental Evaluations

In this section, we evaluate two SSD internal behaviors about multiple levels of parallelism inside SSDs, which have a notable impact on SSD performance and endurance, namely, 1) advanced commands in Section 4.1; and 2) allocation schemes in Section 4.2. This is followed by a study on the priority order of parallelism levels inside SSD, presented in Section 4.3.

4.1 Advanced Commands

In this section, we evaluate the performance impact of advanced commands provided by Flash manufacturers, and how restrictions R_a, R_b, and R_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. We only present the results of the dynamic scheme, since static allocation is shown to have the same performance and endurance trend as dynamic allocation.

4.1.1 Copyback

When using the copyback command, restrictions R_a and R_b must be adhered to. Fig. 7 illustrates the process of executing a copyback command in a plane, where the data stored in $PPN = 26$ needs to be migrated to a free physical page. This data migration is triggered by a garbage collection operation. Since the pages in a block must be programmed sequentially (R_a), the next available page is $PPN = 1{,}217$. However, R_b forbids writing the data to $PPN = 1{,}217$, forcing the invalidation (or waste) of $PPN = 1{,}217$ and migration of the data into $PPN = 1{,}218$. We can use the command *blindly* or *wisely*. Using the copyback command *blindly* means that the command will be used in all cases, whether or not the free page will be wasted. Fig. 7 illustrates the case of using copyback blindly; In contrast,

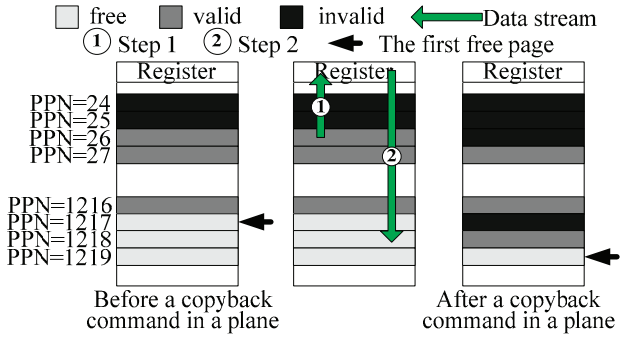


Fig. 7. An example of the copyback command.

use the command *wisely* means that the command will only be used when the free pages will not be wasted.

The performance impact of how the copyback 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 left Y-axis of Fig. 8a. The erasure count of using the copyback command blindly, normalized to that of only using basic commands, is shown by small triangles and labeled on the right Y-axis of the figure. Fig. 8b plots the cumulative distribution function (CDF) of the response time (Due to the space limit, we only list the CDF of MSN workload). The inserted figure is an amplified view of the inflexion point of CDF.

From Fig. 8a, it is clear that using the copyback command blindly has a notable negative impact on the average response time and the erasure count under the Dev, MSN, and Ex workloads. This is because using the copyback commands blindly, compared with only using the basic commands, leads to 1.49, 2.87, 0.17 times more pages being invalidated under these workloads, respectively, which trigger more frequent garbage collections and further decrease the overall performance and increase erasure count. On the contrary, using the copyback command wisely does improve performance without increasing the erasure count. This is because there are no extra invalidated pages and no extra erasure operation induced. Moreover, we can find in Fig. 8a that using the copyback command blindly leads to a large number of erase operations, which results in

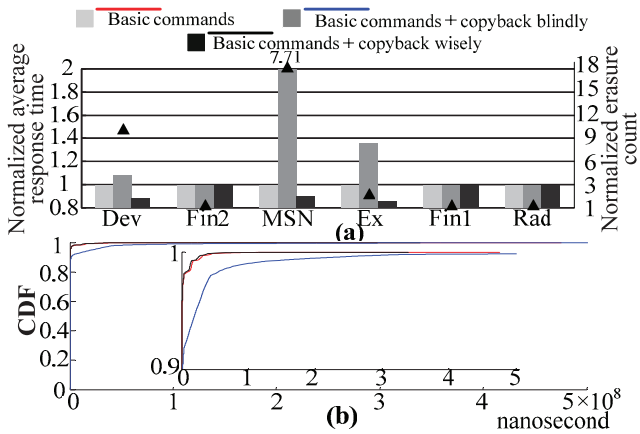


Fig. 8. (a) Performance and endurance impacts of the two methods of using copyback command; (b) CDF of the response time.

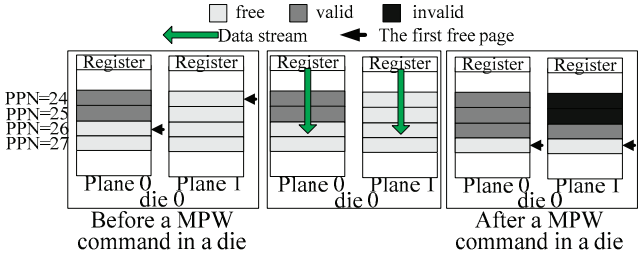


Fig. 9. An example of using the MPW command.

significant response-time latency of about 10 percent requests shown as the blue line of Fig. 8b. While using the command wisely saves data transfer time without increasing erase operation, therefore it slightly better than only using the basic commands shown as the red and black lines.

Insight 1. The copyback command should be used wisely only when the addresses of the source page and the target page have the same parity. Otherwise, the I/O performance and endurance of SSDs can be significantly reduced.

4.1.2 Multiplane

In this section, we analyze the multiplane command for reads and writes, which we call MPW (Multiplane Write) and MPR (Multiplane Read) for short in the remainder of the paper.

Multiplane write. As mentioned in Section 2.4, a multiplane command can execute the same basic commands in all planes within the same die. Therefore, it exploits the plane-level parallelism. When using an MPW command, restrictions R_a and R_c must be adhered to.

Fig. 9 illustrates an MPW command in 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,$\mathrm{PPN} = 24$ and $\mathrm{PPN} = 25$ in plane 1 will be invalidated. Therefore, in this case, executing the MPW command invalidates (and wastes) two free pages.

Similar to the copyback command, the MPW command can be used blindly or wisely. Fig. 9 shows the case of using the MPW command blindly. In Fig. 10a, we plot the average response times and erasure counts of the MPW command in the same way as in Fig. 8a. Fig. 10b plots CDF of the response time under the MSN workload.

As shown in Figs. 10a and 10b, using MPW blindly improves average response time compared with the basic commands and using MPW wisely. However, a large number of free pages are invalidated (e.g., 0.72, 1.04, 0.58 times more pages are being wasted under the Dev, MSN, and Ex workloads compared with using the basic commands only), which leads to more extra erase operations. Note that the saved time of plane-level parallelism outweighs the wasted time of extra erase operations. Therefore, using MPW blindly can still improve response time under all workloads. On the other hand, since the conditions required by 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.

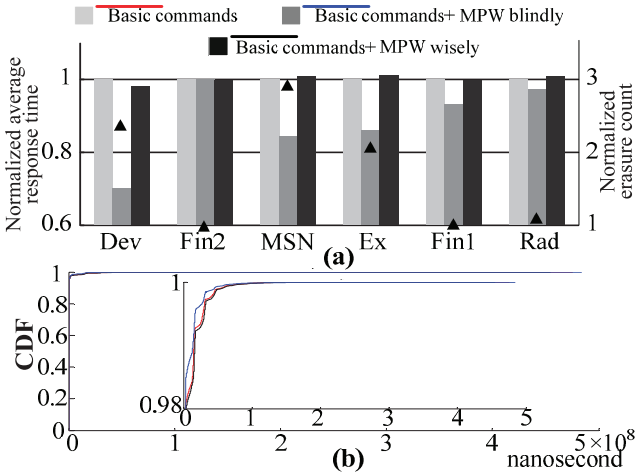


Fig. 10. (a) Performance and endurance impacts of the two methods of using the MPW command; (b) CDF of the response time.

Insight 2. 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 conditions required by its application can rarely be met.

Multiplane Read. MPR performs multiple-page read operations in different planes of the same die simultaneously. When using MPR, restriction R_c must be adhered to. In Fig. 11, we show the average response times of MPR under the dynamic and static allocation schemes, respectively, normalized to the response time of using the basic commands only. We find that the performance gains are negligible under a majority of the workloads. An improvement of only 15 percent is observed for a two-page MPR command. Moreover, the fact that Ex and Fin1 are write-dominant workloads causes the performance gains by MPR to be negligible. Since the request size of Fin2 is too small to be striped into multiple pages, MPR is not applicable there. Under Web, Dev, and MSN, the performance gains are relatively higher, since these three workloads are read dominant, whose request sizes are multiples of a Flash page size.

Insight 3. MPR cannot provide significant performance improvement under most workloads. But in the application environments with read-dominant workloads of large request size (i.e., Web, Dev, and MSN), using MPR can help improve I/O performance.

In addition to MPW and MPR, the multiplane 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

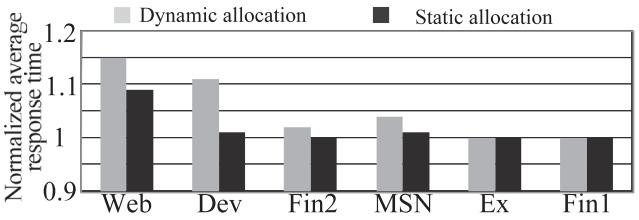


Fig. 11. Performance gain of MPR over the basic commands under the dynamic and static allocation schemes, respectively.

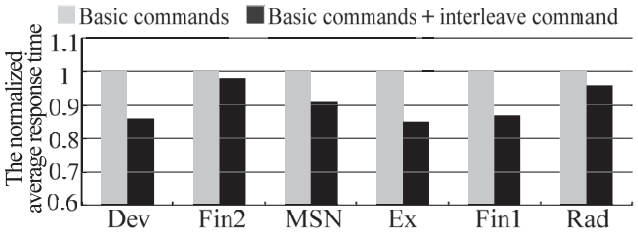


Fig. 12. Performance impact of the interleave command.

collection algorithm and wear-leveling algorithm used, which are beyond the scope of this paper, we will not independently evaluate the performance impact of using multiplane erase command.

4.1.3 Interleave

The interleave command exploits the parallelism among dies in the same chip. Pages and blocks from different dies of 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 R_a must be adhered to. Therefore, unlike the other advanced commands, there is no endurance loss when using the interleave command.

We plot the performance gains due to the interleave read/write/erase command as a function of the workloads in Fig. 12, measured in the average response time normalized to that based on the basic commands. Fig. 12 shows that the I/O performance is improved.

Insight 4. 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 for the case of the multiplane erase command at the end of Section 4.1.2, 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 later.

4.1.4 The Combined Use of the Three Advanced Commands

In this section, we employ the three advanced commands simultaneously and evaluate their combined impacts on the performance and endurance of SSDs. Based on Insights 1-4, there are two recommended approaches to using the advanced commands, namely, 1) use the copyback command wisely, the MPW command blindly, and the interleave command ubiquitously; and 2) use the copyback command, the MPW command wisely, and the interleave command ubiquitously.

In Fig. 13a, we plot the average response time and erasure count of using the advanced commands in the two recommended approaches as a function of workloads, in the same way as Figs. 8a and 10a, and Fig. 13b plots CDF of the response time under the MSN workload.

From these figures, we find that the combined use of the advanced commands based on Approach 1 achieves the best performance but leads to some SSD endurance degradation, while Approach 2 achieves less performance gains but without any endurance loss.

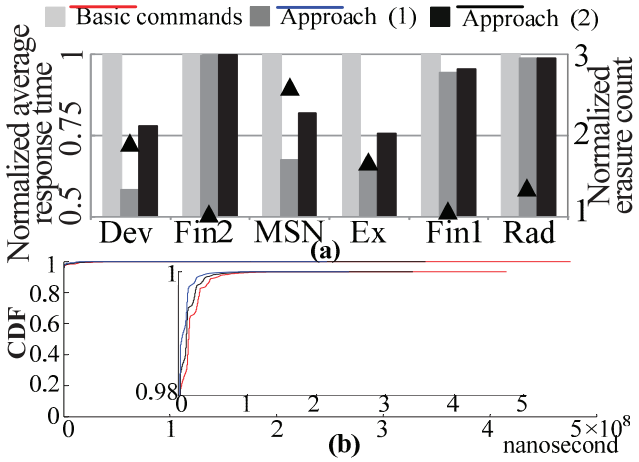


Fig. 13. (a) Performance and endurance impacts of the two recommended approaches to the combined use of the advanced commands; (b) CDF of the response time.

4.2 Allocation Schemes

In this section, we compare the performances of the static allocation and dynamic allocation schemes when exploiting different levels of parallelism inside SSDs, and evaluate the related wear-leveling issues.

Fig. 14 illustrates six different static allocation schemes, referred to as S1, S2, S3, S4, S5, and S6. As we discussed in Section 2.5, the channel, chip, die, and plane addresses of each logical page in each special static allocation scheme can be calculated by certain formulas. In what follows, we present these formulas for each static allocation scheme. The variables and operators that will be used in the formulas are listed in Table 4.

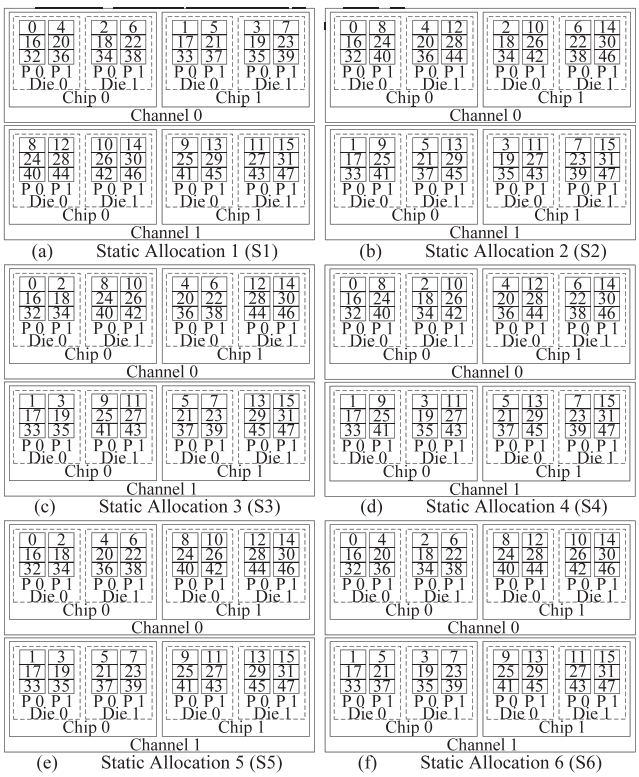


Fig. 14. The six typical static allocation schemes.

TABLE 4
The Variables and Operators Used in the Formulas of Assigning Physical Addresses of Logical Pages in Static Allocation Schemes

V/O	Explanation
cal_n	The number of channels in the entire SSD
chp_n	The number of chips in a channel
die_n	The number of dies in a chip
pln_n	The number of planes in a die
/	Division and round down
%	Modulo arithmetic
×	Multiplication

S1: As shown in Fig. 14a, the priority order of parallelism levels in this scheme is:

1. the chip-level parallelism;
2. the die-level parallelism;
3. the plane-level parallelism; and
4. the channel-level parallelism.

The formulas for S1 are:

Formula S1

Channel= [lpn/(pln_n×die_n×chp_n)]%cal_n
Chip= lpn%chp_n
Die= (lpn/chp_n)%die_n
Plane= [lpn/(die_n×chp_n)]%pln_n

S2: As shown in Fig. 14b, the priority order of parallelism levels in this scheme is:

1. the channel-level parallelism;
2. the chip-level parallelism;
3. the die-level parallelism; and
4. the plane-level parallelism.

The formulas for S2 are:

Formula S2

Channel= lpn%cal_n
Chip= (lpn/cal_n)%chp_n
Die= [lpn/(chp_n×cal_n)]%die_n
Plane= [lpn/(die_n×chp_n×cal_n)]%pln_n

S3: As shown in Fig. 14c, the priority order of parallelism levels in this scheme is:

1. the channel-level parallelism;
2. the plane-level parallelism;
3. the chip-level parallelism; and
4. the die-level parallelism.

The formulas for S3 are:

Formula S3

Channel= lpn%cal_n
Chip= [lpn/(pln_n×cal_n)]%chp_n
Die= [lpn/(pln_n×chp_n×cal_n)]%die_n
Plane= (lpn/cal_n)%pln_n

S4: As shown in Fig. 14d, the priority order of parallelism levels in this scheme is:

1. the channel-level parallelism;
2. the die-level parallelism;
3. the chip-level parallelism; and
4. the plane-level parallelism.

The formulas for S4 are:

Formula S4

Channel= lpn%cal_n
Chip= [lpn/(die_n×cal_n)]%chp_n
Die= (lpn/cal_n)%die_n
Plane= [lpn/(die_n×chp_n×cal_n)]%pln_n

S5: As shown in Fig. 14e, the priority order of parallelism levels in this scheme is:

1. the channel-level parallelism;
2. the plane-level parallelism;
3. the die-level parallelism; and
4. the chip-level parallelism.

The formulas for S5 are:

Formula S5

Channel= lpn%cal_n
Chip= [lpn/(pln_n×die_n×cal_n)]%chp_n
Die= [lpn/(pln_n×cal_n)]%die_n
Plane= (lpn/cal_n)%pln_n

S6: As shown in Fig. 14f, the priority order of parallelism levels in this scheme is:

1. the channel-level parallelism;
2. the die-level parallelism;
3. the plane-level parallelism; and
4. the chip-level parallelism.

The formulas for S6 are:

Formula S6

Channel= lpn%cal_n
Chip= [lpn/(pln_n×die_n×cal_n)]%chp_n
Die= (lpn/cal_n)%die_n
Plane= [lpn/(die_n×cal_n)]%pln_n

Dynamic allocation used in this section assigns a logical page on any idle chip of any idle channel of the entire SSD. The optimal order of parallelism levels used in this dynamic allocation scheme will be discussed in Section 4.3. All the data to read are stored evenly among all the chips of all the channels in the entire SSD.

In this section, we conduct two sets of experiments. First, we compare the average response time of dynamic allocation with that of static allocation when only two of the four levels of parallelism, channel-level parallelism, and chip-level parallelism, are exploited. In other words, we only employ basic commands and do not employ advanced commands. In this case, the representative static allocation scheme is S2, because S2 shows the best performance when only the channel-level parallelism and chip-level parallelism are exploited. Second, we compare the average response time of dynamic allocation with that of static allocation when all four levels of parallelism are exploited. That is, we employ basic commands and advanced commands simultaneously. In this case, the representative static allocation scheme is S6, because the priority order of parallelism levels used in S6 is the optimal order, which will be discussed in Section 4.3.

The read/write/overall performance impacts of different allocation schemes in the condition whether or not to exploit die-level and plane-level parallelism are plotted in Fig. 15, under six different workloads.

SAMSUNG EX1014, Page 8 of 15

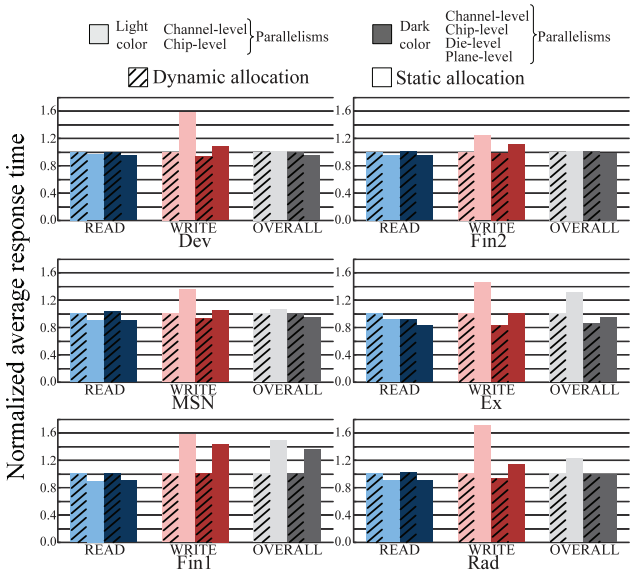


Fig. 15. Performance impact of different allocation schemes, measured in the average response time normalized to that of dynamic allocation with only the channel-level and chip-level parallelism.

4.2.1 Read Performance

From Fig. 15, we can find that the static allocation scheme outperforms the dynamic allocation scheme for read requests under all workloads. For a given read request, whose size is a multiple of one logical page size, the striping nature of static allocation is likely to distribute the sequential logical pages of the request to different channels and chips, which exposes and exploits the multichannel parallelism inside SSDs, thus reducing the response time of this request. In dynamic allocation, on the contrary, it is entirely possible that the sequential logical pages are stored in the same channel. For example, it leads to the increase of 1.5, 163.7, 95.5, 8.5, 365.1, and 0.2 percent more data stored in the same channel requested with dynamic allocation under Dev, Fin2, MSN, EX, Fin1, and RAD, respectively, compared with static allocation, so that these sequential logical pages will be operated in the same channel sequentially, failing to exploit the channel-level parallelism of SSD.

Insight 5. 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.

4.2.2 Write Performance

Fig. 15 shows that the dynamic allocation scheme outperforms the static allocation scheme for write requests under all workloads. The former is also superior to the latter in the overall performance under the Ex, Fin1, and Rad workloads. This is because the sequential logical pages of a multipage write request are likely to be serviced by multiple chips in several channels in the static allocation scheme, while the response time of the request is determined by the last completed logical page. If any one of the logical pages happens to be on a busy chip, the response time of the request can be severely delayed, because there are more garbage collection and erase operations on a busy chip. This, however, does not happen in dynamic allocation, because write requests can be adaptively distributed to idle chips. Ex, Fin1, and Rad are write-dominant workloads; therefore, the overall performance of dynamic allocation is better than that of static allocation under these three workloads.

Insight 6. The dynamic allocation scheme consistently outperforms the static allocation scheme in serving write requests. Thus, in the application environments that demand fast writes, or are write dominant in their workloads, the dynamic allocation scheme should be employed.

4.2.3 Relationship between Allocation Schemes and Advanced Commands

In Fig. 15, we observe another interesting phenomenon. That is, when dynamic allocation is employed, with the exception of the Ex workload, there is very little difference in performance improvement between exploiting all four levels of parallelism and exploiting only channel-level and chip-level parallelism. However, when static allocation is employed, this difference becomes significant, with the four-level parallelism case clearly outperforming the two-level parallelism case. This is because, when static allocation is used, all logical pages are assigned to the fixed chips, dies, and planes, making it very likely for these pages to be located in the same chip. Therefore, exploiting die-level and plane-level parallelism can notably improve performance. On the other hand, when dynamic allocation is used and with the absence of the die-level and plane-level parallelism, logical pages can be assigned to multiple idle chips as much as possible. In this case, exploiting the chip-level parallelism can partially offset the lack of the die-level and plane-level parallelism. Therefore, the performance improvement of adding the die-level and plane-level parallelism is negligible.

For the Ex workload, when dynamic allocation is employed, exploiting all four levels of parallelism brings about more than 20 percent overall performance improvement than exploiting only the channel-level and chip-level parallelism. The main reason lies in the fact that Ex is write-dominant workload and has the largest average write requests size among all the workloads used in our experiments, which makes the channel-level and chip-level parallelism alone inadequate in dealing with of the large write requests. Therefore, if dynamic allocation is employed, adding the die-level and plane-level parallelism can significantly improve the performance under the Ex workload.

Insight 7. In multiple-channel and multiple-chip SSDs, if the dynamic allocation scheme is employed, advanced commands may not be necessary, since the exploitation of the channel-level and chip-level parallelism alone can sufficiently deal with a majority of the requests under most workloads. More importantly, ignoring the implementation of advanced commands in this case can reduce the logic cell resource consumed in an SSD controller and lower the design complexity of the SSD controller and FTL [43]. On the other hand, if the static allocation scheme is employed, all four levels of parallelism must be implemented, in other words, the implementation of advanced commands is a necessity.

TABLE 5
The Standard Deviations of Total Erasure Counts of Blocks in Each Plane when Employing Either the Static Allocation Scheme or the Dynamic Allocation Scheme (B Indicates that Only Basic Commands Are Used; $B+A$ Indicates that Basic Commands and Advanced Commands Are Used)

Workloads	Static allocation		Dynamic allocation	
	B	*B+A*	*B*	*B+A*
Dev	7.5	7.5	3.0	3.9
Fin2	39.5	39.5	2.1	4.2
MSN	85.5	85.5	5.5	7.5
Ex	267.0	267.0	2.1	4.6
Fin1	397.6	273.3	2.4	4.5
Rad	8.9	8.4	1.2	4.0

4.2.4 Weal-Leveling

Wear-leveling algorithms are used to distribute the erase operations evenly to the entire SSD for the purpose of enhancing the endurance of the device. To balance the 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 will have negative impacts on performance and endurance. In Table 5, we list the standard deviation of the total erasure counts of blocks in each plane for the static and dynamic allocation schemes under the six workloads, where a low standard deviation indicates a more evenly distributed erase operations. It is clear from the table, that dynamic allocation has a much better wear-leveling efficiency than the static allocation.

Insight 8. The dynamic allocation scheme consistently outperforms the static allocation scheme on the wear-leveling efficiency.

4.3 Priority Order of Parallelism Levels in SSD

To determine the priority order of the four levels of parallelism 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. One of the objectives of exploiting parallelism inside SSDs is to overlap or pipeline these two steps. The chip-level, die-level, and plane-level parallelism is exposed on the same channel, and share the same channel bus. As a result, these three levels of parallelism can only overlap or pipeline step 2 of read/write operation. In contrast, the channel-level parallelism overlaps not only step 2, but also step 1 of an operation. Therefore, the channel-level parallelism should be given the highest priority among the four levels of parallelism. On the one hand, the 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, the die-level parallelism and plane-level parallelism only involve in a single chip, thus making them a higher priority than the chip-level parallelism.

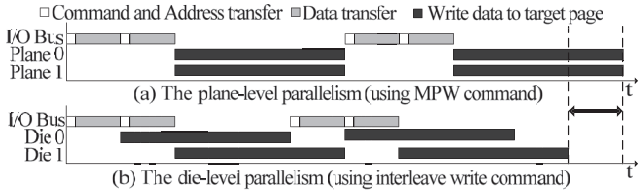


Fig. 16. The plane-level parallelism versus the die-level parallelism.

As shown in Fig. 16, 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 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 exploiting the die-level parallelism has no such disadvantages. Therefore, the die-level parallelism should be given a higher priority than plane-level parallelism.

4.3.1 Evaluation of the Priority Order of SSD Parallelism Levels under Dynamic Allocation

In this section, we use six different configurations of SSDs to conduct a set of experiments to evaluate the priority order of SSD parallelism. The configuration parameters of the six SSDs are listed in Table 6.

The hardware organizations of SSD1 and SSD2 (see Table 6) are different from those of SSD3, SSD4, SSD5, and SSD6, thus we compare their performance in two separated subfigures (Figs. 17a and 17b). Fig. 17a shows that SSD2 outperforms SSD1 consistently. In SSD2, we distribute requests to different channels. When eight 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 of the Flash media (i.e., step 2). This explains SSD2's superiority to SSD1 and confirms quantitatively that the channel-level parallelism should be given the first priority.

Fig. 17b shows that SSD4 consistently outperforms SSD3. While SSD3 prefers the chip-level parallelism to the die-level parallelism, SSD4 does exactly the opposite. When a request involving two pages is served by SSD3, two chips become busy. In contrast, only one chip becomes busy in SSD4,

TABLE 6
Six Configurations of SSDs

SSD	Cl.-Cp.-D.-P.	A	Page	Priority order
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>plane>die>chip
SSD6	1-4-2-2	Yes	2KB	channel>die>plane>chip

($X>Y$ in the "Priority" field signifies that choosing a free page from X is preferred to choosing one from Y. Cl.-Cp.-D.-P. indicates the numbers of channels in an SSD, chips in a channel, dies in a chip, and planes in a die, respectively. The "A" column indicates whether advanced commands are used (Yes) or not (No)).

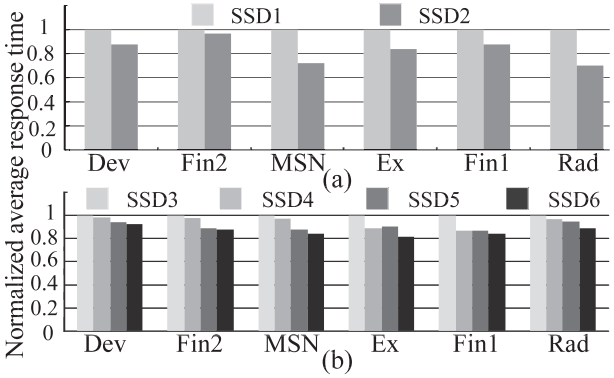


Fig. 17. Normalized average response time of SSD1, SSD2, SSD3, SSD4, SSD5, and SSD6.

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.

SSD6 outperforms SSD5 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 free pages being invalidated, as discussed in Section 4.1.2, more erase operations are triggered in SSD5 than in SSD6. Therefore, the die-level parallelism must be given a higher priority than the plane-level parallelism.

SSD6 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. Nonetheless, for the same request, only one chip is rendered busy in SSD6 with the execution of a single interleave multiplane read/write operation. Therefore, the priority of the chip-level parallelism should be the lowest.

4.3.2 Evaluation of Priority Order under Static Allocation

In the experiments of this section, we use six different cases of SSDs, SSD-S1, SSD-S2, SSD-S3, SSD-S4, SSD-S5, and SSD-S6, which employ six different static allocation schemes, S1, S2, S3, S4, S5, and S6, as shown in Fig. 14. These six SSDs share the following common configuration parameters: Eight channels in the SSDs, four chips in each channel, two dies on each chip, two planes on each die, 2,048 blocks on each planes, and each block contains 64 2 KB-pages. All advanced commands are used. In addition to the six real-world workloads listed in Table 3, we use a set of synthetic workloads in our experiments, whose key characteristics are listed in Table 7.

The performance comparisons of SSD-S1, SSD-S2, SSD-S3, SSD-S4, SSD-S5, and SSD-S6 are shown in Fig. 18. We find that SSD-S1 performs the worst among all the SSDs. It further confirms that the channel-level parallelism should

TABLE 7
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%)

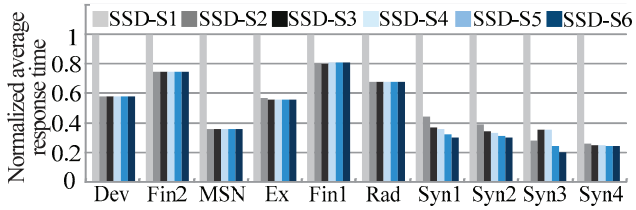


Fig. 18. Normalized average response time of SSD-S1, SSD-S2, SSD-S3, SSD-S4, SSD-S5, and SSD-S6.

be given the highest priority. With the exception of SSD-S1, all SSDs perform almost equally 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 levels inferred at the beginning of Section 4.3.

Insight 9. The optimal priority order of parallelism 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 Design Guidelines

In this section, we discuss the design parameters of high-performance SSDs based on the insights mentioned above, and then followed by a design example.

5.1 Design Parameters

As mentioned in Section 2.3, exploiting parallelism can improve the utilization of channels. We derive the formulas for the channel utilization of exploiting different levels of parallelism, by illustrating the exposure of different levels of parallelism in Fig. 19. For example, assuming that we employ a 2 KB-page Flash whose time parameters are listed in Table 2, the channel utilization of using read command without parallelism is 72.5 percent, as shown in Fig. 19a; the channel utilization of using interleave two-plane write command (i.e., exploiting the plane-level parallelism and die-level parallelism) is 69.1 percent, as shown in Fig. 19e. When the chip-level parallelism is exploited with the basic read operation, and there are two chips in a channel, the utilization of this I/O bus is 100 percent, as shown in Fig. 19g; when the chip-level parallelism is exploited with the basic write operation, and there are five chips in a Flash channel, the utilization of this I/O bus is 100 percent, as shown in Fig. 19h.

Generally speaking, to design a high-performance SSD, a clear performance goal should be determined a priori. For example, to support a special application with a required read-operation bandwidth of 250 MB, and write-operation bandwidth must reach 240 MB. How to choose an appropriate hardware organization and SSD controllers to achieve the performance goal is a design issue that must be addressed.

Let us assume that the target read/write bandwidth is TRB/TWB, the utilization of the I/O bus of a channel as U,

Fig. 19. The utilizations of exploiting different levels of parallelism.

and other variables are listed in Table 8. Figs. 19a, 19b, 19c, 19d, and 19e indicate the utilizations with only one chip in a channel when exploiting different levels of parallelism. Fig. 19f indicates the utilization when using two chips in a channel and exploiting the die-level and the chip-level parallelism. To fill up the I/O bus of a channel, we should add chips in the bus where commands are used. We obtain the minimal number of chips to fill up the I/O bus of the channel by

$$min_cp = \left\lceil \frac{1}{\mathrm{U}} \right\rceil. \tag{1}$$

TABLE 8
The Variables Used to Design an Appropriate SSD Controller

Abb.	Definition
RB	The read bandwidth in a channel
WB	The write bandwidth in a channel
TRB	The target read bandwidth of SSD
TWB	The target write bandwidth of SSD
U	The utilization ratio of I/O bus of channel
min_cp	The minimum chip count to fill a channel
lc_ch	Logic cell resource consumed by a channel
pin_ch	I/O pin resource consumed by a channel
channel	The minimum number of channels to reach the performance goal
lc_f_ch	Logic cells used by channel controllers
pin_f_ch	I/O pins used by channel controllers

We obtain the read/write bandwidth of the I/O bus of a Flash channel by

$$\mathrm{RB} = \frac{1}{\mathrm{t_{RC}}}\ ;\ \mathrm{WB}\frac{1}{\mathrm{t_{WC}}}. \tag{2}$$

To achieve the target read/write bandwidth (TRB, TWB) by exploiting the channel-level parallelism, the number of independent channels has a lower limit, which can be calculated by

$$\mathrm{channel} = \mathrm{MAX}\left[\frac{\mathrm{TRB}}{\mathrm{RB}}, \frac{\mathrm{TWB}}{\mathrm{WB}}\right]. \tag{3}$$

For each independent Flash channel controller, the number of logic cells, denoted by lc_ch, is fixed. To fill up the I/O bus of a Flash channel, we should employ min_cp chips in each independent channel. Based on the statistical information of Table 1, (4) provides the number of I/O pins consumed by one independent channel

$$pin_ch = 13 + 2 \times min_cp. \tag{4}$$

Based on (1), (2), (3), and (4), we obtain the minimum numbers of the I/O pins and logic cells used by Flash channel controllers, as shown in (5) and (6), respectively,

$$lc_f_ch = \mathrm{lc_ch} \times \mathrm{channel}, \tag{5}$$

$$pin_f_ch = \mathrm{pin_ch} \times \mathrm{channel}. \tag{6}$$

As discussed above, an SSD controller must provide lc_f_ch logic cells and pin_f_ch I/O pins for its channel controllers to achieve the target bandwidth.

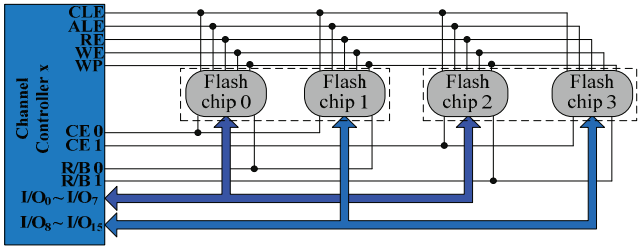


Fig. 20. The bit extension channel.

5.2 Design Example

We have designed a high-performance SSD, which employs a high-performance and cost-effective FPGA chip as the SSD controller. The resource provided by this includes approximately 600 user I/O pins and 128,000 logic cells. In addition to the basic commands, advanced commands, such as copyback and interleave, have been implemented in this SSD. The multiplane is abandoned in the implementation, based on Insight 2 (Section 4.1.2) indicating that only when we use the multiplane command blindly, we can gain an obvious enhancement of performance but at the cost of significantly reduced endurance (namely, SSD lifespan). Therefore, we give up this advanced command in our design. We will use the copy-back command wisely, and the interleave command ubiquitously.

Since implementing all other system components consumes a large number of logic cells, the remaining logic cells can only be used to implement four channel controllers. Although the logic cells are exhausted, dozens of user I/O pins are retained. In this case, the resource of logic cells is scarcer than the user I/O pin resource.

We can use the formulas derived in Section 5.1 to determine some key design parameters for this implementation. In our logic design of the Flash controller, t_{RC} and t_{WC} are 30 ns. From the formula of Fig. 19f, we know that the utilization of I/O bus of Flash channel is about 94 percent, assuming that the page size is 2 KB. According to (1), we must use at least two chips in a channel to fill up the channel. According to (4), we know that a channel will consume 17 user I/O pins; therefore 4 independent channels will consume 68 user I/O pins. Subtracting the user I/O pins consumed by other components, including host interface, buffer interface, and so on, there are about 80 I/O pins left.

Since 4 KB-page SSD achieves the best read/write performance in most application environments [47], we choose the page size of 4 KB in our design. There are two available methods to implement a 4 KB-page SSD. The first method is to use 4 KB-page Flash directly, with a channel width of 8 bits, as shown in Fig. 2. The other is to compact two 2 KB-page Flashes as one 4 KB-page Flash by bit extension. In the second method, the channel width is 16 bits, as shown in Fig. 20. Compared with the 8-bit-width channel, the 16-bit-wide channel doubles the bandwidth of read/write operations, and utilizes the remaining user I/O pins without increasing the consumption of logic cells.

6 Related Work

This paper focuses on exploring and exploiting the four-level parallelism inside SSDs. In this section, we briefly review the work most relevant to our study, concentrating on schemes exploiting parallelism inside SSDs.

Kang et al. [25] and Park et al. [27] exploit the channel-level parallelism. Seol et al. [17] make use of the chip-level parallelism, while Park et al. [26] employ the plane-level parallelism. Previous studies in the literature mainly focus on the channel-level and chip-level parallelism. In this paper, we evaluate the performance and endurance impact of exploiting all four levels of parallelism inside SSDs, the interplay of advanced commands, allocation schemes of SSD, and the priority order of four levels of parallelism.

Agrawal et al. [24] address the performance tradeoff issues from the hardware and software points of view. The authors have mainly studied the impact on performance of the chip-level and die-level parallelism. Dirik and Jacob [8] report that exploiting the channel-level and chip-level parallelism significantly improves SSD performance, and provides a detailed description about the internal commands and structures of SSDs.

Hu et al. [47] and Chen et al. [48] represent the state of the art in the literature about the parallelism inside SSDs. Hu et al. [47] discuss four key factors that have direct impact on SSD performance and endurance. One of the four key factors is the optimal priority order of parallelism. The authors have partially studied the parallelism's impact on performance and endurance. In contrast, this paper focuses exclusively on the issues of parallelism inside SSDs, with a more comprehensive and systematic approach to addressing this issue. Chen et al. [48] consider the four levels of parallelism inside SSDs. The authors present the research on the essential roles played by exploiting the internal parallelism of SSDs. They regard an SSD as a black box, implementing a set of experimental approaches to inferring indirectly the performance and endurance impact of parallelism inside SSDs. The conclusions reported by the authors can guide the application and system designers to better exploit the parallelism inside SSDs. This is a top-down methodology. On the contrary, we treat SSDs as a white box, observing the internal behaviors directly. The insights obtained from this paper can guide the design of high-performance SSDs. It is a bottom-up methodology. These two methods are orthogonal and mutually complementary.

7 Conclusion

There are four levels of parallelism inside SSDs, namely, channel-level parallelism, chip-level parallelism, die-level parallelism, and plane-level parallelism. In this paper, we first discussed the impact on performance and endurance of using advanced commands that have a direct relationship to the die-level and plane-level parallelism. Subsequently, we evaluated the impact on performance and endurance of employing different allocation schemes that have a strong connection to the exploitation of the four levels of parallelism. Finally, we studied the optimal priority order of exploiting parallelism inside SSDs. In these three parts, we conducted several sets of experiments and obtained the following key insights: 1) there are two recommended approaches to using the advanced commands, namely, use the copyback command, the MPW command, and the interleave command wisely as well as use the copyback command wisely, the MPW command blindly, and the interleave command ubiquitously; 2) Static allocation is found to perform the best on read performance under all

workloads. Dynamic allocation performs the best on write and overall performance and endurance under most workloads; 3) the optimal priority order of exploiting parallelism in SSD is: the channel-level parallelism → the die-level parallelism → the plane-level parallelism → the chip-level parallelism. Based on these insights, we discussed the design guidelines of high-performance SSDs and presented a design example.

Acknowledgments

The authors 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 No. 2011CB302301, NSFC No.61025008, 61232004, 60933002, 863 project No.2012AA012403, The National Key Technology R&D Program No.2011BAH04B02 US National Science Foundation (NSF) under Grants IIS-0916859, CCF-0937993, CNS-1016606, and CNS-1116609. Dan Feng is the corresponding author of this paper.

References

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

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

[3] M-System, "Two Technologies Compared: NOR vs NAND," In White Paper, http://maltiel-consulting.com/Nonvolatile_Memory_ NOR_vs_NAND.pdf, 2003.

[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, 2013.

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

[6] K.M. Greenan, D.D.E. Long, E.L. Miller, T. Schwarz, and A. Wildani, "Building Flexible, Fault-Tolerant Flash-Based Storage Systems," *Proc. Fifth Workshop Hot Topics in System Dependability (HotDep '09)*, June 2009.

[7] M. Moshayedi and P. Wilkison, "Enterprise SSDs," *ACM Queue*, vol. 6, pp. 32-39, July/Aug. 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," *Proc. 36th Ann. Int'l Symp. Computer Architecture (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," *Proc. ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC '10)*, Nov. 2010.

[10] A. Leventhal, "Flash Storage Today," *ACM Queue*, vol. 6, pp. 24-30, July/Aug. 2008.

[11] G. Graefe, "The Five-Minute Rule Twenty Years Later, and How Flash Memory Changes the Rules," *Proc. Third Int'l Workshop Data Management on New Hardware (DaMoN '07)*, June 2007.

[12] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A Flash Translation Layer Employing Demand-Based Selective of Page-Level Address Mapping," *Proc. ACM Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, Mar. 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," *Proc. IEEE 26th Symp. Mass Storage Systems and Technologies (MSST '10)*, May 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," *Proc. 23rd Int'l Conf. Supercomputing (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 Trans. 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," *Proc. Sixth USENIX Conf. File and Storage Technologies (FAST '08)*, Feb. 2008.

[17] J. Seol, H. Shim, J. Kim, and S. Maeng, "A Buffer Replacement Algorithm Exploiting Multi-Chip Parallelism in Solid State Disks," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES '09)*, Oct. 2009.

[18] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES '06)*, Oct. 2006.

[19] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee, "FAB: Flash-Aware Buffer Management Policy for Portable Media Players" *IEEE Trans. Consumer Electronics*, vol. 52, no. 2, pp. 485-493, May 2006.

[20] F. Chen, D.A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives," *Proc. 11th Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS/Performance '09)*, June 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," *Proc. IEEE/ACM 42nd Ann. Int'l Symp. Microarchitecture (MICRO '09)*, Dec. 2009.

[22] S. Boboila and P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis," *Proc. Eighth USENIX Conf. File and Storage Technologies (FAST '10)*, Feb. 2010.

[23] P. Desnoyers, "Empirical Evaluation of NAND Flash Memory Performance," *Proc. SOSP Workshop Hot Topics in Storage and File Systems (HotStorage '09)*, Oct. 2009.

[24] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proc. Usenix Ann. Technical Conf. (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," *J. Systems Architecture*, vol. 53, pp. 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*, vol. 9, no. 1, pp. 9-12, 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 Trans. Consumer Electronics*, vol. 55, no. 3, pp. 1392-1400, Aug. 2009.

[28] K9XXG08UXA datasheet. http://www.samsung.com/products/semiconductor/flash/technicallinfo/datasheets.htm, 2013.

[29] K9NCG08U5M datasheet. http://www.samsung.com/products/semiconductor/flash/technicallinfo/datasheets.htm, 2013.

[30] Micro MT29F16G08FAA NAND Flash Memory Datasheet. http://www.micron.com//document_download/?documentId=4308, 2013.

[31] Micro MT29F256G08CUCBB NAND Flash Memory Datasheet, http://www.micron.com//document_download/?documentId = 4368, 2013.

[32] Intel JS29F64G08CAMD1 MD332 NAND Flash Memory datasheet. http://www.intel.com/design, 2013.

[33] Toshiba TH58TVG7S2F NAND Flash Memory datasheet, http://www.toshiba.com/, 2013.

[34] Hynix H27UCG8U5(D)A Series 64Gb NAND Flash datasheet, http://www.hynix.com/datasheet/, 2013.

[35] Microsoft Enterprise Traces. http://iotta.snia.org/traces/list/BlockIO, 2013.

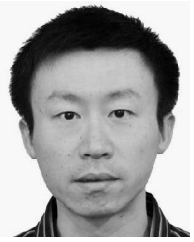
[36] Application Note for nand Flash Memory (revision 2.0) http://www.samsung.com/glo bal/business/semiconductor/products/flash/downloads/applicationnote/app_nand.pdf, 2013.

[37] "Open NAND Flash Interface Specification,"revision2.2. http://onfi.org/wp-content/uploads/2009/02/ONFI%202_2%20Gold.pdf, 2013.

[38] "NAND Flash Performance Improvement Using Internal Data Move,"Technical Note TN-29-15, http://download.micron.com/pdf/tech-notes/nand/tn2915.pdf, 2013.

[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, 2013.

[40] "SSD Extension for DiskSim Simulation Environment," http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4, 2013.
[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, Dept. of Computer Science and Eng., The Pennsylvania State Univ., PA, 2009.
[42] J. Bucy, J. Schindler, S.W. Schlosser, and G.R. Ganger, "The DiskSim Simulation Environment Version 4.0 Reference Manual," Mellon Univ., Parallel Data Lab Technical Report CMU-PDL-08-101, May 2008.
[43] E. Fal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138-163, June 2005.
[44] T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, "System Software for Flash Memory: A Survey," *Proc. Int'l Conf. Embedded and Ubiquitous Computing (EUC '06)*, pp. 394-404, 2006.
[45] "UMass Trace Repository," http://traces.cs.umass.edu, 2013.
[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, 2013.
[47] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity," *Proc. Int'l Conf. Supercomputing (ICS '11)*, May/June 2011.
[48] F. Chen, R. Lee, and X. Zhang, "Essential Roles of Exploiting Internal Parallelism of Flash Memory Based Solid State Drives in High-Speed Data Processing," *Proc. IEEE Seventh Int'l Conf. High Performance Computer Architecture (HPCA '11)*, Feb. 2011.
[49] C. Park, P. Talawar, D. Won, M. Jung, J. Im, S. Kim, and Y. Choi, "A High Performance Controller for NAND Flash-Based Solid State Disk," *Proc. IEEE Non-Volatile Semiconductor Memory Workshop (NVSMW '06)*, Feb. 2006.
[50] L. Bouganim, B. Jonsson, and P. Bonnet, "uFLIP: Understanding Flash IO Patterns," *Proc. Fourth Biennial Conf. Innovative Data Systems Research (CIDR '09)*, Jan. 2009.



Yang Hu is currently working toward the PhD degree in the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), Wuhan, China. His research interest includes nonvolatile memory-based storage system, and performance evaluation. He has more than five publications in journals and international conferences including ICS, MSST, and NAS.

Hong Jiang received the BSc degree in computer engineering in 1982, from Huazhong University of Science and Technology, Wuhan, China, the MASc degree in computer engineering in 1987, from the University of Toronto, Toronto, Canada, and the PhD degree in computer science in 1991 from the Texas A&M University, College Station, Texas. Since August 1991, he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, where he served as a vice chair of the Department of Computer Science and Engineering (CSE) from 2001 to 2007 and is a professor of CSE. At UNL, he has graduated 10 PhD students who upon their graduations either landed academic tenure-track positions (e.g., Stevens Institute of Technology, New Mexico Tech, University of Maine, University of Alabama, etc.) or were employed by major US IT corporations (e.g., Microsoft, Google, Seagate, Amazon, etc.). His present research interests include computer architecture, computer storage systems and parallel I/O, parallel/distributed computing, cluster and grid computing, performance evaluation, real-time systems, middleware, and distributed systems for distance education. He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 180 publications in major journals and international Conferences in these areas, including *IEEE-TPDS, IEEE-TC, ACM-TOS, JPDC*, ISCA, MICRO, FAST, USENIX ATC, ICDCS, IPDPS, OOPLAS, ECOOP, SC, ICS, HPDC, ICPP, and so on, and his research has been supported by US National Science Foundation (NSF), DOD, and the State of Nebraska. He is a senior member of the IEEE and a member of the ACM.

Dan Feng received the BE, ME, and PhD degrees in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 80 publications in journals and international conferences, including FAST, USENIX ATC, ICDCS, HPDC, SC, ICS and IPDPS. She is a member of the IEEE and a member of the ACM.

Lei Tian received the PhD degree in computer architecture from Huazhong University of Science and Technology, China, in 2010. He is currently a postdoc researcher at the University of Nebraska-Lincoln, and his research interests mainly lie in Flash memory SSD, data deduplication, cloud storage, large-scale metadata management, and RAID. He has more than 20 publications in premier journals and conferences including FAST, ICS, SC, HPDC, ICDCS, MSST, ICPP, IPDPS, CLUSTER, *IEEE TOC, IEEE TPDS*, and *ACM TOS*. He is a member of the IEEE.

Hao Luo received the BE degree in computer science and technology from Huazhong University of Science and Technology, Wuhan, China, in 2011, and is a first year PhD student in computer science and engineering in the University of Nebraska, Lincoln. His research interests include computer architecture and operating system.

Chao Ren received the BE degree from Huazhong University of Science and Technology (HUST) in June, 2012, and is currently a senior student of School of Computer Science and Technology, HUST, Wuhan. His research interest includes computer architecture and storage systems.