Parallel all the time: Plane Level Parallelism Exploration for High Performance SSDs

Congming Gao*†‡, Liang Shi*, Chun Jason Xue§, Cheng Ji§, Jun Yang†, and Youtao Zhang†
*East China Normal University, China; †University of Pittsburgh, USA
‡Chongqing University, China; §City University of Hong Kong, China
Email: {gaocm92, shi.liang.hk}@gmail.com,
jasonxue@cityu.edu.hk, chengji4-c@my.cityu.edu.hk,
juy9@pitt.edu, zhangyt@cs.pitt.edu

***Abstract*—Solid state drives (SSDs) are constructed with multiple level parallel organization, including channels, chips, dies and planes. Among these parallel levels, plane level parallelism, which is the last level parallelism of SSDs, has the most strict restrictions. Only the same type of operations which access the same address in different planes can be processed in parallel. In order to maximize the access performance, several previous works have been proposed to exploit the plane level parallelism for host accesses and internal operations of SSDs. However, our preliminary studies show that the plane level parallelism is far from well utilized and should be further improved. The reason is that the strict restrictions of plane level parallelism are hard to be satisfied. In this work, a *from plane to die* parallel optimization framework is proposed to exploit the plane level parallelism through smartly satisfying the strict restrictions *all the time*. In order to achieve the objective, there are at least two challenges. First, due to that host access patterns are always complex, receiving multiple same-type requests to different planes at the same time is uncommon. Second, there are many internal activities, such as garbage collection (GC), which may destroy the restrictions. In order to solve above challenges, two schemes are proposed in the SSD controller: First, a die level write construction scheme is designed to make sure there are always N pages of data written by each write operation. Second, in a further step, a die level GC scheme is proposed to activate GC in the unit of all planes in the same die. Combing the die level write and die level GC, write accesses from both host write operations and GC induced valid page movements can be processed in parallel at all time. As a result, the GC cost and average write latency can be significantly reduced. Experiment results show that the proposed framework is able to significantly improve the write performance without read performance impact.**

***Index Terms*—SSD, Parallelism, Storage, Performance Improvement**

I. Introduction

Solid state drives (SSDs) are widely adopted in modern computer systems, ranging from embedded systems, personal computers, to large servers in data centers. SSDs have many advantages, such as shock resistance, high random access performance, and low power consumption [1]. An SSD usually consists of multiple channels with each channel having multiple chips, each chip having multiple dies, and each die having multiple planes [2] [3]. To achieve high performance, the prior studies strive to exploit the parallelism at channel/chip/die/plane levels so that multiple accesses, such as reads, writes, and erases, can be processed in different parallel units simultaneously [4] [5] [6].

However, the parallelism at the last level, referred to as plane level parallelism, exhibits strict restrictions – for two operations that can be issued simultaneously to two different planes, they not only need to be of the same type (i.e., read or write) but also need to have the same in-plane address (i.e., the same offset within each plane), making it challenging to explore as shown in recent studies [7] [8] [9] [10] [11] [12]. For example, to concurrently write two planes, their write points need to aligned. Unfortunately, host sends uneven write requests to individual planes [9] while the activities originated from SSDs (e.g., garbage collection operations) further introduce asynchronicity [9] [13]. This leads to sub-optimal exploration of plane level parallelism and prevents modern SSDs from achieving further performance improvement.

In recent studies, Tavakkol *et al.* proposed *TwinBlk* to write data to the different planes in a die in a round-robin fashion [11]. *TwinBlk* faces two problems: (i) a single-page write operation can mis-align the write points of different planes; (ii) the write points may be misaligned by GC or WL (wear leveling) activities originated inside the SSD. In this case, most of accesses to the multiple planes cannot be processed in parallel. What's more, GC, when being initiated asynchronously in different planes, disables the plane level parallelism of related planes [9] [13] [14] [15]. To reduce GC-induced plane idleness, Shahidi *et al.* proposed *ParaGC* to activate the GC process at all planes in the same die at the same time [9]. However, it is only able to opportunistically use the plane level parallelism when all the pages at the same address of different planes are valid. For GC, *TwinBlk* selects blocks with same offset in different planes at the same time and activates GCs simultaneously. However, it cannot process all valid page movements in parallel when not all paired pages are valid. What all of these works did is to optimize plane level parallelism passively. None of them is able to satisfy the strict restrictions all the time. The key problem of previous works, such as TwinBlk and ParaGC, is that they cannot actively construct *multi-plane command* supported requests on all planes in the same die at all time, especially after GC is processed. Since the number of valid pages may be different for GC selected blocks, write points

of different planes will be moved to unaligned positions so that subsequent requests can not be processed in parallel. This problem also exist in superpage enabled SSDs [1] [16] [17]. Superpage is adopted to strip multiple requests to all planes in a die at each time so that more sequential write access is generated for improving write performance of SSDs. However, if there is a GC triggered on one of these planes, although GC induced valid page movements can be processed by writing superpage to all planes, the reclaimed free blocks will be unaligned, causing write points unaligned while these free blocks are allocated. That is, under superpage design, write points cannot be maintained all the time, too. Motivated by previous work, if we can construct the *multi-plane command* oriented writes from both host and GC at all time, the plane level parallelism can be maximally exploited.

In this paper, we propose SPD, an SSD *from plane to die* parallel optimization framework, to fully exploit the last level parallelism of SSDs for performance improvement by smartly satisfying the restrictions all the time. We summarize our contributions as follows.

- We propose SPD to treat all planes (e.g., N planes) in a die as a single unit so that a die write results in N page writes while a die read fetches N or fewer pages. Similarly, internal activities, e.g., GC, get triggered for N blocks from different planes that have the same in-plane block address. To our best knowledge, this is the first work on actively maintaining aligned write points for multiple planes in a die combining writes from both host and internal activities for all the time;
- We then propose die level write construction and die level GC schemes to fully exploit the plane level parallelism enabled by SPD. The write construction scheme is to construct write operation with N pages of data and issue them to a die at once; The die level GC scheme is to process valid page movements, aligning the write points of all planes in the same die.
- We evaluate the proposed SPD using a significantly extended SSDSim [10] and compare it to the state-of-the-arts. The experimental results show that SPD is able to significantly improve write performance of SSDs without read performance impact.

The rest of this paper is organized as follows: In Section II, the background is presented. In Section III, the problem statement is presented. In Section IV, the SPD framework is presented. In Sections V and VI, the experiment setup and evaluations are presented. In Section VII, related works are discussed. Finally, the work is concluded in Section VIII.

II. Background

In this section, we briefly discuss the background, including SSD organization, advanced SSD commands, parallelism, and garbage collection (GC).

A. SSD Organization

A modern SSD usually consists of multiple channels with each channel containing multiple flash chips. Within each flash chip, there are multiple dies with each die containing multiple planes. Figure 1 illustrates the organization of a typical SSD that has 4 channels, 2 chips per channel, 2 dies per chip, and 2 planes per die. The SSD parallelism can be exploited at channel/chip/die/plane levels, which have one major focus of previous studies for performance improvement [2] [13] [18]. To manage the flash memory as well as to explore the parallelism, an SSD controller comprises several components, including flash translation layer (FTL), data allocation (DA), wear leveling (WL), garbage collection (GC).

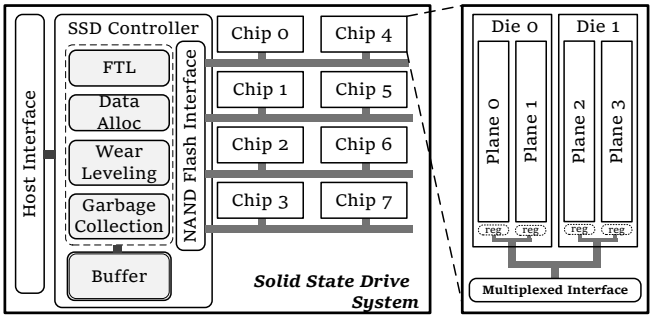


Fig. 1. The organization of SSDs.

The FTL is to manage the mapping between logical addresses and physical addresses. Based on the operation granularity, there are three types of mapping schemes, i.e., page mapping [4], block mapping [19], and hybrid mapping [20] [21] [22]. In this work, we assume the widely adopted page mapping as it tends to have its better performance.

The DA is to determine the allocations of channel, chip, die and plane for write operations. The pages within a plane are written sequentially, with the location of the next page to write indicated by a *write point* [2] [10] [23].

The WL is to distribute written data evenly to flash pages for prolonging the SSD lifetime [24] [25]. Since WL is not the focus, we do not discuss WL in the following sections — the proposed scheme works with widely adopted WL schemes in the literature.

Since flash memory cannot reprogram a programmed flash page before executing an erase operation to reclaim the whole block, modern SSDs widely adopt out-of-place-update scheme for data updating. To update a page, the corresponding updated data are programmed to a free flash page while the original flash page is set as invalid. When the number of free pages drops below a predefined threshold, the GC is activated to reclaim the invalid pages. GC first selects a victim block, e.g., the one with the most invalid pages; it then reads and programs the valid pages in the block to other blocks; and finally, it erases the whole victim block. Since page movements and erase operations are slow operations, GC has been identified as the most time-consuming activity in SSDs [9] [13] [14] [26]. In this work, we optimize GC by fully exploiting plane level parallelism.

In addition, modern SSDs widely equip a built-in Random Access Memory (RAM), referred to as the *SSD buffer*, within SSD controller for temporarily storing hot data and metadata. Since the access latency of RAM is much smaller than that of

flash memory, buffer-equipped-SSDs can provide much better performance for data hit in the buffer [27] [28] [29] [30].

B. Parallelism and Advanced Commands

The hierarchical SSD architecture provides four level parallelism, from channel, chip, die to plane. For channel and chip level parallelism, data can be processed in different chips in parallel. The parallelism of these two levels is naturally supported by SSDs while that of the rest two levels are supported by advanced commands [31] [18] [10] [9] [12]. The die and plane level parallelism is also referred to as internal parallelism [3].

For die level parallelism, operations issuing to the same chip but different dies can be processed in parallel with *interleaving command* [10] [9]. There is no restriction on when to use the interleaving command. For the last level parallelism, plane level parallelism may be exploited to further improve performance through processing operations concurrently on different planes of the same die. Due to circuit restrictions [7], as shown in the open NAND flash interface (ONFI) standard specification [8], the plane level parallelism can be exploited when satisfying the two operation type and in-plane address restrictions of *multi-plane command*. A *multi-plane command* improves plane utilization as it operates multiple planes within the same die in parallel and only takes the time to finish one operation. However, when the restrictions can not be met, it processes different planes sequentially to the requested operation. In particular, an operation processed on one plane blocks other planes of the same die from servicing other operations.

The number of planes per die can be 2 or 4 in most products, where 2 is the most popular design. If there are two planes within die, *multi-plane command* can be used when two operations accessing these two planes satisfy the restrictions. If there are four planes within a die, two different types of *multi-plane command* utilization are adopted in different SSDs. For the most popular one, *multi-plane command* is executed on either paired plane 0&1 or plane 2&3 [1] [10]; For the another one, all four planes are accessed in parallel only when four operations accessing planes satisfy the restrictions of *multi-plane command* [32].

Another advanced command, i.e., *copy-back command*, is designed to mitigate the inter-plane data movement cost [31] [1] [33]. With copy-back command, the register on a plane can temporarily store data from the current plane and write them back to other pages in the same plane [1] [33].

III. Problem Statement

In this section, we present the challenges in exploiting the plane level parallelism. Due to the restrictions of the *multi-plane command*, the plane level parallelism is hard to exploit, as shown in previous studies [9] [18]. For example, we assume a die with two planes. Without considering GC, the operations that access the same die can be categorized into one of the following four cases. In this work, we focus on write operations as they are much slower than read operations and thus have larger impact on the overall performance.

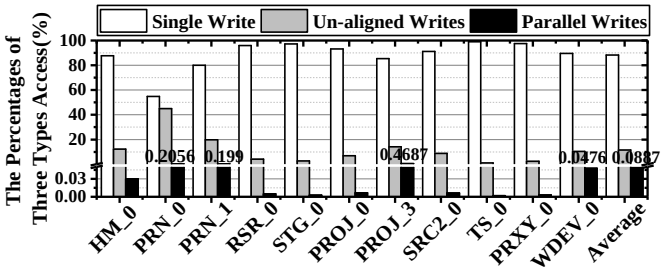


Fig. 2. The percentages of write operations in three cases.

`Case 1`: Operations are issued to one plane only (**Single Write**). In this case, the write operation will introduce unaligned write point;

`Case 2`: Two different types of operations are issued to the two planes of the die. Due to the operation type restriction, the operations cannot be processed in parallel;

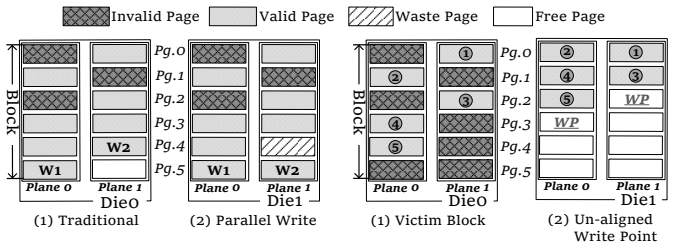
`Case 3`: Two same type operations with unaligned in-plane addresses are issued to the two planes of the die (**Unaligned Writes**). Due to the address restriction, the operations cannot be processed in parallel either;

`Case 4`: Two same type operations with aligned in-plane addresses are issued to the two planes (**Parallel Writes**). In this case, they can be processed in parallel.

We next analyze how to address the four cases to fully exploit the plane level parallelism. For Case 2, mixed operations cannot be scheduled in parallel due to the circuit restriction of *multi-plane command*. We then collect the numbers of operations falling in Case 1, Case 3 and Case 4, respectively, and report the results in Figure 2. The experiment setting details can be found in the experiment section. We have two observations from the results: (i) plane level parallelism is far from well utilized; (ii) a large percentage of write operations issued to the die are unaligned write operations, which can be exploited for performance improvement.

First, a naive solution to address the above issues is to write data at the aligned points greedily [10]. However, if the current write points are unaligned, writing data at the aligned points lead to wasted space. For example, we assume there are two planes per die, one block per plane, and six pages per block, as shown in Figure 3(a). In Figure 3(a)-(1), the current write points are unaligned. Traditionally, if two write operations, W1 and W2, are issued to the two planes in the same die, they will be processed sequentially. If they are written to the aligned pages, a free page in Plane 1 would be wasted, as shown in Figure 3(a)-(2). *In this work, we strive to design a write construction scheme to align the write points in each die.*

Second, internal SSD activities, e.g., GC, also introduce non-negligible performance impact [13] [14]. Given a die with multiple planes, if one plane activates GC, the other planes cannot be accessed before this GC finishes. To solve this problem, Shahidi *et al.* proposed to activate GCs in all



(a) Unaligned write operations (b) Unaligned write operations in GC

Fig. 3. The problems of unaligned write operations.

planes at the same time so that GC induced time cost can be overlapped [9]. To avoid significant parallel GC induced write amplification, ParaGC first selected a block containing most invalid pages in a plane, then, if its paired block in another plane contains enough invalid pages, these two blocks can be reclaimed by GCs simultaneously. Otherwise, only one block is processed by GC. However, such a solution faces two issues: `First`, since the number of valid pages in paired blocks are different, ParaGC may lead to unaligned write points across different planes after valid page movements. For example, in Figure 3(b), after moving valid pages in each plane in Figure 3(b)-(1), the new write points (*WP* in the figure) become unaligned, as shown in Figure 3(b)-(2). `Second`, if there is only one block is processed by GC, write points will be unaligned while reclaimed free block is allocated and its paired block still has not been reclaimed. That is, to maintain aligned write points at all time, we need to construct *multi-plane* oriented writes for host requests and GC induced operations.

IV. SPD: From Plane to Die Parallelism Exploration

A. Overview

To maximize plane level parallelism, the access addresses of writes on all planes in the same die should be aligned at all time. In this work, we propose SPD, an SSD *from plane to die* framework, to exploit the plane level parallelism for performance improvement by smartly maintaining aligned write point for the multi-planes in each die all the time. Basically, SPD takes the following strategies to achieve the objective, as shown in Figure 4. SPD adds two new components — a die level write construction and a die level GC. The die level write construction is designed to maintain aligned write points for host writes. The die level GC is designed to maintain aligned write points for GC induced page movement. Note that for other activities, such as WL, they can adopt the same design principle of GC. For simplicity, only GC is taken as an example in this paper. For die level write construction, SPD exploits the SSD buffer to choose N dirty pages and writes them back to one die simultaneously. This helps to convert one die access to N page writes at the aligned in-plane address. This is referred to as `Die-Write`. Similarly, the read access to the die is referred to as `Die-Read`. Note that `Die-Read` only needs to read required number of data, which does not introduce any read amplification. For die level GC, it is activated at the multiple planes in a die at the same time. In addition, all writes induced from the valid page movements is processed in the unit of N page writes to maintain the aligned write point. This is referred to as `Die-GC`. N is set to two in the following discussion while we evaluate different N values in the experiments. We will elaborate the details of these two components in following sections.

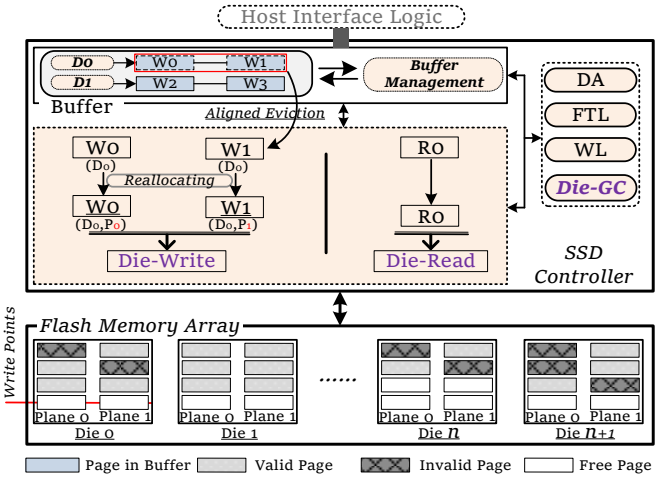


Fig. 4. The Overview of the *from plane to die* framework

B. Die Level Write Construction

Given that *multi-plane commands* would be disabled if the in-plane addresses are mis-aligned, the basic idea of die level write construction is to maintain aligned write points all the time by write the same amount of data synchronously to all planes in the same die. That is, (1) the amount of data issued to a die should be a multiple of N pages, assuming there are N planes in a die; and (2) the starting locations of data should be aligned for all the planes in the same die. With this scheme, whenever there are multiple write operations issued to a die, they can be processed in parallel.

SPD exploits SSD buffer to assist die level write construction. An SSD buffer evicts a multiple of N dirty pages from one die at a time such that these pages can be written using `Die-Write`. For data allocation, we adopt a plane level dynamic allocation scheme [11]. The data allocation at higher levels can either be static or dynamic, as discussed in Section 2.1. In the following discussion, we assume static allocation at the channel, chip, and die levels.

1) Buffer Supported `Die-Write`: Figure 5 illustrates how the SSD buffer assisted `Die-Write` works. Figure 5(a) shows how the SSD buffer is organized. It maintains a die queue that keeps a list of dirty pages for each die in the system. The pages in each list are linked together using LRU algorithm. The data evicted from the buffer are written to their corresponding dies. To balance the number of writes sent to different dies, SPD adopts round-robin to choose the next die from which its LRU pages are evicted.

For the example, in Figure 5(b), the SSD has four dies, each die has two planes, and the current turn is Die 0. When the SSD buffer is full and there is a host requirement for inserting

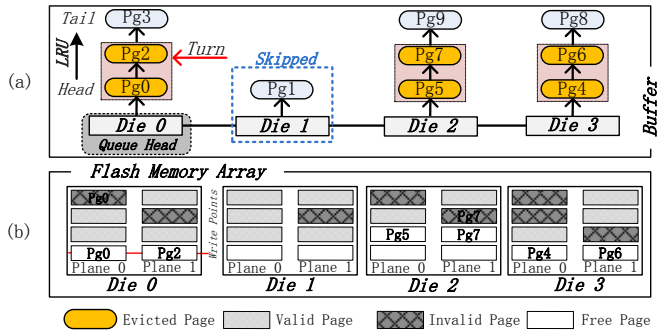


Fig. 5. Organization of write buffer and the die level write construction

five dirty pages to the buffer, SPD chooses the victim dies with at least two dirty pages (i.e., two is the number of planes in a die) and evicts the two LRU pages from each selected die. In the example, it first chooses Die 0 and then skips Die 1 as the latter does not have enough dirty pages. It continuously chooses Die 2 and Die 3 and then evicts two pages from Die 0, 2, and 3, respectively.

From this example, the write points of all planes are effectively aligned. The proposed scheme may evict one more dirty page than the number of dirty pages from the host. Since one `Die-Write` takes the same amount of time as one page write, the scheme is able to speed up the storage access if there exist several dirty pages evicted to the same die. But if only one dirty page from the host, evicting one more dirty page can align the write points without introducing additional time cost. In addition, since all `Die-Writes` operations can be scheduled in parallel, SPD avoids the access conflicts on the same die [3] [18]. Due to that we always evict the pages at LRU positions, the write amplification can be minimized.

Since the addresses of requested data are fixed, die level read operations cannot be constructed the same way at that for `Die-Write`. In this work, `Die-Read` only read the requested data, i.e., if there exist read operations with aligned access locations, they can be issued to the die in parallel; otherwise, only single page read gets processed next. The goal of `Die-Read` is to maximize the number of *multi-plane command* supported read operations without introducing read amplification.

2) Implementation and Analysis: Most of the state-of-the-art SSDs have equipped with a RAM based buffer inside SSD controller for metadata and data caches [1] [27] [28] [29] [34]. The buffer sizes range from 8MB in early products to 1GB in recent ones. To assist die level write constructions, SPD enhances the SSD buffer management to expose more parallel processing opportunities.

Different from traditional buffer management scheme, SPD needs to evict a multiple of N dirty pages from one die queue. In this work, the N pages of dirty data at the head of LRU are selected for eviction. When inserting new dirty pages to the buffer, SPD first checks if they are already in the queue and moves hit pages to the tail of the queue. Comparing to traditional LRU eviction, SPD evicts N pages instead of one page at a time. SPD does not require an extra built-in buffer and thus does not introduce extra space demand. However, SPD requires a minimal of $M*N*Size\_of\_Page$-byte buffer for smooth buffer management where M is the number of dies in an SSD, and each die has N planes. When the number of planes within each die increases, the minimal buffer size increases as well. A tradeoff exists between the minimal buffer space requirement and the number of planes within a die. Most existing SSD devices have two or four planes per die [1] [35] [23] [36] [32], where the space requirement can be easily met. For example, for a 512GB SSD, with 16GB die, two or four planes in each die and 4KB page size, the minimal buffer is 256KB for two planes and 512KB for four planes, which can be satisfied by most existing SSD products.

Another issue that SPD needs to consider is the power interruption induced data loss. Since the write buffer is used to store dirty data, these data would be lost when there is a sudden power failure. This is often mitigated by integrating a super capacitor, a popular scheme in state-of-the-art SSDs (such as PCIe SSDs) for buffer protection [37] [38] [39] [34] [40]. In addition to capacitor protection, non-volatile memories, such as 3D-Xpoint [41], Phase change memories [42], can also be employed as the write buffer to mitigate data loss under sudden power failure.

C. Die Level GC

A GC process includes three steps: victim block selection [14] [1]; valid page movement; and victim block erase. The dominate cost of a GC comes from valid page movement [13]. The design goal of `Die-GC` is to speed up the GC process with minimal GC cost. For this purpose, SPD activates GC at all the planes in the same die at the same time with carefully selected victim blocks. By adopting `Die-Write` instead of sequential page writes, SPD improves reclaim effectiveness by reducing the most timing cost. We elaborate the details as follows.

1) GC Process: Figure 6 shows an example for `Die-GC`. Different to the traditional GC process, `Die-GC` includes four steps: First, SPD selects N blocks from the N planes of the target die — one from each plane and all the selected blocks share the same in-plane addresses. The selection process takes the N aligned blocks as a GC unit. During this process, we adopts the greedy based victim block selection [13] [1], where the N blocks with maximal invalid pages are selected. With this scheme, the total GC cost will be minimized. Second, SPD uses `Die-Read` (in Section IV-B1) to read the valid pages to the SSD buffer. Third, after reading N pages of valid data, SPD groups the N pages of data to construct a `Die-Write` operation and then writes the valid data back to the die. Finally, when all the valid pages are written back, the N aligned blocks can be erased in parallel. Given SPD reclaims N blocks from one GC invocation, the GC gets triggered less frequently than that of the traditional one. In addition, since the N aligned blocks are taken as the GC unit for victim selection, GC with multiple blocks induced lifetime impact can be minimized.

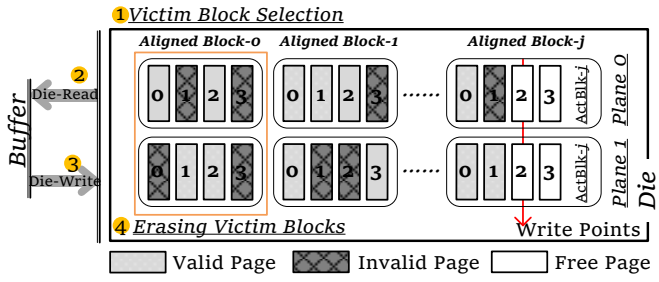


Fig. 6. The Process of Die-GC

For the example shown in Figure 6, let us assume the two aligned blocks 0 are selected as the victim blocks. According to `Die-GC`, the valid pages in these two blocks are read and written with `Die-Read` and `Die-Write`, respectively.

Step1: Read page 0 from plane 0 and page 1 from plane 1 to the SSD buffer. Since they are not aligned, they are read sequentially.

Step2: Group the two valid pages together to construct a `Die-Write` operation and written them back to the current aligned write point of both planes at block j. The current write points are marked using red arrows in the figure.

Step3: Then, read page 2 from plane 0 and plane 1 to the SSD buffer. These two pages are read in parallel as they have aligned addresses.

Step4: Repeat step (2) for the last two valid pages.

Step5: Then, erase the two victim blocks in parallel. From the above discussion, `Die-GC` significantly reduces GC cost because it maintains aligned write points in the die such that many strip reads and writes can operate in parallel.

An exception for the above scheme happens when the total number of valid pages in the victim aligned blocks is odd. In this case, the write points of different planes become misaligned after GC. To address this issue, the last `Die-Write` operation is constructed from remaining valid pages and dirty pages from the write buffer (as discussed in Section IV-B).

2) Implementation, Analysis and Discussion: We next elaborate the implementation overhead of SPD. We identify the construction of `Die-Write` as the most critical component in SPD. Since die level GC reclaims more blocks from each invocation, more data need to be transferred from the planes to the SSD controller sequentially, which introduces larger transfer cost. However, the cost of writing valid pages is much higher than that of data transfer [1] [10]. By writing multiple pages in parallel, SPD reduces the overall GC cost even though the data transfer cost increases.

Given that SPD transfers more data to the write buffer in the controller, it demands larger data storage. Considering the worst that all dies are activated with the die level GC, each die needs at least N pages in the write buffer. For a typical SSD setting as presented in Section 4.2, the required buffer size for `Die-GC` is 256KB for a 512GB two-plane SSD and 512KB for a 5I2GB four-plane SSD. In summary, the storage requirement is modest for modern SSDs. In addition, there are no additional power, implementation area and latency costs for the framework. The proposed work can be easily implemented for state-of-the-art SSDs.

In the discussion, we assume SPD adopts 4KB flash memory page. However, recent studies proposed the adoption of larger flash pages [43] [9]. SPD remains effective for larger page sizes. This is because there still exist multiple planes within a die so that several big data write operations also can be processed in parallel using *multi-plane command*. For `Die-Read`, a sub-page read operation [44] may be adopted to mitigate read amplification resulted from reading data from big flash pages.

V. Experiment Setup

A. Simulated SSD Devices

Due to that the proposed scheme needs firmware support of SSDs, in this work, we use a popular trace driven simulator, SSDsim [10], to evaluate the effectiveness of the proposed framework. In order to simulate a state-of-the-art SSD, SSDsim is significantly extended based on ONFI [8]. During the evaluation, a 512 GB SSD is simulated, and page mapping and greedy based GC scheme are adopted [10] [3]. The threshold value for GC activation is set to 7% [9]. To triggering GC process, SSD is warmed up by filling SSD with valid and invalid data ahead. The warming up process contains two steps: first, each plane of the SSD is randomly filled with data from 93% to 95% to trigger GC immediately, of which 80% are valid; second, the evaluated workload is pre-processed in the SSD to validate read data [13]. The over-provisioning ratio is set to 25%, which complies with the setting in previous work [9]. For the data allocation scheme, the most widely used Channel-Chip-Die-Plane scheme is adopted. The experiment settings represent an aged state-of-the-art SSD. Other details are presented in Table I.

TABLE I
Parameters of the Simulated SSD [9].

SSD Configuration	512GB;16 Channels; 8 Chips/Channel; 1 Die/Chip; 2 Planes/Die;2048 Blocks/Plane; 256 Pages/Block; 4KB Page;
Timing Parameters	0.075 ms for page read; 1.5 ms for page write; 3.8 ms for block erase; 25 ns for byte transfer.

During the evaluation, a DRAM buffer is configured in the SSD. We set the buffer size to be 1‰ of the footprint of the evaluated workload [27] [45], which helps to prevent setting a large buffer from generating biased results in evaluation. The default data organization of die lists in the buffer is designed based on the scheme of the Element-Level Parallelism Optimization (EPO) [46]. EPO evicts dirty pages from buffer based on its die location so that the utilization of die level parallelism can be maximized. The data are organized in LRU for each die list of the buffer.

B. Evaluated Workloads

The workloads studied in this work include a subset of MSR Cambridge Workloads from servers [47]. These workloads are

widely used in previous works for studying SSD performance [18] [9] [14]. The characteristics of workloads are presented in Table II. Each workload is characterized by three metrics: $W/R\ Ratio$, FP, R_V, W_V, R_S and W_S. $W/R\ Ratio$ represents the write and read operation ratios, FP is the footprints of each workload, R_V is the total amount of read data, W_V represents the total amount of written data, R_S represents the average size of read requests, and W_S is the average size of write requests.

TABLE II
THE CHARACTERISTICS OF EVALUATED WORKLOADS

Workloads	W/R Ratio[§]	FP[§]	R_V[§]	W_V[§]	R_S[§]	W_S[§]
HM_0	67.9%	1.35	6.9	15.2	11.2	11.6
PRN_0	93.7%	2.93	3.0	20.5	24.8	11.6
PRN_1	32.1%	5.16	31.4	10.9	24.2	11.4
RSR_0	90.7%	0.31	1.8	14.6	15.0	12.6
STG_0	76.9%	0.28	7.4	9.3	33.6	12.6
PROJ_0	82.9%	1.58	7.2	56.5	21.9	35.7
PROJ_3	4.89%	1.86	21.6	2.8	11.9	29.9
SRC2_0	88.6%	0.52	1.9	13.6	12.2	11.0
TS_0	82.6%	0.57	4.9	15.9	17.5	11.8
PRXY_0	97.06%	0.17	0.27	5.8	9.6	6.2
WDEV_0	79.9%	0.34	3.2	9.2	16.5	12.1

[§] W/R Ratio: Write and Read Requests Ratio;
FP: FootPrint (GB);
R_V/W_V: Read/Write Data Volume (GB);
R_S/W_S: Average Read/Write Request Size (KB).

C. Evaluated Schemes:

Five schemes are implemented to show the effectiveness of SPD.

Baseline-D: This scheme is implemented to represent the traditional SSD design [10]. The buffer management of Baseline-D adopts EPO to exploit die level parallelism through adding dirty pages to different die lists based on their die locations [46]. With this organization, dirty data evicted from write buffer can be distributed to different dies so that die level parallelism can be exploited;

Baseline-P: This scheme is similar to Baseline-D. The difference is that Baseline-P evicts dirty data based on their plane locations to further exploit plane level parallelism. In this case, dirty pages accessing different planes within the same die are evicted at a time. Baseline-P evenly distributes dirty pages to different planes to better exploit plane level parallelism, which is similar to the previous studies [48] [18];

TwinBlk: This scheme is designed based on the work proposed by Tavakkol *et al.* [11], which aims to align write points of all planes in a die via round-robin policy. In this case, several host requests can be processed in parallel when write points are aligned. During GC process, the adopted round-robin policy is designed to align write points of active blocks in victim blocks as well, aiming to move valid pages with the support of *multi-plane command*;

ParaGC: This scheme is designed by Shahidi *et al.* [9], which aims to align valid page movement during GC to minimize the GC cost. Differing from TwinBlk, ParaGC aligns write points of active blocks through sequentially moving valid pages to one active block until write points of all planes are aligned. After that, with cache assistance, all valid pages can be written back to active blocks with the support of *multi-plane command*;

SPD: This is the proposed framework, which includes `Die-Write` and `Die-GC`.

VI. EXPERIMENT RESULTS AND ANALYSIS

In this section, SPD is evaluated with two scenarios based on whether GC is triggered. For the first scenario without triggering GC, it is evaluated to show the advantages of the proposed `Die-Write` scheme. For the second scenario with triggering GC, it is evaluated to show the effectiveness of SPD, including `Die-Write` and `Die-GC`. In addition, the `Die-GC` is also evaluated in term of its cost and lifetime impact. Finally, the impact of different buffer sizes and results on SSD with 4 planes per die are presented.

A. Experiment Results without GC

(1) Write Latency Evaluation: Figure 7 shows the results of write latency for the five schemes. Note that, since ParaGC is designed to optimize GC process, the results of ParaGC in this part are same to that of Baseline-D. The results show that SPD achieves write latency reduction for all evaluated workloads. For example, for HM_0, PRN_0, PROJ_3, SRC2_0 and PRXY_0, the write latency is reduced by more than 15% compared with Baseline-D. These results show that deploying `Die-Write` to maintain aligned write points for the multiple planes in a die is important in improving the access performance. In Figure 8, we collected the percentages of write operations processed by *multi-plane command*. The results show that the proposed `Die-Write` is able to maintain aligned write points for all write operations. However, this is not a promise for the other schemes.

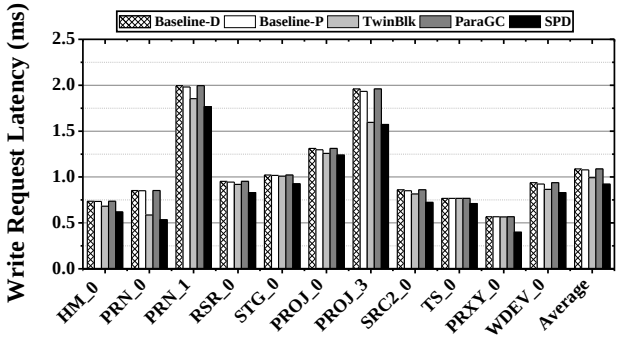


Fig. 7. Write Latency Reduction.

To obtain more details, we compare SPD with other two schemes, Baseline-P and TwinBlk. Two observations can be concluded from the results: **First**, compared with these two schemes, SPD achieves the best write performance. Baseline-P is proposed to distribute the same type requests to all planes evenly. However, the address restriction is not taken into consideration. As a result, Baseline-P only achieves little write latency reduction, which is only up to 1.4%. TwinBlk

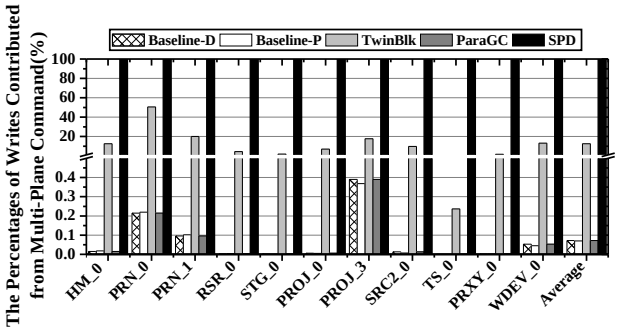


Fig. 8. Percentages of Write Operations Processed by *multi-plane command*.

aims to align write points of all planes in the same die as well. However, the write points still may be unaligned due to the unaligned accesses on planes of the same die. On average, TwinBlk achieves 7.8% write latency reduction compared with Baseline-D. As shown in Figure 8, the percentages of write operations processed by *multi-plane command* for Baseline-P is similar to that of Baseline-D. For TwinBlk, the percentage is largely increased compared with Baseline-D. **Second**, for several workloads, TwiBlk only achieves similar performance improvement to that of Baseline-D, such as RSR_0, STG_0, TS_0 and PRXY_0. This can be explained from the results in Figure 8, where the percentage of write operations supported by *multi-plane command* is limited. The reason is that TwinBlk cannot guarantee aligned write points for all planes all the time.

For read latency, the average read latency improvement compared with Baseline-D is presented in Table III. The results show that read latency is similar among the five schemes. The key reasons are from two aspects: first, read requests of all evaluated schemes are processed with higher priority [49] [50] [3]; second, `Die-Read` is designed to only read requested data. In conclusion, the proposed `Die-Read` is same to that of normal read operations without introducing read amplification.

TABLE III
READ LATENCY IMPROVEMENT WITHOUT GC

	Baseline-D	Baseline-P	TwinBlk	ParaGC	SPD
Reduction	0	0.049%	0.011%	0%	0.096%

(2) Plane Utilization: *Plane Utilization* is defined to present the average number of planes being occupied in parallel. In order to obtain plane utilization, the number of planes being accessed is counted when each buffer eviction process is completed. Figure 9 shows the plane utilization (Bars) and the maximal number of planes being accessed in parallel (Dots+Line) for the five schemes. The results have a matching pattern with the write performance improvement in Figure 7. SPD can significantly increase the plane utilization through doubling the number of parallel planes with satisfying the restrictions of *multi-plane command*. On average, the plane utilization is increased by 36.5% compared with Baseline-D. For the maximal number of planes accessed in parallel, all planes of the SSD can be accessed in parallel for most workloads. However, for Baseline-D, Baseline-P and TwinBlk, there still exists a large gap compared with SPD. In conclusion, `Die-Write` is not only able to increase plane utilization, but also can make a full use of all planes of the SSD.

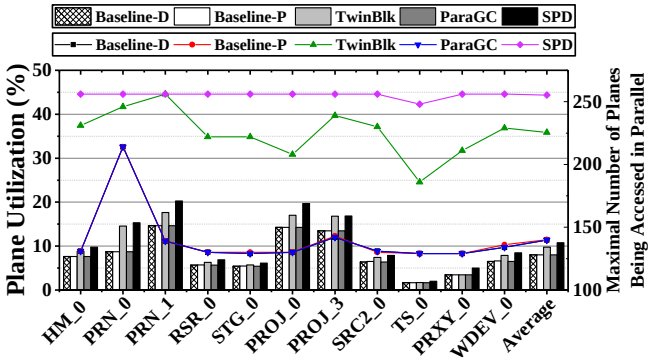


Fig. 9. The Plane Utilization and Maximal Number of Planes being Accessed in Parallel.

(3) Buffer Hit Ratio: Differently from previous work, `Die-Write` may need to evict more data from the buffer to align the write points. In this case, it may have impact to the hit ratio of buffer. Figure 10 presents the results of buffer hit ratios for the five schemes. The results show that SPD has little impact to the hit ratio of buffer. The average buffer hit ratio is reduced by only 1.92%, which is negligible. The reason for the slight reduction is that `Die-Write` is designed with following principles: first, it always only need to evict one more dirty page, which is critical in aligning write points; second, the buffer is designed to only evict the cold dirty data from the LRU position.

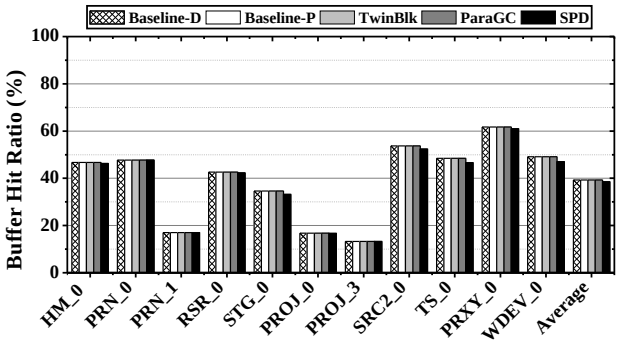


Fig. 10. The Buffer Hit Ratios of Evaluated Schemes.

B. Experiment Results with GC

Figure 11 shows the results of write latency with GC triggered. The results show that SPD is able to significantly reduce the write latency for all workloads. The write latency is reduced by 48.61%, 47.65%, 42.05%, and 28.58% compared with Baseline-D, Baseline-P, TwinBlk, and ParaGC, on average. The significant improvement comes from two aspects: **First**, SPD constructs aligned write access to reduce write latency, which has been verified in Section VI-A. **Second**, the

GC cost is further reduced through moving all valid pages with the support of `Die-Write` and reclaiming two planes at once time.

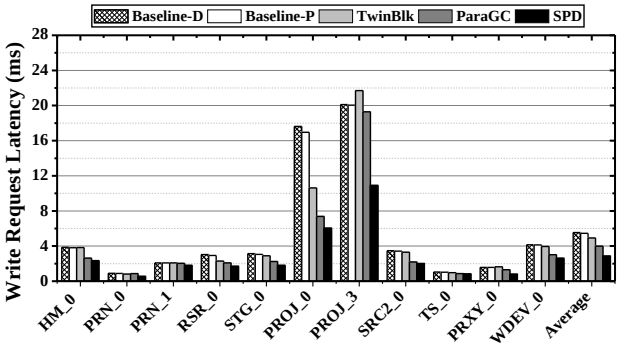


Fig. 11. The Write Latencies of Evaluated Schemes.

To understand more details, the total GC costs are presented in Figure 12. The results show that first, TwinBlk generally has much higher cost than ParaGC. On average, compared with Baseline-D, total GC cost of ParaGC is reduced by 30.8% while TwinBlk only reduces the total GC cost by 6.9%. For ParaGC, it activates GCs in paired planes only when the number of free pages in the other plane is smaller than 7%. In this case, it can avoid introducing high GC cost while moving valid pages. In addition, ParaGC proposed to align write points during the process of valid page movement so that valid pages in the same position of paired planes can be read and written in parallel. However, for TwinBlk, it activates paired GCs without considering the number of valid pages in the paired planes. In this case, more valid pages from paired planes may be moved during GC process. In addition, TwinBlk adopted round-robin policy. If current write points are not aligned, valid pages having same position in different planes still can not be read and written in parallel. Therefore, for some workloads, the total GC cost of TwinBlk is larger than Baseline-D. Second, even though SPD also activates GC at the all planes at the same time, it is proposed to regard the whole die as the smallest access unit and all the write operations during GC are processed via `Die-Write`. As a result, the total GC cost is reduced by 36.4%, on average. In conclusion, SPD achieves the best write performance compared with all other related works.

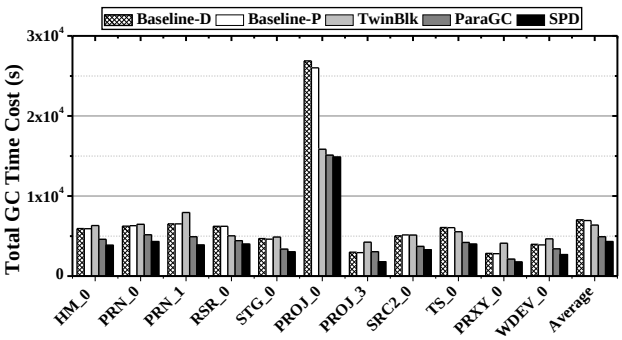


Fig. 12. Total GC Cost of Evaluated Schemes.

For read latency, results of read latency improvement with considering GC are presented in Table IV. Similarly, the read latency is similar among each scheme. The results show that SPD has no impact to read access with significant write performance improvement.

TABLE IV
READ LATENCY IMPROVEMENT WITH GC

	Baseline-D	Baseline-P	TwinBlk	ParaGC	SPD
Reduction	0	0.052%	-0.042%	1.144%	1.203%

C. GC Evaluation

In this part, `Die-GC` is evaluated. First, the average GC cost and the number of triggered GC in different schemes are evaluated. Second, the number of erase operations induced by GC is collected to show its impact on the lifetime of SSDs.

(1) Average GC Cost: Average GC costs are collected in Figure 13. In the Figure, the average GC cost is broken into four parts: read cost, write cost, transfer cost and erase cost. Read cost is the cost in reading valid pages from the victim block; write cost is the cost in writing the valid data to free pages; transfer cost is the cost in transferring the valid data among planes or between controller and chips; and erase cost is the time cost in erasing the victim block. The results show that the write cost takes the dominate part of the total cost [18]. This is because write latency of flash memory is several times of read latency. In addition, there are always a large number of valid page movement during GC. There are two observations from the results: **First**, SPD has the minimal GC cost compared with TwinBlk and ParaGC. Clearly, the reduced GC cost is from the `Die-Write` used in `Die-GC`, which is triggered to write dirty pages back to the multiple planes in parallel. For TwinBlk, it also trigged GC in the paired planes. However, TwinBlk adopted round-robin policy for write operations among planes, which is not able to always align the write points. In this case, many valid pages written back may be processed sequentially. **Second**, the GC cost of SPD is similar to that of Baseline-D and Baseline-P. As presented in the technique part, `Die-GC` is designed to reclaim several blocks in one GC. Several block reclaiming costs are similar with single block reclaiming cost in Baseline-D and Baseline-P due to that we carefully select victim blocks among planes as a single unit and use `Die-Write` to speed up the process.

(2) GC Count: Figure 14 shows the total number of triggered GCs during runtime. We can find that `Die-GC` highly reduces the number of GCs. Therefore, the frequency of triggering GC is reduced. The results show that GC count is reduced in the range of 32.9% to 50.1%, compared with Baseline-D. As a result, the total GC cost during whole runtime can be highly reduced as well so that the performance of SSDs can be improved. For related works, the number of triggered GCs in Baseline-P is similar to Baseline-D. Both TwinBlk and ParaGC can reduce the number of triggered GCs as well. This is because that TwinBlk and ParaGC erase more

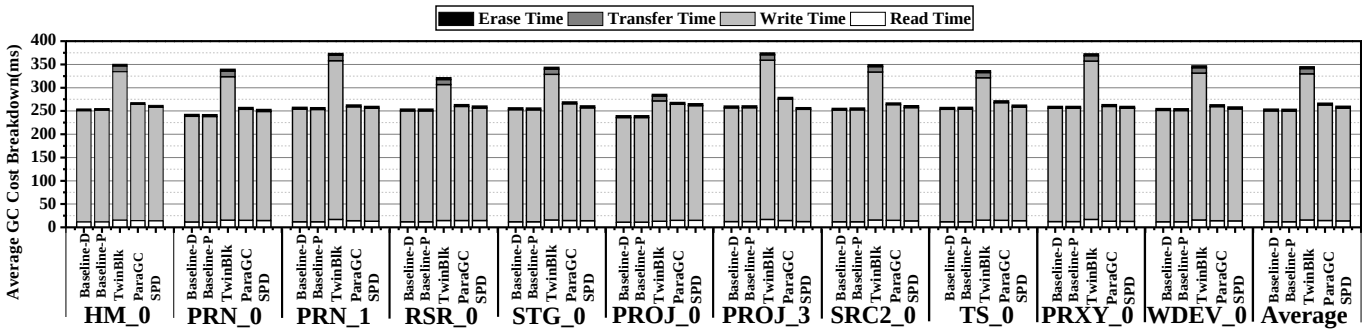


Fig. 13. Average GC Cost Breakdown of Evaluated Schemes.

blocks in each GC process as well. But for TwinBlk, it selects victim blocks inefficiently so that its GC counts are slightly higher in most cases. For a exception, PROJ_0, since SPD may slightly increase write operations, the total triggered GC count of SPD may be slightly increased.

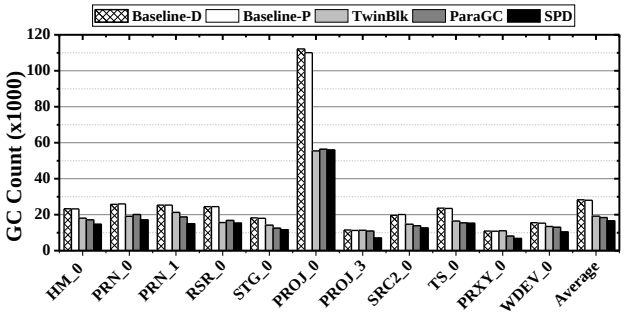


Fig. 14. The Total Number of Triggered GC.

(3) GC Induced Erases: Figure 15 shows the number of erase operations for the five schemes. Since TwinBlk, ParaGC and `Die-GC` are designed to erase more blocks in each GC process, the number of erase operations are larger than that of Baseline for most workloads. The reason is that, reclaiming blocks from different planes at once time may trigger premature GCs [51] [52]. However, the results show that the number of erase operations of `Die-GC` is much smaller than TwinBlk and ParaGC. For example, TwinBlk, in the worst case, introduces more than 102.2% erase operations for PRXY_0, compared with Baseline-D. ParaGC, introduces more than 65.8% erase operations compared with Baseline-D. Compared with these two related works, SPD introduces fewer erase operations in most cases. On average, the number of erase operations is reduced by 13.43% and 10.04% compared with TwinBlk and ParaGC. The reason comes from that `Die-GC` is triggered with regarding the whole die as the smallest unit without introducing additional valid page movements.

D. Sensitive Studies

(1) Buffer Size Impact: In this part, the write intensive workload, RSR_0, is selected for buffer size sensitivity study. Buffer size is different within different devices. Its impact on SPD is presented. Figure 16 shows the results of the normalized write latencies of the five schemes by varying

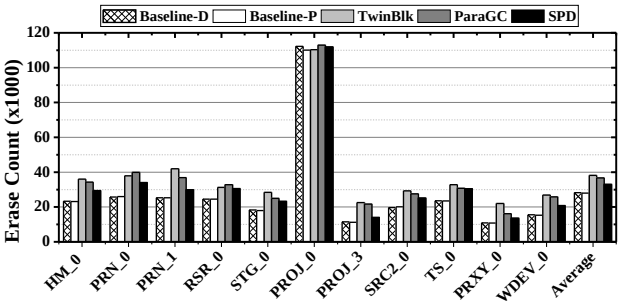


Fig. 15. The Total Number of Erase Operations

buffer size from 256KB to 16MB. During the evaluation, GC is not triggered to only understand the impact from different buffer sizes. Two observations can be concluded from the results. **First**, with larger buffer size, the write latencies of all schemes can be further reduced. This is because that more dirty pages can be stored and higher hit ratio can be achieved. **Second**, compared with other schemes, stable write latency reduction is achieved by SPD with different buffer sizes. The proposed framework is designed to align the write point of planes all the time. It has benefit once there are multiple write operations issued to a die.

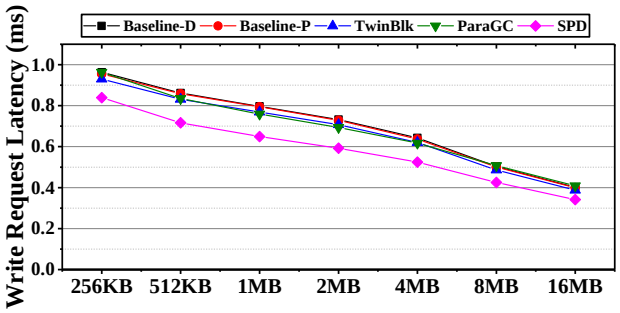


Fig. 16. Write Latency with Different Buffer Sizes.

(2) Four-Plane SSD Evaluation: In this part, SSD with four planes per die is evaluated for SPD. For the four planes of a die, each paired planes can be accessed in parallel with the support of *multi-plane command* [1] [10]. The results of write latencies for Baseline-D, Baseline-P, TwinBlk, ParaGC and SPD are presented in Figure 17, where GC is triggered. **First**, Baseline-P has similar write latency to that of Baseline-

D. Four-plane SSD requires that only paired plane 0&1 or 2&3 can be processed in parallel. Only a few write operations can be processed with the support of *multi-plane command*. **Second**, the write latency of TwinBlk is increased compared with Baseline-D, especially for PROJ_3. The reason comes from that more blocks are immaturely reclaimed so that the total GC cost is highly increased. Take PROJ_3 as an example, the total GC cost of TwinBlk is increased by 63.4% (Due to space limitation, we did not show the total GC cost). **Third**, the write latency of ParaGC is reduced by 19.3% compared with Baseline-D. Since triggering GC in paired planes will block the whole die, there still exist idle planes when there are four planes in a die. **Last**, for SPD with four-plane SSD, the write latency is further reduced. This is because that all four planes are regarded as one unit in *Die-Write* and they are reclaimed at one time by *Die-GC* as well. Therefore, the frequency of triggering GC can be highly reduced when the number of planes in a die increases. On average, compared with Baseline-D, SPD achieves 65.6% write latency reduction, on average.

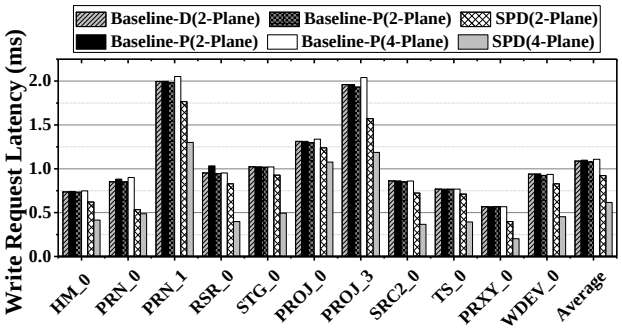


Fig. 17. Write Latency with 4 Planes per Die.

VII. Related Works

In this section, related works on improving the plane level parallelism and reducing GC impact on performance are presented, respectively.

(1) Plane Level Parallelism Exploration: In order to improve plane level parallelism, several previous works have been proposed. Gao *et al.* [18] and Jung *et al.* [48] proposed to increase the potential of using *multi-plane command* through distributing requests belonging to different planes at one time. Similarly, Abdurrab *et al.* [31] proposed DLOOP to modify mapping policy to evenly distribute data across planes based on a fixed location calculation. However, the achieved performance is limited since they highly depend on the access patterns of workloads to match the limitations of *multi-plane command*. On the other hand, Tavakkol *et al.* [11] and Hu *et al.* [10] proposed to align writing points of planes. Tavakkol *et al.* [11] proposed to maintain the write points to distribute writes among planes in round-robin fashion. However, due to the above mentioned unaligned access problem, plane level parallelism still can not be fully exploited. Hu *et al.* [10] proposed a greedy *multi-plane command*. They proposed to allocate new writing points in the same position. However, this will waste space.

Different from all these works, SPD is the first on proposing to align the write points in an active way. `Die-Write` is designed to align the write point all the time. In this case, all write operations issued to multiple planes in a die can be processed in parallel.

(2) Garbage Collection Impact Minimization: Previous works aiming at reducing GC impact on performance can be classified into two groups: The first group proposed to reduce the time cost of GC activity [13] [53]; For example, Gao *et al.* [13] proposed to reduce the time cost of valid page movement through migrating valid pages to idle chips. Park *et al.* [53] proposed a new hotness identification method for accurately capturing the recency and frequency of data. The second group proposed to schedule requests or GCs to reduce the impact on performance of SSDs [54] [14] [12]. For example, Wu *et al.* [14] used cache to store requests conflicted by GC. Jung *et al.* [54] proposed to advance or delay GC through moving the time-consuming activity from busy period to idle period. Choi *et al.* [12] proposed to combine host I/O operations with valid pages migration. However, the aforementioned GC optimization methods still have not taken unaligned access problem of plane level parallelism into consideration.

There are two works proposed to reduce GC impact resulted from unaligned access problem. Shahidi *et al.* [9] proposed ParaGC to select paired planes, where GC activities can be processed in parallel. However, if the paired planes can not be found, unaligned access problem still exist. Tavakkol *et al.* [11] proposed TwinBlk, which can minimize the unaligned access induced impact on GC. TwinBlk is designed to trigger GCs on all planes of the same die simultaneously so that symmetric victim blocks on planes can be reclaimed in parallel. During this process, valid pages are evenly moved to all planes in round robin policy for aligning write points of all planes.

Different from these works, SPD uses `Die-GC` to speed up the GC process and reduce the GC cost. `Die-GC` is designed to select multiple blocks in the unit of die and adopt `Die-Write` to speed up the GC process.

VIII. Conclusion

In this work, a *from plane to die* optimization framework is proposed to exploit the plane level parallelism, which is the last level parallelism of SSDs. Two components are designed in the framework: die level write construction and die level GC. Different from previous work, this work is the first which is able to maintain the aligned write points for the multiple planes for each die at the time. There are two components designed to align the write points of all planes in the same die all the time. In this case, the last level parallelism, plane level parallelism, is fully exploited to improve the performance of write requests and internal activities. Experiment results show that SPD achieves significant write performance improvement and much smaller lifetime impact compared with state-of-the-art works.

IX. Acknowledgment

This work is supported by NSFC 61772092 and 61572411, National Science Foundation 1718080, 1422331, 1725657 and 1617071.

References

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark S Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC*. USENIX, 2008.

[2] Feng Chen, Rubao Lee, and Xiaodong Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *HPCA*. IEEE, 2011.

[3] Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H-M Sha. Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In *MSST*. IEEE, 2014.

[4] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: A flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS*. ACM, 2009.

[5] Feng Chen, Tian Luo, and Xiaodong Zhang. Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *FAST*, 2011.

[6] Myoungsoo Jung and Mahmut T Kandemir. An evaluation of different page allocation strategies on high-speed ssds. In *HotStorage*. USENIX, 2012.

[7] Rino Micheloni, A Marelli, and S Commodaro. Nand overview: from memory to systems. In *Inside NAND Flash Memories*. Springer, 2010.

[8] ONFI. Open NAND Flash Interface Specification 4.1. Website, 2017. http://www.onfi.org/~/media/onfi/specs/onfi_4_1_gold.pdf?la=en.

[9] Narges Shahidi, Mohammad Arjomand, Myoungsoo Jung, Mahmut T Kandemir, Chita R Das, and Anand Sivasubramaniam. Exploring the potentials of parallel garbage collection in ssds for enterprise storage systems. In *SC*. IEEE, 2016.

[10] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *ICS*. ACM, 2011.

[11] Arash Tavakkol, Pooyan Mehrvarzy, and Hamid Sarbazi-Azad. Tbm: Twin block management policy to enhance the utilization of plane-level parallelism in ssds. In *Computer Architecture Letters*. IEEE, 2016.

[12] Wonil Choi, Myoungsoo Jung, Mahmut Kandemir, and Chita Das. Parallelizing garbage collection with i/o to improve flash resource utilization. In *HPDC*. ACM, 2018.

[13] Congming Gao, Liang Shi, Yejia Di, Qiao Li, Chun Jason Xue, Kaijie Wu, and Edwin Sha. Exploiting chip idleness for minimizing garbage collection induced chip access conflict on ssds. In *TODAES*. ACM, 2017.

[14] Suzhen Wu, Bo Mao, Yanping Lin, and Hong Jiang. Improving performance for flash-based storage systems through gc-aware cache management. In *TPDS*. IEEE, 2017.

[15] Suzhen Wu, Yanping Lin, Bo Mao, and Hong Jiang. Gcar: Garbage collection aware cache management with improved performance for flash-based ssds. In *ICS*. ACM, 2016.

[16] Myoungsoo Jung, Wonil Choi, Shekhar Srikantaiah, Joonhyuk Yoo, and Mahmut T Kandemir. Hios: A host interface i/o scheduler for solid state disks. In *ACM SIGARCH Computer Architecture News*. IEEE Press, 2014.

[17] Adrian M Caulfield, Laura M Grupp, and Steven Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. *ACM Sigplan Notices*, 2009.

[18] Congming Gao, Liang Shi, Cheng Ji, Yejia Di, Kaijie Wu, Jason Xue, and Edwin Sha. Exploiting parallelism for access conflict minimization in flash-based solid state drives. In *TCAD*. IEEE, 2017.

[19] Siddharth Choudhuri and Tony Givargis. Performance improvement of block based nand flash translation layer. In *CODES + ISSS*. IEEE/ACM, 2011.

[20] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. A log buffer-based flash translation layer using fully-associative sector translation. In *TECS*. ACM, 2007.

[21] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. System software for flash memory: a survey. In *EUC*. Springer, 2006.

[22] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. In *CSUR*. ACM, 2005.

[23] Seungjae Lee, Chulbum Kim, Minsu Kim, Sung-min Joe, Joonsuc Jang, Seungbum Kim, Kangbin Lee, Jisu Kim, Jiyoon Park, Han-Jun Lee, et al. A 1tb 4b/cell 64-stacked-wl 3d nand flash memory with 12mb/s program throughput. In *ISSCC*. IEEE, 2018.

[24] Yangyang Pan, Guiqiang Dong, and Tong Zhang. Exploiting memory device wear-out dynamics to improve nand flash memory system performance. In *FAST*. USENIX, 2011.

[25] Yeong-Jae Woo and Jin-Soo Kim. Diversifying wear index for mlc nand flash memory to extend the lifetime of ssds. In *EMSOFT*. IEEE, 2013.

[26] Bingsheng He, Jeffrey Xu Yu, and Amelie Chi Zhou. Improving update-intensive workloads on flash disks through exploiting multi-chip parallelism. In *TPDS*. IEEE, 2015.

[27] Hyojun Kim and Seongjun Ahn. Bplru: A buffer management scheme for improving random writes in flash storage. In *FAST*. USENIX, 2008.

[28] Nimrod Megiddo and Dharmendra S Modha. Arc: A self-tuning, low overhead replacement cache. In *FAST*. USENIX, 2003.

[29] Liang Shi, Jianhua Li, Chun Jason Xue, Chengmo Yang, and Xuehai Zhou. Exlru: a unified write buffer cache management for flash memory. In *EMSOFT*. ACM, 2011.

[30] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. In *Computer Architecture Letters*. IEEE, 2011.

[31] Abdul R. Abdurrab, Tao Xie, and Wei Wang. Dloop: A flash translation layer exploiting plane-level parallelism. In *IPDPS*. IEEE, 2013.

[32] Tomoharu Tanaka, Mark Helm, Tommaso Vali, and Ramin Ghodsi. 7.7 a 768gb 3b/cell 3d-floating-gate nand flash memory. In *ISSCC*. IEEE, 2016.

[33] Micron. Nand flash performance improvement using internal data move. Website, 2006. https://www.micron.com/resource-details/d8322e29-f893-4c73-ade6-ad341f7f2b32.

[34] Congming Gao, Shi Liang, Yejia Di, Qiao Li, Chun Xue, and H.M. Edwin Sha. An efficient cache management scheme for capacitor equipped solid state drives. In *GLSVLSI*. ACM, 2018.

[35] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. Mqsim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *FAST*. USENIX, 2018.

[36] Ryuji Yamashita, Sagar Magia, Tsutomu Higuchi, Kazuhide Yoneya, Toshio Yamamura, Hiroyuki Mizukoshi, Shingo Zaitsu, Minoru Yamashita, Shunichi Toyama, Norihiro Kamae, et al. 11.1 a 512gb 3b/cell flash memory on 64-word-line-layer bics technology. In *ISSCC*. IEEE, 2017.

[37] Min Huang, Yi Wang, Liyan Qiao, Duo Liu, and Zili Shao. Smart-backup: An efficient and reliable backup strategy for solid state drives with backup capacitors. In *HPCC*. IEEE, 2015.

[38] Jie Guo, Jun Yang, Youtao Zhang, and Yiran Chen. Low cost power failure protection for mlc nand flash storage systems with pram/dram hybrid buffer. In *DATE*. ACM, 2013.

[39] Woon Hak Kang, Sang Won Lee, Bongki Moon, Yang Suk Kee, and Moonwook Oh. Durable write cache in flash memory ssd for relational and nosql databases. In *SIGMOD*. ACM, 2014.

[40] Micron. 7100 m.2 nvme pcie ssd. Website, 2016. https://www.micron.com/~/media/documents/products/data-sheet/ssd/7100_m2_pcie_ssd.pdf.

[41] Wiki. 3d xpoint. Website, 2017. https://en.wikipedia.org/wiki/3D_XPoint.

[42] Ruijin Zhou and Tao Li. Leveraging phase change memory to achieve efficient virtual machine execution. In *VEE*. ACM, 2013.

[43] Xuebin Zhang, Jiangpeng Li, Hao Wang, Kai Zhao, and Tong Zhang. Reducing solid-state storage device write stress through opportunistic in-place delta compression. In *FAST*. USENIX, 2016.

[44] Feng Chen, David A Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *SIGMETRICS*. ACM, 2009.

[45] Sang-Won Lee, Bongki Moon, and Chanik Park. Advances in flash memory ssd technology for enterprise database applications. In *SIGMOD*. ACM, 2009.

[46] Tao Xie and Janak Koshia. Boosting random write performance for enterprise flash storage systems. In *MSST*. IEEE, 2011.

[47] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds: Analysis of tradeoffs. In *EuroSys*. ACM, 2009.

[48] Myoungsoo Jung, Ellis H Wilson III, and Mahmut Kandemir. Physically addressed queueing (paq): improving parallelism in solid state disks. In *ISCA*. IEEE, 2012.

[49] Tatyana Brokhman. Row scheduling algorithm in block layer. Website, 2012. https://lwn.net/Articles/509829/.

[50] Jaeho Kim, Yongseok Oh, Eunsam Kim, Jongmoo Choi, Donghee Lee, and Sam H Noh. Disk schedulers for solid state drivers. In *EMSOFT*. ACM, 2009.

[51] Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. In *TECS*. ACM, 2004.

[52] Li-Pin Chang and Chen-Yi Wen. Reducing asynchrony in channel garbage-collection for improving internal parallelism of multichannel solid-state disks. In *TECS*. IEEE, 2014.

[53] Dongchul Park and David HC Du. Hot data identification for flash-based storage systems using multiple bloom filters. In *MSST*. IEEE, 2011.

[54] Myoungsoo Jung, Ramya Prabhakar, and Mahmut Taylan Kandemir. Taking garbage collection overheads off the critical path in ssds. In *Middleware*. ACM, 2012.