Boosting the Performance of SSDs via Fully Exploiting the Plane Level Parallelism

Congming Gao, Liang Shi, Kai Liu, Chun Jason Xue, Jun Yang, and Youtao Zhang, *Member, IEEE*

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 that 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 article, 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. To further improve the performance of SSDs, host write operations blocked by GCs are suggested to be processed in parallel with GC induced valid page movements, bringing lesser waiting time cost of host write operations. 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, scheduling, performance improvement

1 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].

- C. Gao is with the College of Computer Science, Chongqing University, Chongqing 400044, P.R. China, and also with the University of Pittsburgh, Pittsburgh, PA 15260. E-mail: albertgaocm@gmail.com.
- K. Liu is with the College of Computer Science, Chongqing University, Chongqing 400044, P.R. China. E-mail: liukai0807@gmail.com.
- J. Yang and Y. Zhang are with the University of Pittsburgh, Pittsburgh, PA 15260. E-mail: juy9@pitt.edu, zhangyt@cs.pitt.edu.
- L. Shi is with the School of Computer Science and Technology, East China Normal University, Shanghai 200241, P.R. China. E-mail: shi.liang.hk@gmail.com.
- C. Xue is with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: jasonxue@cityu.edu.hk.

Manuscript received 9 July 2019; revised 18 Feb. 2020; accepted 8 Apr. 2020. Date of publication 15 Apr. 2020; date of current version 5 May 2020.
(Corresponding author: Liang Shi.)
Recommended for acceptance by C. Ding.
Digital Object Identifier no. 10.1109/TPDS.2020.2987894

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 be aligned. Unfortunately, a host often sends uneven numbers of write requests to different planes [9] and the activities originated from SSDs (e.g., garbage collection operations) are often imbalanced across different planes [9], [13]. Such asynchronicity leads to sub-optimal exploration of plane level parallelism and prevents modern SSDs from achieving further performance improvement.

To exploit plane level parallelism, Tavakkol *et al.* proposed *TwinBlk* to write data to the different planes in a die in a round-robin fashion [11] such that concurrent writes can be issued to different planes at the same time. However, the write points from different planes may be mis-aligned due to (1) single-page write operations; (2) GC or wear leveling activities originated inside the SSD [9], [13], [14], [15], disabling the concurrent writes in these cases. To reduce GC-induced write point mis-alignment, Shahidi *et al.* proposed *ParaGC* to activate GC from all planes of the same die at the same time [9], which opportunistically exploits the plane level parallelism when all the pages at the same address of

1045-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

different planes are valid. *TwinBlk* can concurrently activate multiple GCs by choosing multiple blocks with the same offset in different planes. It cannot process all valid page movements in parallel when not all paired pages are valid. Superpage enabled SSDs [1], [16], [17] strip requests to all planes in a die, which increases the number of sequential writes as well as concurrent write opportunities. However, activating GC in one plane introduces mis-aligned free blocks so that subsequent requests can not be processed in parallel. In summary, a major limitation of existing studies is that they explore plane level parallelism passively, making it difficult to satisfy the access restrictions all the time. In particular, it is challenging to construct *multi-plane command* after GC and/or wear leveling mis-align the write points in different planes.

In this paper, we propose SPD, an SSD *from plane to die* parallel optimization framework, to fully exploit the plane level parallelism of SSDs for performance improvement. 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.
- To further improve the write performance, we propose SPD+, which is designed to process host write operations blocked by die level GCs in parallel with GC induced valid page movements. Therefore, host write operations are able to be completed with lesser waiting time.
- We evaluate the proposed approach using a significantly extended SSDSim [10] and compare it to the state-of-the-arts. The experimental results show that proposed approach is able to significantly improve write performance of SSDs without read performance impact.

The rest of this paper is organized as follows: In Section 2, the background is presented. In Section 3, the problem statement is presented. In Section 4, the SPD framework is presented. In Sections 5 and 6, the experiment setup and evaluations are presented. In Section 7, related works are discussed. Finally, the work is concluded in Section 8.

2 Background

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

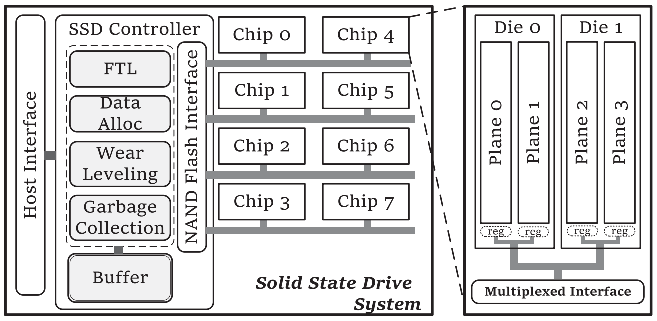


Fig. 1. The organization of SSDs.

2.1 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. Fig. 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).

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 `data allocation` is to determine the allocations of channel, chip, die and plane for write operations. The `wear leveling` is to distribute written data evenly to flash pages for prolonging the SSD lifetime [23], [24]. 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 reclaim invalid pages, `GC` is activated while the number of free pages drops below a predefined threshold.

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 [25], [26], [27], [28].

2.2 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 [9], [10], [12], [18], [29]. 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

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

interleaving command [9], [10]. 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.

2.3 Garbage Collection

Within flash memory, pages can not be updated in place [7]. In order to solve this issue, data is always updated out of place by programming updated data in another block and invalidating original version. Invalid pages can not be reused until they are erased. With the increasing of invalid pages' number and reduction of free pages' number, garbage collection (GC) is triggered for reclaiming invalid pages [1], [13]. The process of GC can be described as follows: First, a victim block is selected; Second, valid pages in the victim block are read and wrote to free pages in other blocks; Third, the victim block is erased. During this process, valid page movement is performed page by page, which is able to introduce significant time cost of SSD system while the incoming host requests are blocked and delayed [13].

To solve such a problem, a simple and effective approach, termed greedy GC, has been proposed and widely used to reduce the time cost of valid page movement [22], [30]. Greedy GC is designed to minimize the cost of valid page movement by selecting block with minimum number of valid pages. Thus, the total GC induced time cost can be minimized so that fewer host requests are blocked and delayed, increasing the performance of SSDs. In this work, greedy GC also is considered as a typical GC algorithm while other GC algorithms [31], [32], [33] also can be applied in proposed approach without loss of generality.

3 Problem Statement

In this section, we study the challenges in exploiting the plane level parallelism, which comes mainly from the restrictions of the *multi-plane command* [9], [18]. For clarity, we focus on write operations as they are much slower than read operations and thus have larger impact on the overall performance.

We next conduct a study on the operations to an SSD with each die consisting of two planes. Without losing generality, the non-GC operations that access the same die may be categorized to the following four cases.

`Case 1:` The operations are issued to one plane only (*Single Write*). Such write operations introduce unaligned write points across different planes;

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

`Case 3:` Two same type operations with unaligned in-plane addresses are issued to 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 two planes (*Parallel Writes*), which can be processed in parallel with the support of *multi-plane command*.

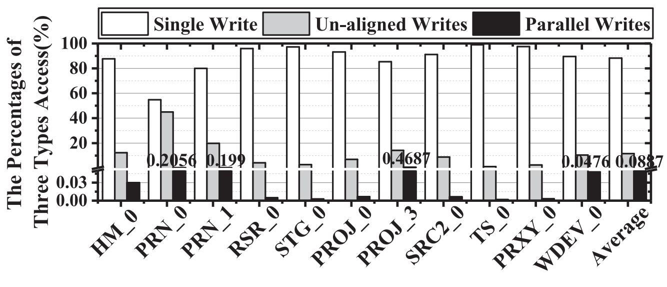


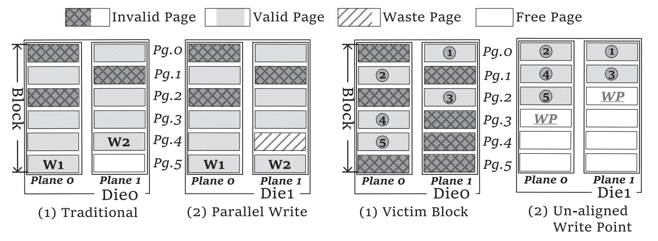
Fig. 2. The percentages of write operations in three cases.

Among these four cases, Case 2 can not be avoided due to the circuit restriction of *multi-plane command* while mixed types of operations being issued to different planes of the same die. Then, the numbers of operations falling in Case 1, Case 3 and Case 4 are collected and reported in Fig. 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.

To solve the issue of unaligned write points across planes, a naive solution 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 Fig. 3a. In Fig. 3a-(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 Fig. 3a-(2). *In this work, we strive to design a write construction scheme to align the write points in each die.*

Apart from host requests, internal SSD activities, e.g., GC, also introduce non-negligible performance impact from the unaligned write points across planes [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 planes at a time so that GC induced time cost can be overlapped [9]. To avoid significant parallel GC induced write amplification, ParaGC proposed to select a block containing most invalid pages in a plane first. Then, if the number of invalid pages in the paired block resided in the paired plane is large too, 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

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.



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

Fig. 3. The problems of unaligned write operations.

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 Fig. 3b, after moving valid pages in each plane in Fig. 3b-(1), new write points (*WP* in the figure) become unaligned, as shown in Fig. 3b-(2). Second, if only one block is occupied by GC, write points will be unaligned while this block is erased and switched as a free block 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 both of host requests and GC induced operations.

4 SPD: From Plane to Die Parallelism Exploration

4.1 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 points across multi-planes in each die at all time.

Basically, SPD takes the following strategies to achieve the objective, as shown in Fig. 4. The basic SPD design adds three new components — a die level write construction, a die level GC and a combination scheme. 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 movements. Note that for other activities, such as WL, they also can adopt the same design principle of GC. For simplicity, only GC is taken as an example in this paper due to its non-negligible performance impact on SSDs. The combination scheme is proposed to process write operations and GC related valid page movements in parallel so that write operations from host system can be processed with less waiting time.

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 pages, which does not introduce any read amplification. For die level GC, it is activated at the multiple planes of the same die at the same time. During the process of die level GC, all writes induced from the valid page movements also is processed in the unit of N page writes to maintain the aligned write points. After moving all valid pages to new free pages, erase operations are executed in parallel to reclaim victim blocks with same

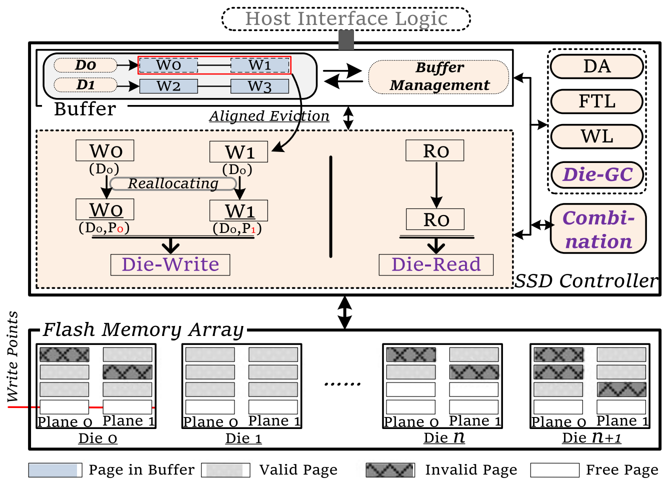


Fig. 4. The Overview of the *from plane to die* framework.

in-plane address. 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. Since GC process is time-consuming, host write operations blocked by `Die-GC` (called as access conflict in this paper) are able to be significantly delayed, causing write performance degradation. To minimize `Die-GC` induced access conflict, we combine `Die-Write` and `Die-GC` together for processing host write operations and GC induced valid page movements in parallel. Therefore, host write operations can be processed with lesser waiting time, bringing better write performance. We will elaborate the details of these three components in following sections.

4.2 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 round-robin plane allocation scheme within a die [11], which evenly distributes N dirty pages to different planes at each cycle. 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, which is simple and has been widely equipped in real SSD devices.

4.2.1 Buffer Supported Die-Write

Fig. 5 illustrates how the SSD buffer assisted `Die-Write` works. Fig. 5a 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

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

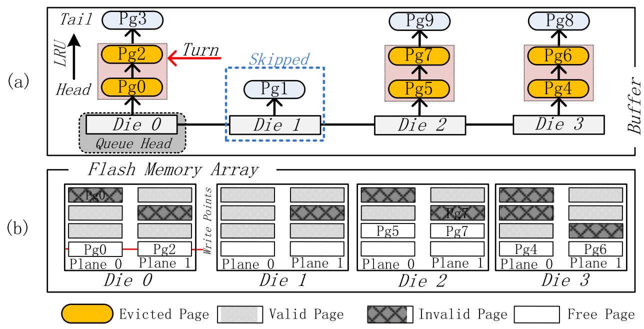


Fig. 5. Organization of write buffer and the die level write construction.

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 Fig. 5b, 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 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 dirty 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 by leveraging the parallel architecture of SSDs, 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 also 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.

4.2.2 Implementation and Analysis

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. 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

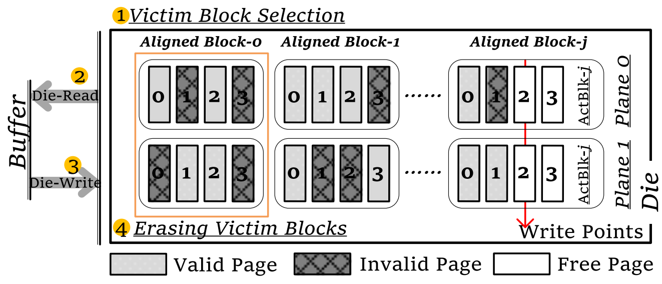


Fig. 6. The Process of Die-GC.

management where M is the number of dies in an SSD, and each die has N planes. For example, for a 512 GB SSD, with 32 dies, two planes in each die and 4 KB page size, the minimal size of buffer should be 256 KB, which can be satisfied by most existing SSD products.

In addition, to maintain such die queues, more space overhead is introduced. First, total 32 queue heads are required and each queue head points to the the LPN of a page. Each LPN requires 4 bytes and the total space cost of all queue heads is 128 bytes. Second, another pointer is set to locate current active die queue, where pages are going to be evicted in the next eviction innovation. Since there at most are 32 die queues, this pointers needs 5 bits space cost. Third, each die queue should set one counter to record the number of pages linked in this queue. In worst case, all pages in the buffer are linked in one die queue. That is, each counter requires 17 bits and total space cost of counters is 68 bytes. In summary, less than 197 bytes are needed to maintain these die queues.

Another issue that SPD needs to consider is the power interruption induced data loss, which is often mitigated by integrating a super capacitor [34], [35], [36], [37], [38].

4.3 Die Level GC

A GC process includes three steps: victim block selection [1], [14]; 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 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.

4.3.1 GC Process

Fig. 6 shows an example for `Die-GC`. Within each die, N stripped blocks from N planes — one from each plane and all the selected blocks share the same in-plane address, are grouped together as aligned block, which is set as the minimal granularity of `Die-GC`. That is, different to the traditional GC process, `Die-GC` should be modified while aligned block is taken as the minimal granularity. Totally, there exist four steps: First, we adopts the greedy based victim block selection [1], [13], where the aligned block with maximal number of invalid pages is selected. With this scheme, the time cost of valid page movement will be minimized. Such scheme can be realized without any modification. Since traditional GC

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

process scans the states (valid/invalid/free) of all pages in a block and finds a victim block which contains maximal number of invalid pages, proposed Die-GC is realized by summing the number of valid pages of blocks in each aligned block. Based on the summed number, victim aligned block is selected. Second, SPD uses Die-Read (in Section 4.2.1) to read the valid pages to the SSD buffer, where N page slots are required to store valid pages from victim blocks. 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. However, since each GC erases and reclaims two blocks after one invocation, the total number of erase operations during the whole lifetime of SSDs can be increased while the frequency of triggering GC can not be significantly reduced, causing lifetime degradation of SSDs. In the experiment, the impact on lifetime is going to be evaluated and presented.

For the example shown in Fig. 6, let us assume the two aligned blocks 0 from two planes 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 last Die-Write cannot be constructed due to the lack of one more valid page, causing the write points of different planes misaligned after GC while the last Die-Write is carried out with only one valid page inside. To address this issue, the last Die-Write operation is constructed by the remaining valid page in victim block and one dirty page in the write buffer (as discussed in Section 4.2).

4.3.2 *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. 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

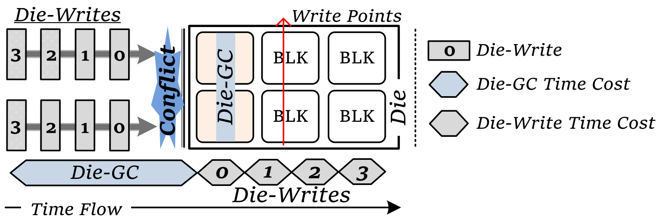


Fig. 7. Access Conflict between Die-Writes and Die-GC.

write buffer. For a typical SSD setting as presented in Section 4.2, the required buffer size for Die-GC is 256 KB for a 512 GB two-plane SSD and 512 KB for a 512 GB four-plane SSD at least. In summary, the storage requirement is modest for modern SSDs.

4.4 Combining Die Level Writes and GC

Both of Die-Write and Die-GC can be used to improve the write performance by fully exploiting the plane level parallelism. However, if there is a die being occupied by Die-GC, constructed Die-Writes towards this die will be delayed and wait for the completion of Die-GC. As shown in Fig. 7, there are four Die-Writes accessing current die, where a Die-GC is being processed in two blocks. In this case, these four Die-Writes have to be delayed until the completion of Die-GC. The bottom of Fig. 7 shows the time flow of Die-GC and Die-Writes. All these four Die-Writes suffer from a long waiting time caused by Die-GC. Such a GC induced conflict between Die-GC and Die-Write is regarded as access conflict in this work. In order to minimize the impact from access conflict between Die-Write and Die-GC, the most straightforward method is to process Die-Write with higher priority and postpone Die-GC. However, such a method may make Die-GC induced writes starve. Therefore, to avoid the starvation of Die-GC induced writes, we propose a combination scheme, which aims to reduce the waiting time of Die-Write without destroying the aligned write points.

4.4.1 *Combination Scheme*

As shown in top part of Fig. 8, there are eight valid pages in victim blocks (light blue boxes), and eight dirty page (gray boxes) generated write operations, which can be processed in parallel by constructing Die-GC and Die-Writes, respectively. Totally there are three steps (Step 0 to Step 2), which are processed in sequence order. For Die-Writes, which arrive during the process of Die-GC, they have to be delayed before the completion of Die-GC, causing write performance degradation.

To minimize the waiting time of Die-Writes, Die-GC is divided into two parts, including valid page movements and erase operations, where valid page movements are processed by constructing new Die-Writes. To solve this problem, in the bottom of Fig. 8, valid pages in victim blocks and dirty pages from buffer are combined to construct new Die-Writes. In this case, we assume there are three and five valid pages in two victim blocks, respectively. The whole combination scheme can be divided into three steps (Step 3 to Step 5 in Fig. 8). For Step 3, one victim block is selected to execute valid page movements while the paired

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

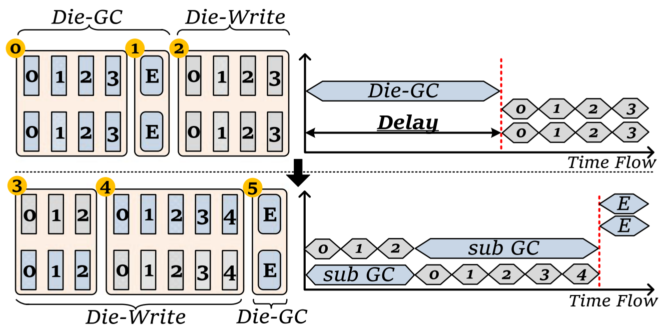


Fig. 8. Combining Die-Writes and Die-GC for Minimizing the Impact from Access Conflict (light blue box indicates GC related operations and gray box indicates dirty page generated write operations).

block is used to service write operations. Totally, three write operations from the buffer can be written in parallel with valid page movements of GC as three `Die-Writes`. Similarly, in Step 4, valid page movements in another victim block are realized while additional five dirty pages from the buffer are being wrote to the paired block. Since write points of all planes in the same die are aligned, the new `Die-Writes` are constructed and processed in parallel via accessing aligned write points. Lastly, in Step 5, two erase operations are executed in parallel as a `Die-GC` that does not contain valid page movements. For the time cost of this combination scheme, although the total time cost is the same as the original scheduling case (the top part in Fig. 8), the time cost of write operations from host system can be reduced while they are being processed in parallel with GC process.

4.4.2 Pseudocode Analysis

To explain more details, a combination scheme based algorithm is presented in Algorithm 1. Initially, we assume there are two planes in a die. Before evicting dirty pages to a corresponding die, the following algorithm is used to check whether these evicted dirty pages can be processed in parallel with valid page movements in GCs. If there is a `Die-GC` being processed in a die and some dirty pages are going to be evicted from buffer (Line 1), the combination scheme is activated. Otherwise, proposed `Die-GC` and `Die-Write` are processed as presented in above sections (Line 11-14). For the combination scheme, the numbers of valid pages and evicted pages should be larger than 0 (Line 2, 6) so that at least one valid page in victim block can be read out and constructed as a new `Die-Write` with an evicted dirty page from the buffer (Line 3, 7). After that, this new `Die-Write` containing valid page and dirty page is written back to the aligned write points (Line 4, 8). This process is repeatedly executed until all valid pages have been moved out or there is no more dirty pages being evicted from buffer. After that, `Die-GC` is resumed (Line 10). If there still exist some valid pages in victim blocks, paired valid pages are read out and written back in parallel based on the design of `Die-GC`. Otherwise, two erase operations are processed in parallel, such as the Step 5 in Fig. 8.

4.4.3 Implementation and Analysis

To implement proposed combination scheme, the `Die-GC` required buffer space is used to maintain one dirty page in the buffer, and then another valid page from victim block is transferred into this buffer space (two page space is required while a die contains two planes), where two pages can be constructed as a `Die-Writes`. Based on the implementation of `Die-Write` and `Die-GC`, proposed combination scheme can be realized without introducing additional implementation overhead.

Algorithm 1. Optimizing Access Conflict between Die-GC and Die-Writes

Input:
Assume that there are two victim blocks in a die, Blk_0 and Blk_1;
Die_GC: indicates a die is occupied by Die-GC;
Die_Write: indicates a write operation distributed to a die with Die-GC;
$Blk1\_VP$ and $Blk2\_VP$: indicates the number of valid pages in two blocks.
Output:

```
1: if $Die\_GC \neq NULL$ and $Die\_Write \neq NULL$ then
2:   while $Blk0\_VP \neq 0$ and $Die\_Write \neq NULL$ do
3:     $DieWrite\_Generation(Valid\ Page, Dirty\ Page)$;
4:     $Processing\_New\_DieWrite$;
5:   end while
6:   while $Blk1\_VP \neq 0$ and $Die\_Write \neq NULL$ do
7:     $DieWrite\_Generation(Valid\ Page, Dirty\ Page)$;
8:     $Processing\_New\_DieWrite$;
9:   end while
10:  $Processing\_DieGC$;
11: else if $Die\_GC \neq NULL$ then
12:   $Processing\_DieGC$;
13: else if $Die\_Write \neq NULL$ then
14:   $Processing\_DieWrite$;
15: end if
```

5 Experiment Setup

5.1 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 [3], [10]. The threshold value for GC activation is set to 7 percent [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 percent to trigger GC immediately, of which 80 percent 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 percent, 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 1.

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 [25], [40], which helps to prevent setting a large buffer from generating biased results in evaluation.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Parameters of the Simulated SSD [9], [39]

SSD Configuration	512 GB;16 Channels; 8 Chips/Channel; 1 Die/Chip; 2 Planes/Die;2048 Blocks/Plane; 256 Pages/Block; 4 KB 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.

The default data organization of die lists in the buffer is designed based on the scheme of the Element-Level Parallelism Optimization (EPO) [41]. 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.

5.2 Evaluated Workloads

The workloads studied in this work include a subset of MSR Cambridge Workloads from servers [42]. These workloads are widely used in previous works for studying SSD performance [9], [14], [18], [43]. The characteristics of workloads are presented in Table 2. 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.

5.3 Evaluated Schemes:

Seven 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 [41]. 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 [18], [44];

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*;

SuperPage. This scheme is implemented based on [17], [45], which groups pages into one super page that is set as the smallest access granularity of flash memory. Such a large-granularity accessing approach can fully exploit plane level parallelism by writing pages to all planes in a die at any time. Differing from our proposed work, if there exist

TABLE 2
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).

an update operation, the paired data in other planes should be read out first if they are valid. Later, the read data and updated data are constructed as one new super page, and then are written to the die 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`.

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

6 Experiment Results and Analysis

In this section, basic 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. Then, proposed SPD+ is evaluated and compared with basic SPD scheme to identify its effectiveness. Since SPD+ is designed to solve the access conflict between `Die-Write` and `Die-GC`, there is only one scenario with triggering GC. Finally, the impact of different buffer sizes and results on SSD with 4 planes per die are presented.

6.1 Experiment Results Without GC

1) Write Latency Evaluation: Fig. 9 shows the results of write latency for the six 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 percent

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

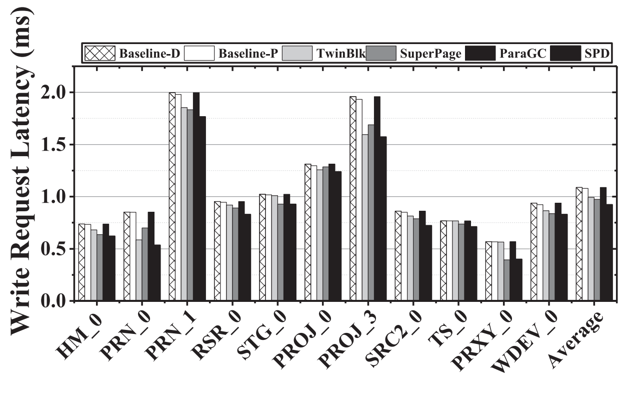


Fig. 9. Write latency reduction.

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 Fig. 10, 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.

To obtain more details, we compare SPD with other three schemes, Baseline-P, TwinBlk and SuperPage. Two observations can be concluded from the results: *First*, compared with these three 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 percent. TwinBlk 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 percent write latency reduction compared with Baseline-D. As shown in Fig. 10, 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. SuperPage also can fully exploit plane level parallelism at any time by constructing super page on a die. However, such a super page would introduce write amplification and cause write traffic. Before constructing an update operation related super page, read operations are required to read out all valid pages resided in original super page. Therefore, compared with Baseline-D, although SuperPage can achieve 10.4 percent write latency reduction, SPD still outperforms SuperPage at the performance improvement.

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 Fig. 10, 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.

TABLE 3
Read Latency Results Without GC

	Baseline-D	Baseline-P	TwinBlk	SuperPage	ParaGC	SPD
Reduction	0	0.049%	0.011%	0.304%	0%	0.096%

For read latency, the average read latency improvement compared with Baseline-D is presented in Table 3. The results show that read latency is similar among the six schemes. The key reasons are from two aspects: first, read requests of all evaluated schemes are processed with highest priority [3], [31], [46], [47]; second, `Die-Read` is designed to only read requested pages, which are barely located in the same in-plane addresses. In conclusion, the proposed `Die-Read` is same to that of normal read operations without introducing read amplification. Note that, in SuperPage, read operations always are executed with the support of *multi-plane command*, but only required data is transferred out for time cost saving.

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. Fig. 11 shows the plane utilization (Bars) and the maximal number of planes being accessed in parallel (Dots +Line) for the six schemes. The results have a matching pattern with the write performance improvement in Fig. 9. 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 34.5 percent 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. Similarly, SuperPage also can achieve the maximal plane level parallelism, whose plane utilization is averagely increased by 32.9 percent compared with Baseline-D. However, for Baseline-D, Baseline-P and TwinBlk, there still

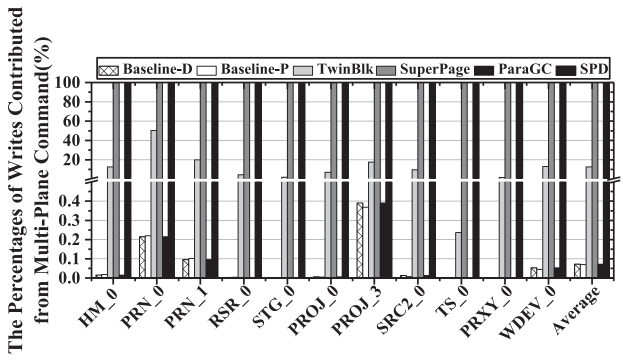


Fig. 10. Percentages of write operations processed by *multi-plane command*.

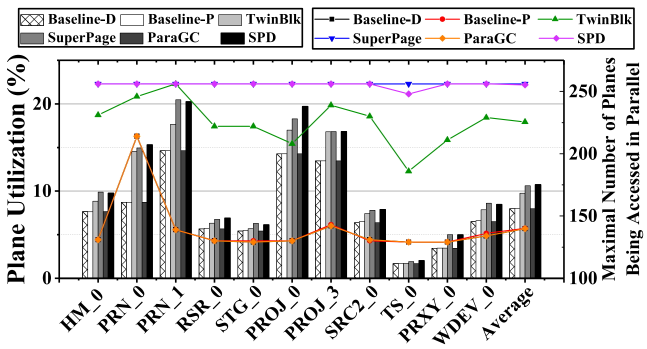


Fig. 11. The plane utilization and maximal number of planes being accessed in parallel.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

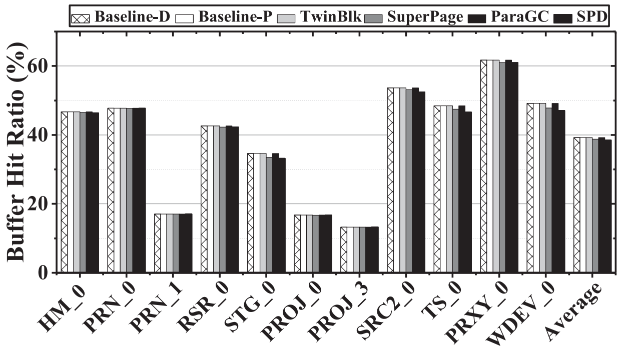


Fig. 12. The buffer hit ratios of evaluated schemes.

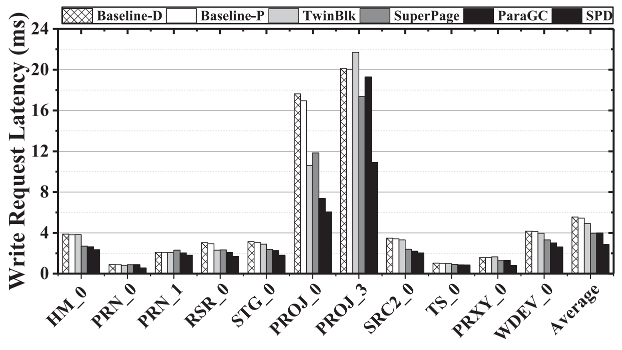


Fig. 13. The write latencies of evaluated schemes.

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.

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. Fig. 12 presents the results of buffer hit ratios for the six 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 percent, 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.

6.2 Experiment Results With GC

Fig. 13 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, 28.19, and 28.58 percent compared with Baseline-D, Baseline-P, TwinBlk, SuperPage, and ParaGC, on average. The significant improvement comes from three aspects: *First*, SPD constructs aligned write access to reduce write latency, which has been verified in Section 6.1. *Second*, the GC cost is further reduced through moving all valid pages with the support of `Die-Write`. *Third*, total GC count is noticeably reduced by reclaiming two planes at once time, and write amplification is avoided compared with SuperPage.

To understand more details, the total GC costs are presented in Fig. 14. The results show that first, TwinBlk generally has much higher cost than SuperPage and ParaGC. On average, compared with Baseline-D, total GC costs of SuperPage and ParaGC are reduced by 22.4 and 30.8 percent while TwinBlk only reduces the total GC cost by 6.9 percent. For SuperPage, it reclaims two blocks at once time to reduce GC cost, but the total GC count can be increased when update operation would read other valid pages for constructing a new super page, consuming free space more rapidly. More details about GC cost is presented in Section 6.3. For ParaGC, it activates GCs in paired planes only when the number of free pages in the other plane is smaller than 7 percent. 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`. Moreover, parallel access in SPD is constructed without introducing write amplification, avoiding triggering more GCs. As a result, the total GC cost is reduced by 36.4 percent, on average. In conclusion, SPD achieves the best write performance compared with all other related works.

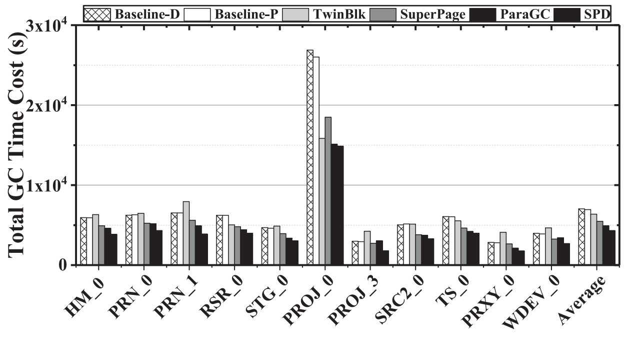


Fig. 14. Total GC cost of evaluated schemes.

TABLE 4
Read Latency Results With GC

	Baseline-D	Baseline-P	TwinBlk	SuperPage	ParaGC	SPD
Reduction	0	0.052%	-0.042%	4.173%	1.144%	1.203%

For read latency, results of read latency improvement with considering GC are presented in Table 4. Similarly, the read latency is similar among each scheme. The reason also comes from the highest priority of read request, which can be processed without delay [31], [46]. The results show that SPD has no impact to read access with significant write performance improvement.

6.3 GC Evaluation

In this part, `Die-GC` is evaluated. First, the average GC cost and the number of triggered GC in different schemes are

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

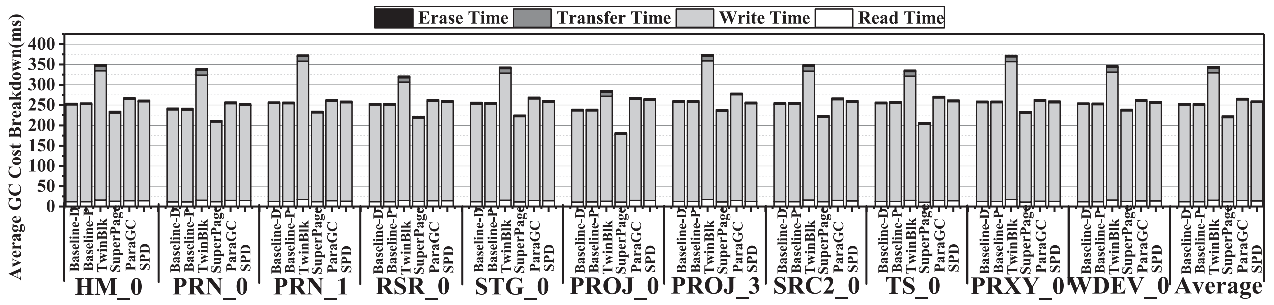


Fig. 15. Average GC cost breakdown of evaluated schemes.

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 Fig. 15. 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 three observations from the results: *First*, SPD has the less 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. *Third*, SuperPage can achieve the minimal GC cost compared with other schemes. The reason is that, SuperPage can aggressively invalidate pages in a block, reducing the total number of valid pages in a block. This is because each update operation can invalidate all paired pages for maintaining consistent states (valid or invalid) of all pages in a super page.

2) GC Count: Fig. 16 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 percent, 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 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. For SuperPage, since it can cause significant write amplification, total number of GCs is increased as well.

3) GC Induced Erases: Fig. 17 shows the number of erase operations for the six schemes. Since TwinBlk, SuperPage, 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 [48], [49]. However, the results show that the number of erase operations of `Die-GC` is much smaller than TwinBlk, SuperPage and ParaGC. For example, TwinBlk, in the worst case, introduces more than 102.2 percent erase operations for PRXY_0, compared with Baseline-D. ParaGC, introduces more than 65.8 percent erase operations compared with Baseline-D. Worst of all, SuperPage triggers 177.7 percent erase operations compared with Baseline-D. Compared with these three related works, SPD introduces fewer erase operations in most cases. On average, the number of erase operations is reduced by 13.43, 34.23 and

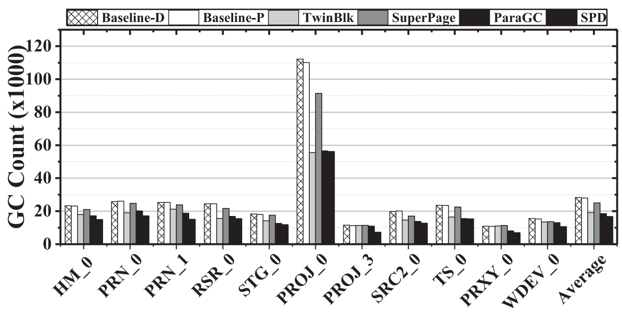


Fig. 16. The total number of triggered GC.

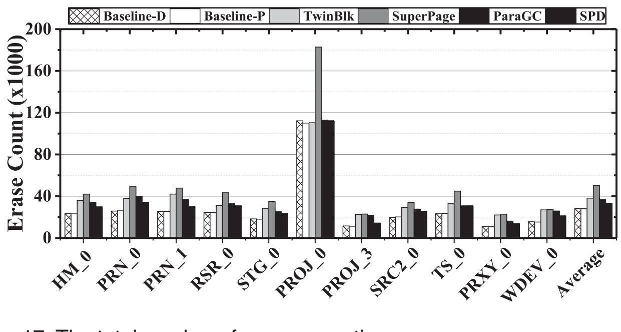


Fig. 17. The total number of erase operations.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

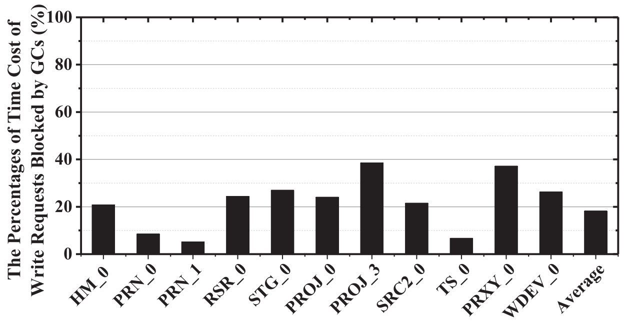


Fig. 18. The impact of access conflict between Die-write and Die-GC. The percentages of total time cost of write requests blocked by GCs are evaluated.

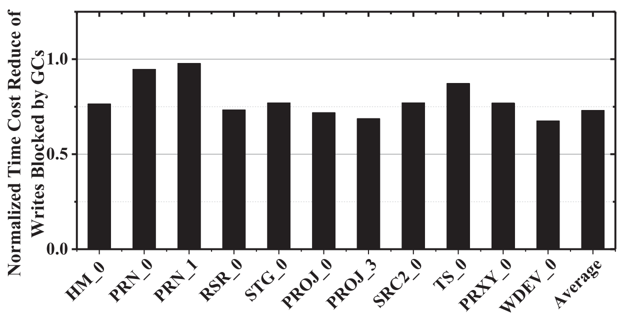


Fig. 20. Normalized time cost reduce of write requests blocked by GCs.

10.04 percent compared with TwinBlk, SuperPage 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. In addition, the write amplification is also avoided to reduce total GC count.

6.4 Experiment Results of SPD+

In order to evaluate the effectiveness of proposed SPD+, the impact of access conflict between Die-Write and Die-GC is measured first, which is the potential of SPD+. In SPD scheme, write requests in the constructed Die-Write are identified as blocked write requests while there is a Die-GC in the accessing die. To indicate the impact of access conflict, the percentages of blocked write requests' time cost are evaluated and presented in Fig. 18. On average, total time cost of write requests blocked by GCs accounts for 18.2 percent. The results presented in Fig. 18 show a matching pattern with Fig. 13, where the write performance improvements of PRN_0, PRN_1 and TS_0 are slight while the GC impacts on PRN_0, PRN_1 and TS_0 presented in Fig. 18 is weak. Take PRN_1 as an example. In Fig. 18, the total time cost of write requests blocked by GC of PRN_1 only accounts for 5.1 percent while the achieved write performance improvement of SPD presented in Fig. 13 is 13.7 percent, which is the minimal write performance improvement among all workloads.

By adopting the proposed combination scheme, which constructs Die-Write by grouping evicted page from the cache and valid page from victim block, the waiting time of evicted page generated write requests can be significantly reduced. The results of average write latency of SPD and SPD+ are evaluated and presented in Fig. 19. On average, SPD+ can further reduce the average write latency by 23.2 percent compared with basic SPD scheme. For these workloads little affected by access conflict (PRN_0, PRN_1 and TS_0), the achieved write latency decreases also are slight, accounting for 0.5, 0.2 and 5.1 percent, respectively. On the contrary, for PROJ_3, of which the total time cost of write requests is highly affected by GCs, the achieved write latency decrease reaches 28.7 percent.

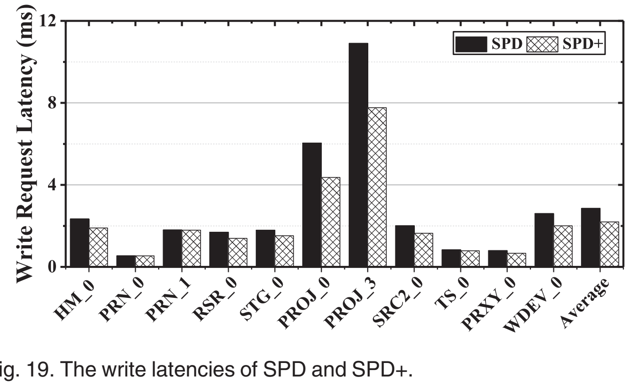


Fig. 19. The write latencies of SPD and SPD+.

To elaborate the reason of achieved significantly write performance improvement, the results of average time cost of write requests blocked by GCs are evaluated and presented in Fig. 20, where the results of SPD+ are normalized to SPD. For SPD+, while write requests can be processed in parallel with valid page movements during GC process, the waiting time of write requests can be reduced, causing the average time cost drop. In this figure, one can see that the average time cost of write requests blocked by GCs is reduced by 27.1 percent. Similarly, for PRN_1, which is least affected by GC induced access conflict, the reduced average time cost of write requests blocked by GCs also is minimal, only being reduced by 2.2 percent. On the other hand, for PROJ_3, the achieved average time cost reduction of write requests blocked by GCs is the maximal one, reaching 31.2 percent. In conclusion, the proposed combination scheme can be applied to significantly reduce waiting time of write requests blocked by GCs, making write performance to be further improved.

For GC evaluation, the total GC time cost, average GC time cost and GC count of SPD+ are similar to SPD, which are increased by 0.012, -0.015 and 0.027 percent, respectively. Since proposed SPD+ is designed to schedule write requests for reducing waiting time, the GC process will not be affected, and then the metrics of evaluated GC process are similar to SPD.

For read latency, SPD+ still executes read requests with highest priority, making the achieved read latency be similar to SPD. On average, the read latency of SPD+ is reduced by 0.33 percent compared with SPD.

6.5 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. Fig. 21 shows the results of the normalized write latencies of the six schemes by varying buffer size from 256 KB to 16 MB. 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

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

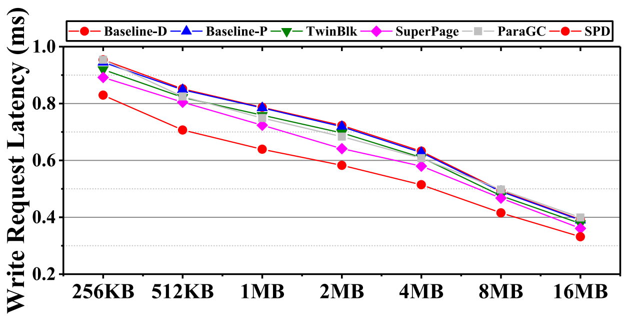


Fig. 21. Write latency with different buffer sizes.

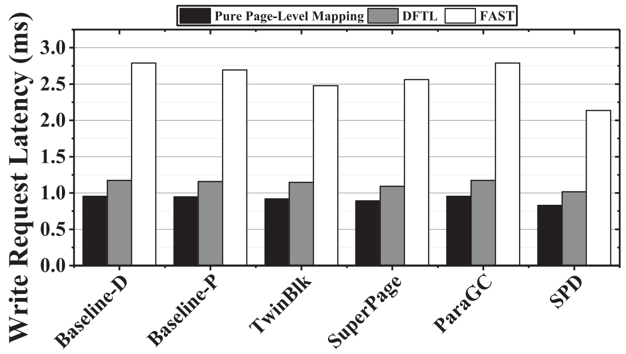


Fig. 23. Write latencies of pure page-level mapping, DFTL and FAST based evaluated schemes.

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.

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, and SPD are presented in Fig. 22, where GC is not triggered to evaluate the single influence from more planes per die. *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*, 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*. Therefore, more dirty pages can evicted and written back as `Die-Write` at the cost of one write operation when the number of planes in a die increases. On average, compared with Baseline-D, SPD achieves 43.9 percent write latency reduction, on average.

3) Various Mapping Schemes based SSDs: In this part, demand-based selective caching of page-level address mappings, termed DFTL [4], and hybrid mapping scheme are implemented to show the effectiveness of proposed SPD. Similar to the settings of above evaluations, GC is not triggered in this evaluation. For DFTL based SPD, mapping entries evicted from RAM buffer are organized as *Die-Write* as well. When mapping entries are supposed to be read from flash memory, *Die-Read* is generated to read required mapping entries as well. For hybrid mapping, FAST is implemented as an example [50]. In FAST based SPD, paired log blocks resided in paired planes are reserved. That is, write requests can be written into paired log blocks in the form of `Die-Write` as well. While merge operation is required, valid pages in selected paired log blocks and corresponding data blocks are read out and written to new paired data blocks by `Die-Write`.

Take RSR_0 as an example, Fig. 23 shows the write latencies of pure page-level mapping, DFTL and FAST based evaluated schemes. Herein, the write latencies of DFTL based schemes are worse than that of pure page-level mapping based schemes. For SPD, the write latency of DFTL based SPD is increased by 22.47 percent compared with pure page-level mapping based SPD. For FAST based schemes, since there are additional merge operations caused by FAST mapping, the write latencies of FAST based schemes are highly larger than that of other mappings based schemes respectively. But for FAST based SPD, it still can achieve substantial write performance improvement compared with other schemes equipped with FAST. This is because that, not only host write requests can be processed in parallel, but also each merge operation can reclaim two log blocks at the time cost of one merge operation. On average, compared with Baseline-D equipped with FAST mapping, FAST based SPD can reduce write latency by 23.43 percent.

4) Hot/Cold Separation based GC: To further reduce GC time cost, a widely adopted method is to distribute data to different blocks according to their hotness. Inside SSD, valid pages in victim block are cold enough, since only hot data are evicted from buffer and only relatively hot data in flash blocks tend to be invalidated. To further improve the efficiency of GC process, cold and relatively hot valid pages in victim blocks are written to different blocks according to their hotness as well in the revision [51], [52]. To realize SPD with considering hot/cold separation in GC process, each plane maintains two active blocks, cold block and hot block. To separate hot and cold data, the update count within a fixed time interval is recorded, which is set to 120 minutes according to [53]. Within 120 minutes, most data in evaluated workloads are updated. Except for valid pages from GC, host data evicted from buffer are considered as hot data and written into hot block as well.

The writ latencies of evaluated schemes with and without hot/cold data separation are collected and presented in Fig. 24. In this figure, two observations can be concluded.

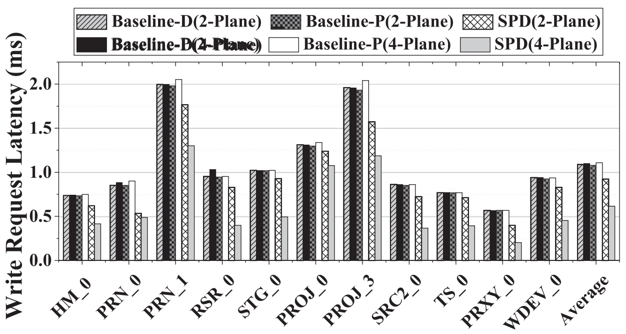


Fig. 22. Write latency with 4 planes per die.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

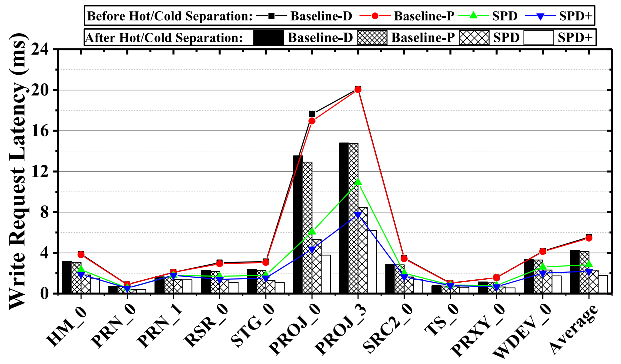


Fig. 24. Write latencies of evaluated schemes with and without hot/cold separation.

First, after applying hot/cold data separation, proposed SPD and SPD+ still can noticeably reduce write latency. On average, compared with Baseline-D, SPD+ can reduce write latency by 57.64 percent. Second, hot/cold data separation can significantly reduce write latency. Averagely, take Baseline-D as an example, scheme applied with hot/cold data separation can reduce write latency by 23.83 percent. The reasons come from two aspects: first, GC efficiency is improved while fewer valid pages are moved during GC process; second, total GC count can be reduced while cold pages are barely moved.

7 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.* [44] proposed to increase the potential of using *multi-plane command* through distributing requests belonging to different planes at one time. Similarly, Abdurrab *et al.* [29] 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. Caulfield and Seong *et al.* [17] and [45] proposed a new design principle for fully exploiting plane level parallelism, which groups all pages residing in the same in-chip location as a large logical page, termed super page. Such design can boost the performance and bandwidth of SSDs at the penalty of lifetime.

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. Also, SPD can minimize the impact on the lifetime of SSDs by grouping pages together more flexible.

(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], [54]; 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.* [54] 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 [12], [14], [55]. For example, Wu *et al.* [14] used cache to store requests conflicted by GC. Jung *et al.* [55] 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.

8 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. Three components are designed in the framework: die level write construction, die level GC and combination scheme. 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. In addition, the combination scheme is used to construct new die level write containing dirty page evicted from cache and valid page in victim block. The combination scheme can largely reduce the waiting time of write requests blocked by GCs, bringing write latency decrease. Experiment results show that SPD and SPD+ achieve significant write performance improvement and much smaller lifetime impact compared with state-of-the-art works.

Acknowledgments

This work is supported in part by NSFC 61772092 and 61872049, NSF 1910413,1617071,1725657 and 1718080, the

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 11219319), the Frontier Interdisciplinary Research Funds for the Central Universities (Project No. 2018CDQYJSJ0034). This work was carried out while Congming Gao was visiting the University of Pittsburgh on a CSC scholarship.

REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” in *Proc. Annu. Tech. Conf.*, 2008, pp. 57–70.

[2] 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,” in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, 2011, pp. 266–277.

[3] C. Gao, L. Shi, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, “Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives,” in *Proc. 30th Symp. Mass Storage Syst. Technol.*, 2014, pp. 1–11.

[4] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings,” in *Proc. 14th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2009, pp. 229–240.

[5] F. Chen, T. Luo, and X. Zhang, “CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives,” in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 77–90.

[6] M. Jung and M. T. Kandemir, “An evaluation of different page allocation strategies on high-speed SSDs,” in *Proc. 4th USENIX Conf. Hot Topics Storage File Syst.*, 2012, pp. 9–13.

[7] R. Micheloni, A. Marelli, and S. Commodaro, “NAND overview: From memory to systems,” *Inside NAND Flash Memories*. Berlin, Germany: Springer, 2010.

[8] ONFI, “Open NAND flash interface specification 4.1. website,” 2017. [Online]. Available: http://www.onfi.org/ /media/onfi/specs/onfi_4_1_gold.pdf?la=en

[9] N. Shahidi, M. Arjomand, M. Jung, M. T. Kandemir, C. R. Das, and A. Sivasubramaniam, “Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2016, pp. 561–572.

[10] 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,” in *Proc. Int. Conf. Supercomputing*, 2011, pp. 96–107.

[11] A. Tavakkol, P. Mehrvarzy, and H. S. -Azad,“TBM: Twin block management policy to enhance the utilization of plane-level parallelism in SSDs,” *IEEE Comput. Archit. Lett.*, vol. 15, no. 2, pp. 121–124, Jul.-Dec. 2016.

[12] W. Choi, M. Jung, M. Kandemir, and C. Das, “Parallelizing garbage collection with I/O to improve flash resource utilization,” in *Proc. 27th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2018, pp. 243–254.

[13] C. Gao *et al.*, “Exploiting chip idleness for minimizing garbage collection induced chip access conflict on SSDs,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, pp. 1–29, 2017.

[14] S. Wu, B. Mao, Y. Lin, and H. Jiang, “Improving performance for flash-based storage systems through GC-aware cache management,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2852–2865, Oct. 2017.

[15] S. Wu, Y. Lin, B. Mao, and H. Jiang, “GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs,” in *Proc. Int. Conf. Supercomputing*, 2016, pp. 1–12.

[16] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, “HIOS: A host interface I/O scheduler for solid state disks,” *ACM SIGARCH Comput. Archit. News*, vol. 42, pp. 289–300, 2014.

[17] A. M. Caulfield, L. M. Grupp, and S. Swanson, “Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications,” *ACM SIGARCH Comput. Archit. News*, vol. 37, pp. 217–228, 2009.

[18] C. Gao *et al.*, “Exploiting parallelism for access conflict minimization in flash-based solid state drives,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 168–181, Jan. 2018.

[19] S. Choudhuri and T. Givargis, “Performance improvement of block based NAND flash translation layer,” in *Proc. 5th IEEE/ACM Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2011, pp. 257–262.

[20] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, “A log buffer-based flash translation layer using fully-associative sector translation,” *ACM Trans. Embedded Comput. Syst.*, vol. 6, pp. 18–44, 2007.

[21] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, “System software for flash memory: A survey,” in *Proc. Int. Conf. Embedded Ubiquitous Comput.*, 2006, pp. 394–404.

[22] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” *ACM Comput. Surv.*, vol. 37, pp. 138–163, 2005.

[23] Y. Pan, G. Dong, and T. Zhang, “Exploiting memory device wear-out dynamics to improve NAND flash memory system performance,” in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 18–31.

[24] Y.-J. Woo and J.-S. Kim, “Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs,” in *Proc. Int. Conf. Embedded Softw.*, 2013, Art. no. 6.

[25] H. Kim and S. Ahn, “BPLRU: A buffer management scheme for improving random writes in flash storage,” in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 1–14.

[26] N. Megiddo and D. S. Modha, “ARC: A self-tuning, low overhead replacement cache,” in *Proc. 2nd USENIX Conf. File Storage Technol.*, 2003, pp. 115–130.

[27] L. Shi, J. Li, C. J. Xue, C. Yang, and X. Zhou, “ExLRU: A unified write buffer cache management for flash memory,” in *Proc. 9th ACM Int. Conf. Embedded Softw.*, 2011, pp. 339–348.

[28] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “DRAMSim2: A cycle accurate memory system simulator,” *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan.-Jun. 2011.

[29] A. R. Abdurrab, T. Xie, and W. Wang, “DLOOP: A flash translation layer exploiting plane-level parallelism,” in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 908–918.

[30] P. Desnoyers, “Analytic models of SSD write performance,” *ACM Trans. Storage*, vol. 10, 2014, Art. no. 8.

[31] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, “A semi-preemptive garbage collector for solid state drives,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2011, pp. 12–21.

[32] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang, “Using data clustering to improve cleaning performance for flash memory,” in, *Software: Practice and Experience*. Hoboken, NJ, USA: Wiley, 1999.

[33] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, “SFS: Random write considered harmful in solid state drives,” in *Proc. USENIX Conf. File Storage Technol.*, 2012, pp. 1–16.

[34] M. Huang, Y. Wang, L. Qiao, D. Liu, and Z. Shao, “SmartBackup: An efficient and reliable backup strategy for solid state drives with backup capacitors,” in *Proc. IEEE 17th Int. Conf. High Perform. Comput. Commun.*, 2015, pp. 746–751.

[35] J. Guo, J. Yang, Y. Zhang, and Y. Chen, “Low cost power failure protection for MLC NAND flash storage systems with PRAM/DRAM hybrid buffer,” in *Proc. Des. Autom. Test Eur. Conf. Exhibit.*, 2013, pp. 859–864.

[36] W. H. Kang, S. W. Lee, B. Moon, Y. S. Kee, and M. Oh, “Durable write cache in flash memory SSD for relational and NoSQL databases,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 529–540.

[37] C. Gao, S. Liang, Y. Di, Q. Li, C. Xue, and H.M. E. Sha, “An efficient cache management scheme for capacitor equipped solid state drives,” in *Proc. Great Lakes Symp. VLSI*, 2018, pp. 463–466.

[38] Micron, 7100 M.2 NVMe PCIe SSD, 2016. [Online]. Available: https://www.micron.com/ /media/documents/products/data-sheet/ssd/7100_m2_pcie_ssd.pdf

[39] C. GaoL. Shi, C. J. Xue, C. Ji, J. Yang, and Y. Zhang, “Parallel all the time: Plane level parallelism exploration for high performance SSDs,” in *Proc. 35th Symp. Mass Storage Syst. Technol.*, 2019, pp. 172–184.

[40] S.-W. Lee, B. Moon, and C. Park, “Advances in flash memory SSD technology for enterprise database applications,” in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 863–870.

[41] T. Xie and J. Koshia, “Boosting random write performance for enterprise flash storage systems,” in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol.*, 2011, pp. 1–10.

[42] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, “Migrating server storage to SSDs: Analysis of trade-offs,” in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 145–158.

[43] C. Gao *et al.*, “Constructing large, durable and fast SSD system via reprogramming 3D TLC flash memory,” in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 493–505.

[44] M. Jung, E. H. Wilson III, and M. Kandemir, “Physically addressed queueing (PAQ): Improving parallelism in solid state disks,” in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, 2012, pp. 404–415.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.

[45] Y. J. Seong *et al.*, "Hydra: A block-mapped parallel flash memory solid-state disk architecture," *IEEE Trans. Comput.*, vol. 59, no. 7, pp. 905–921, Jul. 2010.

[46] T. Brokhman, "ROW scheduling algorithm in block layer. Website," 2012. [Online]. Available: https://lwn.net/Articles/509829/

[47] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drivers," in *Proc. 7th ACM Int. Conf. Embedded Softw.*, 2009, pp. 295–304.

[48] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 3, pp. 837–863, 2004.

[49] L.-P. Chang and C.-Y. Wen, "Reducing asynchrony in channel garbage-collection for improving internal parallelism of multichannel solid-state disks," in *ACM Trans. Embedded Comput. Syst.*, vol. 13, 2014, Art. no. 63.

[50] S.-W. Lee, W.-K. Choi, and D.-J. Park, "FAST: An efficient flash translation layer for flash memory," in *Proc. Int. Conf. Embedded Ubiquitous Comput.*, 2006, pp. 879–887.

[51] H.-J. Kim and S.-G. Lee, "A new flash memory management for flash storage system," in *Proc. Int. Comput. Softw. Appl. Conf.*, 1999, pp. 284–289.

[52] O. Kwon, J. Lee, and K. Koh, "EF-greedy: A novel garbage collection policy for flash memory based embedded systems," in *Proc. Int. Conf. Comput. Sci.*, 2007, pp. 913–920.

[53] W. Choi, M. Arjomand, M. Jung, and M. Kandemir, "Exploiting data longevity for enhancing the lifetime of flash-based storage class memory," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, 2017, Art. no. 21.

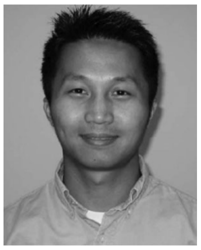
[54] D. Park and D. H. C. Du, "Hot data identification for flash-based storage systems using multiple bloom filters," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol.*, 2011, pp. 1–11.

[55] M. Jung, R. Prabhakar, and M. T. Kandemir, "Taking garbage collection overheads off the critical path in SSDs," in *Proc. 13th Int. Middleware Conf.*, 2012, pp. 164–186.

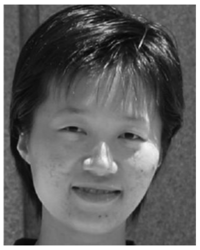
Congming Gao received the BS degree in computer science and technology from Chongqing University, China, in 2014. He is working toward the PhD degree in the College of Computer Science, Chongqing University. currently, he is a visiting scholar with the Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA. His research interests include flash memory, non-volatile memory and architecture optimizations.

Liang Shi received the BS degree in computer science from the Xi'an University of Post & Telecommunication, Xi'an, Shanxi, China, in 2008, and the PhD degree from the University of Science and Technology of China, Hefei, China, in 2013. He is currently a full-time professor with the School of Computer Science and Technology, East China Normal University. His research interests include flash memory, embedded systems, and emerging non-volatile memory technology.

Kai Liu received the PhD degree in computer science from the City University of Hong Kong in 2011. From December 2010 to May 2011, he was a visiting scholar with the Department of Computer Science, University of Virginia, USA. From 2011 to 2014, he was a postdoctoral fellow at Singapore Nanyang Technological University, City University of Hong Kong, and Hong Kong Baptist University. He is currently a professor with the College of Computer Science, Chongqing University, China. His research interests include Internet of Vehicles, Mobile Computing and Pervasive Computing.



Chun Jason Xue received the BS degree in computer science and engineering from the University of Texas at Arlington, in May 1997, and the MS and PhD degrees in computer science from the University of Texas at Dallas, in 2002 and 2007, respectively. He is currently an associate professor with the Department of Computer Science at the City University of Hong Kong. His research interests include memory and parallelism optimization for embedded systems, software/hardware co-design, real time systems, and computer security.



Jun Yang received the BS degree in computer science from Nanjing University, China, in 1995, the PhD degree in computer science from the University of Arizona, in 2002. She is a professor with the Electrical and Computer Engineering Department, University Pittsburgh, Pittsburgh, PA. She is the recipient of US NSF Career Award in 2008. She has won best paper awards from ICCD 2007 and ISLPED 2013. Her research interests include GPU architecture, secure processor architecture, emerging non-volatile memory technologies, performance, and reliability of memories.

Youtao Zhang (Member, IEEE) received the PhD degree in computer science from the University of Arizona, Tucson, AZ, in 2002. He is currently an associate professor of Computer Science, University of Pittsburgh, Pittsburgh, PA. His current research interests include computer architecture, program analysis, and optimization. He was the recipient of the U.S. National Science Foundation Career Award, in 2005. He is also the co-author of several papers that received paper awards.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: University of Pittsburgh. Downloaded on March 08,2021 at 19:44:58 UTC from IEEE Xplore. Restrictions apply.