Maximizing Bandwidth Management FTL Based on Read and Write Asymmetry of Flash Memory

Bihan Li, Wei Tong[‡], Jingning Liu, Dan Feng, Yazhi Feng, Junqing Qin, Peihao Li and Bo Liu[#]
Wuhan National Lab for Optoelectronics Huazhong University of Science and Technology, Wuhan, China
#:Hikstor Technology Co., LTD
‡Co-corresponding author: Wei Tong (Tongwei)@hust.edu.cn
Email: (bihan_li, tongwei, jnliu, dfeng, fengyazhi, qjq, clownier97)@hust.edu.cn
Email[#]:(liubo)@hikstor.com

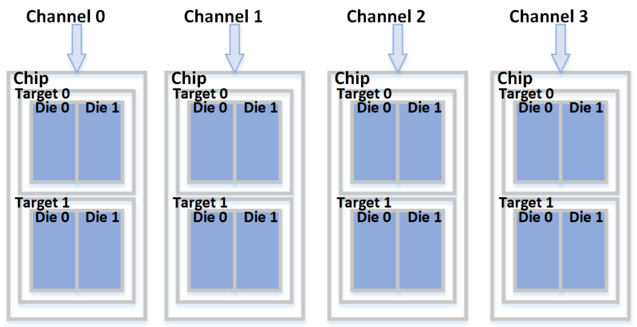
***Abstract*—With the high-density advantage, fewer 3D NAND chips are needed to build higher capacity embedded storage devices. However, this decrease in the number of chips means fewer parallel units, resulting in reduced channel bandwidth utilization and poor performance. By analyzing requests execution timing, we find that write and read operations need to focus on different problems to improve system performance because of read and write asymmetry. Promoting plane-level parallelism is more important for write. Reducing response time and providing a more balanced distribution of data is more crucial for read. Motivated by this observation, we propose a Maximize Bandwidth Management FTL called MBM. MBM includes a parallelism-enhanced Write Strategy (WS) and a parallelism-relaxed Read Strategy (RS). WS extends an active block for GC in each plane to enhance intra-chip parallelism. Additionally, it limits the channel's maximum executable number for superior request distribution. To guarantee response time, RS executes parallel read conditionally and improves read efficiency by rearranging data to a suitable location. Moreover, to reduce long tail latency, we also propose a Minimizing Chip consumption Strategy (MCS) and exploit a program/erase suspension. MCS helps provide enough idle chips for subsequent requests. Experiment results show the proposed MBM reduces the average response time by up to 66.8% and promotes I/O bandwidth to 3x compared to the baseline scheme. Specifically, between 99–99.999th percentiles, MBM significantly reduces the tail latency.**

***Index Terms*—3D NAND Flash, read and write asymmetry, plane-level parallelism, response time, channel bandwidth, tail latency**

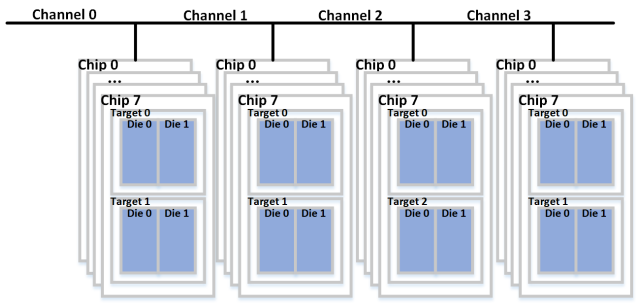
I. Introduction

In the past decade, flash memory has gradually become the most important storage medium [1]. It is widely used in embedded storage devices due to excellent performance [2], [3]. To achieve high performance, NAND-based SSDs provide multi-level parallelism (channel-level, chip-level, die-level, plane-level) [5], [8]. It helps to service more subrequests at a given period of time.

With the development of high-density technology such as the emergence of 3D NAND Flash, embedded devices meet the higher storage capacity requirements with fewer chips. For example, two generations of Samsung SSDs use different chips to build a 512 GB capacity SSD: (1) a Samsung 950 pro SSD [6] using 16 32GB chips and (2) a newer generation Samsung 960 pro SSD [7] using 8 64GB chips. 3D NAND Flash stacks 32 [9], 64 [10], 96 [11] or even 128 [12] layers of NAND cells vertically, a single chip capacity is significantly increased. Unfortunately, Chip capacity increase means the decrease in the number of chips, reducing parallel units of each channel and chip-level parallelism. To demonstrate the impact of chip capacity increase on intra-channel parallelism more clearly, we assume that the device maintains the same maximum bandwidth and the same number of channels, a 1TB flash device using 2D flash chips [30] (each flash chip contains four 64 Gb dies) require 32 chips. Using 3D NAND flash chips [31] (each flash chip contains four 512Gb dies of 512Gb), only four chips are needed as shown in Figure 1. Fewer requests can be processed in parallel at the given time, causing severe degradation in channel bandwidth [13].



(a) 3D NAND Chips.



(b) 2D NAND Chips.

Fig. 1. A comparison of the number of 3D and 2D flash memory chips in the channel.

Solutions proposed to address the above problem fall into two broad categories, request scheduler based [14] and parallelism promoting based [15] schemes. With request scheduler based schemes, the channel is kept busy by serving overcommitting requests. However, these schedulers neglect the strict restriction of plane-level parallelism. Request execution is not efficient. As an alternative to scheduler based solutions, parallelism promoting based schemes avoid these drawbacks. They reorder queued requests to increase the chance of parallel execution at the plane level. But executing requests in parallel unconditionally sacrifices response time especially for read. One parallel read may include many subrequests from different requests. Thus, the completion time of the previous request includes additional data transfer time of other requests.

Read and write operations of SSDs usually consist of a data transmission phase and a read or write phase. But read and write latency of flash memory is asymmetrical. Write latency is greater than data transfer time but read latency is less than it. By analyzing requests execution process, we find that to maximize channel bandwidth and promote SSD performance, read and write requests meet different challenges. For write, the most important is satisfying plane-level parallelism constraint and making full use of parallel units. However, indiscriminately forcing read operations to maximize parallelism can seriously affect read request response time instead. So, for read requests, reducing response time and providing a more balanced distribution of data is more crucial.

Motivated by the above observation, we propose Maximize Bandwidth Management (MBM). It utilizes read and write asymmetry to efficiently service requests. MBM includes a parallelism-enhanced Write Strategy (WS) and parallelism-relaxed Read Strategy (RS). WS extends an active block for GC in each plane to enhance intra-chip parallelism. Additionally, it limits the channel's maximum executable number for balanced request distribution. RS focuses on performing parallel read conditionally. Only when the subrequests come from a same request, we will execute read in parallel. To further improves the chance of parallel read, RS rearranges data in the read-intensive area averagely. Most latency has been optimized after applying the WS and RS. But large performance degradation still occurs when read operations are blocked. To solve the problem, we propose a Minimizing Chip consumption Strategy (MCS) and exploit a program and erase suspension mechanism additionally. They reduce the long tail latency by providing enough idle chips for subsequent write requests. The contributions of this paper include:

- Based on the observation that write and read operations need to focus on different problems to improve channel bandwidth and system performance, we propose MBM to perform write and read operations with different strategies. Moreover, to relieve performance degradation, we also provide MCS to optimize tail latency by exploiting the P/E (program/erase) suspension mechanism and leave more free chips to service subsequent requests through an optimal selection.
- We implement MBM techniques on SSDsim and evaluate it with comparison to two aggressive schemes. Experiment results show the proposed MBM reduces the average response time by up to 66.8% and promotes I/O bandwidth to 3x compared to the baseline scheme. In addition, with the number of chip decreasing, MBM is always guaranteed to provide stable performance. Moreover, at 99–99.999th percentiles, MBM significantly reduces the tail latency.

II. Background

A. SSD Parallelism

To handle multiple data accesses efficiently, SSD provides four levels of parallelism to maximize performance. They are channel-, chip-, die- and plane-level parallelism [5], [8]. Specifically, SSD includes multiple channels, each channel has an independent data bus. All channels can execute requests independently. According to the capacity demand, each channel connects a set of flash chips. These chips are organized in a parallel architecture to achieve high I/O performance by performing multiple flash operations simultaneously. It is called the chip-level parallelism. Each chip consists of multiple dies, which share the chip communication interface and independently execute flash operations. The plane-level parallelism is at the last level. However, it exhibits strict restrictions, i.e. 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).

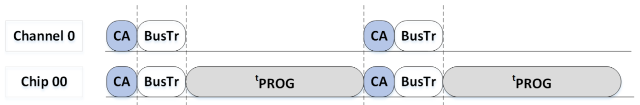
B. FTL

FTL is a key component of any flash software stack, which hides the complexities and constraints of the underlying flash medium like the erase before-write requirement and endurance limits. It receives the read/write requests and issues the operations to the NAND chips. FTL performs two important tasks: 1) Address mapping. Due to the out-of-place update, the logical page number (LPN) of the host requests must be translated into physical page number (PPN). For a write operation, FTL allocates a free physical page and stores the (LPN, PPN) pair in the mapping table for further reads. For read operations, translation is executed by searching the mapping table for LPN entry. 2) GC: Due to the out-of-place update, free flash pages are substantially consumed, resulting in numerous physical pages with invalid data. Consequently, FTL triggers a GC procedure to reclaim the invalid pages. GC procedure selects a victim block, moves its valid data to free pages, and finally performs an erase operation, which introduces additional read/program operations and performance issues.

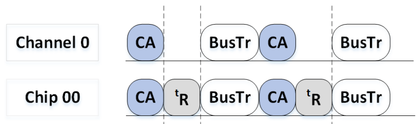
III. Motivation

A. Request Processing

An efficient FTL algorithm can make full use of parallelism and get better performance. When serving a request, FTL takes the request out of the queue and splits it into several subrequests. As shown in Figure 2, the execution of subrequests can be divided into three parts: command and address transfer



(a) The process of write operations.



(b) The process of read operations.

Fig. 2. The process of write and read operations.

(CA), data transfer and read/write. In the data transfer phase, data is transported from memory to the chip controller through the channel bus. When reading or writing, flash chips work independently and the bus will be released for other chips.

Two main factors that affect the response time of a request: 1) All sub-requests belonging to the same request are assigned to different channels and chips, resulting in different completion times. Subrequests completed early must wait for subrequests completed later, extending the overall response time of the request; 2) If the channel is busy executing data transmission of other requests, it needs to wait for the channel free, which increases the request response time.

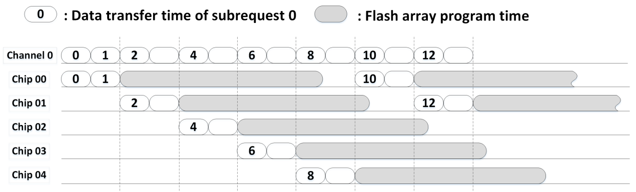
TABLE I
FLASH PARAMETERS FOR EXPERIMENT

Parameter	Value
Page Size	8KB
Page Number/Block	1024
Channel Number	4
Chip Number/Channel	2
Die Number/Chip	2
Plane Number/Die	2
Data Transfer Rate	10ns/8bit
Erase Time	15ms
Flash Array Program Time	1,300,000ns
Falsh Array Read Time	78,000ns
Command & Address Transfer Time	7*10ns
Data Transfer Mode	Synchronous mode

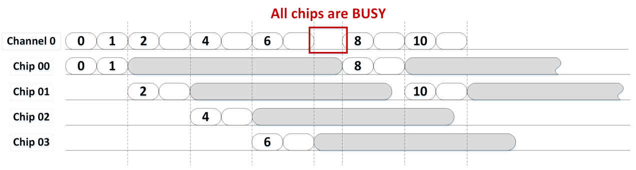
B. Read and Write Asymmetry

To find that performing read and write operations face different challenges for improving performance when the number of chips reduces, we analyze the timing and sequence of request execution. The parameters of the flash memory used in the analysis are those shown in Table I. Since the Command and Address (CA) transmission latency only accounts for 70 ns, it is much smaller than the read or write latency, the following figures will omit the CA latency without affecting the analysis result. First, we analyze three cases for write operation.

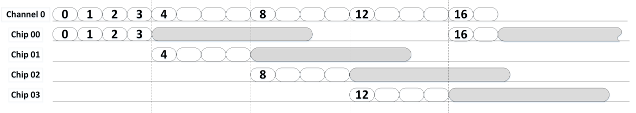
Case1: We assume that each channel contains five flash chips and maximum parallelism degree of these flash chips is four but the available parallelism degree is two now (Due to the different resource consumption rate, the intra-chip parallelism


0 : Data transfer time of subrequest 0
: Flash array program time
Channel 0
Chip 00
Chip 01
Chip 02
Chip 03
Chip 04
0
1
2
4
6
8
10
12

(a) the process of write operation with five chips per channel and 2 parallelism degree.



(b) the process of write operation with four chips per channel and 2 parallelism degree.



(c) the process of write operation with four chips per channel and 4 parallelism degree.

Fig. 3. the process of write operation with three different cases.

of each chip cannot always be fully utilized). The execution of write operations is shown in Figure 3(a). Channel bandwidth is well exploited.

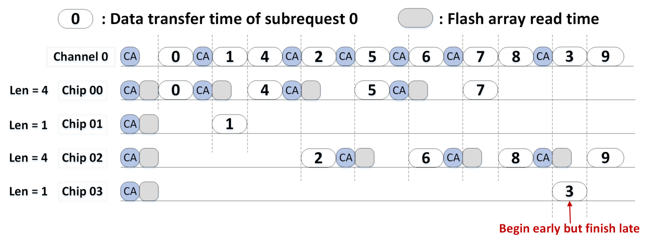
Case2: Keep the above assumptions remain unchanged. Only the number of chips decreases to four. As shown in Figure 3(b), channel bandwidth is wasted. that is to say, the reduction in the number of chips makes a negative impact on bandwidth utilization.

Case3: With the assumptions mentioned in Case2 constant, only the degree of available parallelism increases to four. It is apparent from Figure 3(c) that Channel0 is always busy at any time.

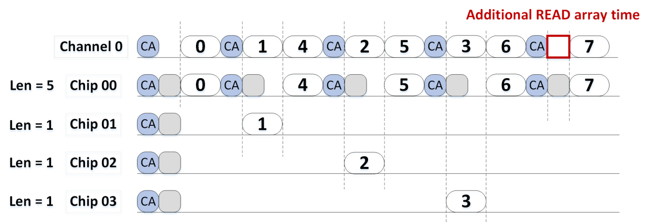
Three specific examples indicate that to provide better performance when executing write operations, promoting utilization of intra-chip parallelism is very important. It can compensate for the negative effect of the chip number reduction on channel bandwidth.

Next, we analyze read operations. It is worth noting that there are some differences between read and write operations: 1) read latency is much smaller than write latency, also than data transfer time, 2) the physical address of a read operation is fixed. If subrequests corresponding to the same chip can not meet parallel execution conditions can only be performed separately. The following cases show how channel bandwidth utilization changes when executing read operations with different settings.

Case4: Assume that one channel employs four chips and read subrequests do not satisfy parallel read constraints. There are four, one, four, and one subrequest locating in four chips respectively. Requests are executed as shown in Figure 4(a). We execute a simple scheduling for subreque3 to maximize



(a) The process of read operation with a simple schedule.



(b) The process of read operation with an extremely uneven distribution.

Fig. 4. The process of write operation with two different cases.

channel bandwidth as shown in Figure 4(a) without increasing parallelism. But after schedule, the response time of subrequest3 increases due to the long waiting time. Thus, we can draw the following conclusion: even if read operations can not always be executed in parallel, channel bandwidth waste will be alleviated in a simple way. So, for read operations, providing response time guarantees is more important and challenging than maximizing channel bandwidth when improving performance. However, there is often a conflict between maximizing bandwidth and guaranteeing response time. For balance, we try to maximize bandwidth when performing one request and guarantee response time between requests.

Case5: In the worst case, as shown in Figure 4(b), subrequests distributed in one chip are greater than the total subrequests distributed in other chips. There is no way to maximize bandwidth by scheduling in this extreme case. Moreover, it increases response time because the chance of parallel read decreases. Thus, a uniform distribution of data is crucial for read performance. It helps eliminate the worst response time.

To summarize, for improving channel bandwidth and system performance, write and read operations need to focus on different problems. Promoting plane-level parallelism is more important for write. Reducing response time and providing a more balanced distribution of data is more crucial for read.

IV. The MBM Design

In this section, we present the specific design of MBM. MBM is incorporated into the Flash Translation Layer of the SSD. It consists of three optimal solutions: parallelism-enhanced Write Strategy (WS), parallelism-relaxed Read Strategy (RS) and Minimizing Chip consumption Strategy (MCS). WS is responsible for satisfying the plane-level parallelism constraint when executing write requests. It extends an additional block for garbage collection (GC). Moreover, WS limits the number of subrequests that a channel can execute at a

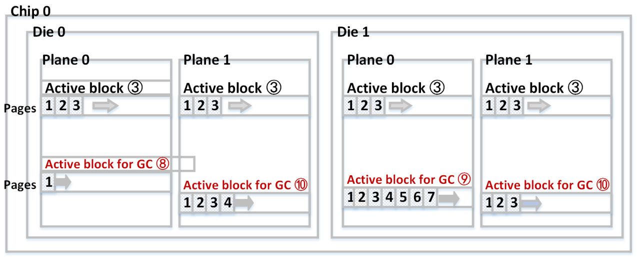


Fig. 5. Each plane additionally extends an active block for GC.

time by a simple algorithm. To guarantee response time, RS specifies necessary conditions to perform parallel read and improves read efficiency by a more uniform distribution of data. In addition, MCS optimizes tail latency by exploiting program/erase suspension and leaves more free chips to service subsequent requests through dynamic selections. For readability purposes, we first define a brief list of important terms commonly used in the following sections.

- **The parallelism of a flash chip**: the number of flash pages that can be simultaneously written in this chip with one program latency. For example, each flash chip itself is composed of 2 dies (called die0 and die1), each die consists of two planes (called plane0 and plane1). Only two planes of die0 can operate in parallel, and the other two planes of die1 did not adhere to the strict restriction of plane-level parallelism. Thus, with one program latency, 3 flash pages can be written at the same time, the parallelism of the chip is 3.
- **SSD transaction**: a set of commands that the SSD controller needs to initiate to perform a write operation. For example, writing three pages of a chip (parallelism is 3) requires initiating a multi-die command and a multi-plane command, transmitting three addresses and data, and occupying the channel bus before the SSD transaction completes.

A. Parallelism-Enhanced Write Strategy

Parallelism enhancement means each SSD transaction should finish more subrequests and utilize all available parallel units. So, the channel will always be busy according to the analysis in section III-B. However, space allocation and garbage collection greatly impact the existence of queued I/O operations mapped onto different planes of a die and, at the same time, accessing identical addresses in these planes. For instance, less queued write operations satisfy the plane-level addressing constraint if memory addresses of the neighboring planes are asymmetrically assigned and invalidated memory locations are recycled without any address consideration. It destroys plane-level parallelism. Available parallel units in each chip gradually decrease. Channel bandwidth will be wasted as shown in case2 of section III-B. To exploit plane-level parallelism more efficiently and alleviate its inherent limitations, we propose that each plane additionally extends an active block for garbage collection (GC), as shown in Figure

5. The last written page in the active block can keep the same by using a round robin assignment strategy for page allocation. When executing GC to reclaim space, valid data is migrated to a dedicated active block instead of the block being written. Thus, available free page addresses of all planes in the chip will keep consistent. To ensure there is enough room to service host requests, GC process will be triggered separately according to the available space in each plane. WS does not increase the area overhead. Data migration of the GC process always needs to consume new physical pages. The previous solutions migrate valid data to the open block when executing GC. WS extends an additional block for migrated data of GC instead of placing it in the active block. Therefore, the space consumption is the same.

Besides, to reduce the impact of a too long subrequest response time on the overall request response time, write requests should be more evenly distributed across all channels. In addition, because read operations also transfer data and occupy the channel bus, the number of subrequests that the channel can execute needs consider read. To balance the distribution of subrequests in all channels, we propose Channel_Max_Subs. It refers to the total number of read and write subrequests that can be executed in each channel currently. It's calculation algorithm as shown in the Algorithm 1.

First, we count the total number of subrequests in each channel (Line 1-3). Then sort channels from smallest to the largest total number (Line 4). Gradually allocate subrequests to the channel according to the order as Line4. If subrequests allocation complete, set the Channel_Max_Subs for every channel (Line 5-9). If not, modify the number of remaining subrequests and continue the above steps (Line 10-12). When all Channel_Max_Subs reach the same, but there are still some subrequests in the queue, we distribute them into all channels on average and update the Channel_Max_Subs (Line 14-16).

The minimum of Channel_Max_Subs and chip parallelism degree determines the number of subrequests performed in an SSD transaction. For example, according to Channel_Max_Subs, an SSD transaction can only execute two subrequests. But the current chip parallelism degree is four, the transaction will take two subrequests. If the chip parallelism degree remains unchanged, the channel can have five more subrequests, the transaction will only take four subrequests.

Algorithm 1 Cnannel_Max_Subs().

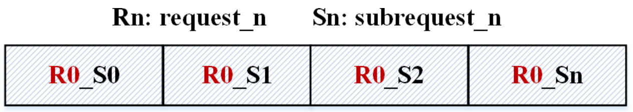
```
Input:
    Read requests queue length of the channel, channel_num[].read_len;
    Number of write subrequests that the channel has executed, channel_num[].execute_w_subs;
    Number of channel, channel_number;
    Requests number of SSD write requests queue, w_req;
Output:
    Subrequests number that channel can execute, channel_max_subs;
1:  for $i = 0$ to $channel\_number$ do
2:      Count the total number of subrequests per channel,
        channel_req_len[i] = channel_num[i].read_len + channel_num[i].execute_w_subs;
3:  end for
4:  Sort channel_req_len[channel_number] from small to large;
5:  for $i = 0$ to $channel\_number$ do
6:      if SSD number of write subrequests < subrequests number of Channel[i-1]- subrequests number of Channel[i] then
7:          channel_max_subs = (w_req%i) ? (w_req/i + 1 + channel_req_len[i-1]): (w_req/i + channel_req_len[i-1]);
8:          set the number of write requests to 0, w_req = 0;
9:          break;
10:     else
11:         update the number of write subrequests, w_req - = (channel_req_len[i] - channel_req_len[i-1]) * i;
12:     end if
13: end for
14: if subrequests number of write requests queue $\neq 0$ then
15:     average the number of remaining write subrequests to each channel and update the channel_max_subs;
16: end if
```

B. Parallelism-Relaxed Read Strategy

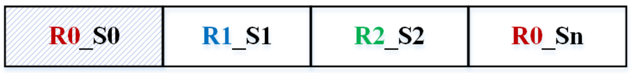
Parallelism-relaxed does not mean giving up channel bandwidth maximization. Instead, it indicates that we can maximize channel bandwidth without maximizing parallelism as WS when executing read requests. As the analysis shows in Section III-B, it is easy to guarantee channel bandwidth maximization by a simple schedule when performing read requests. And the channel will be kept busy when chips are occupied for reading data.

Therefore, using the same enhanced strategy for read is not necessary. Although executing a parallel read operation is efficient and could read more data once, the finish time of the first coming subrequest will be extended because of additional data transfer. Thus, executing read operations in parallel conditionally could achieve bandwidth maximization without sacrificing response time. Specifically, an shown in Figure 6 if the subrequests in a parallel read operation come from a same request, RS will perform the parallel read (see Figure 6(a)). Otherwise, RS responses to the first coming subrequest (see Figure 6(b)).

However, bandwidth could be wasted inevitably when subrequest distribution is extremely unbalanced. To further increase RS efficiency, we propose a more strict data layout for the read-intensive area. It increases the chance to perform parallel reads, promoting execution efficiency and read performance. A Read-intensive area is judged by space locality. Specifically, we set a threshold of a read request length. When a read request length is longer than the threshold, the corresponding logical area is read-intensive; otherwise, the area is not. If write requests are allocated to a read-intensive area, we will distribute them into all corresponding dies of the



(a) An example of parallel read.



(b) An example of responding to first coming subrequest .

Fig. 6. Two examples of RS.

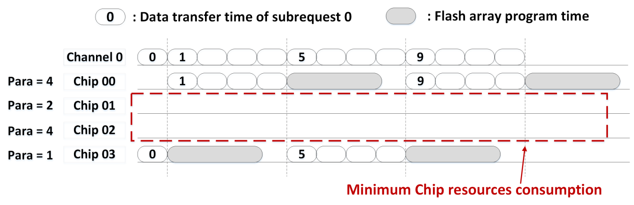

0 : Data transfer time of subrequest 0
: Flash array program time
Channel 0
Para = 4
Chip 00
Para = 2
Chip 01
Para = 4
Chip 02
Para = 1
Chip 03
0
1
5
9
Minimum Chip resources consumption

Fig. 7. Minimizing Chip consumption Strategy (MCS).

area averagely.

If data distribution satisfies one of the following conditions: 1) the transactions number of a request is greater than the theoretical number; 2) the subrequests of a request distributed in a channel are excessive. We will initiate a reallocation. An alternative method to reallocate is migrating redundant data from an overloaded channel to an underloaded channel.

C. Reduce Tail Latency

After WS and RS optimization, most high latency is alleviated. However, performance degradation problem still occurs. Through analysis, we find the reason for this phenomenon is as follows: 1) when a write request is an update operation and the page to be read is busy for programming, we need to wait for the program completed; 2) subrequests number is larger than chip parallelism, so we need to execute at least two write operations.

For the first reason, we exploit P/E suspend and resume [22]. It suspends the on-going program or erase operation (P/E) to service pending reads and resumes the suspended P/E afterward. For the second reason, we propose a Minimizing Chip consumption Strategy (MCS). When allocating pages for write requests, MCS first selects an idle chip with the least parallelism degree. This is because when the chip completes the current task, it will restore the maximum intra-chip parallelism and execute more subrequests next time. Then MCS selects a free chip with the maximal parallelism degree so that more data transfer time can cover the busy time of the working chip. Overall, the SSD can have more free chips for subsequent requests.

Figure 7 shows an example of MCS. Chip 03 with the least parallelism degree is selected for the first time, and the chip 00 with the maximum parallelism is selected for the second time. Chip 01 and Chip 02 keep the maximum parallelism and can respond to subsequent requests timely.

V. EXPERIMENTAL METHODOLOGIES

In this section, we present the experiment environment and apply a variety of workloads to evaluate MBM. For comparison, the other two existing FTLs [15], [14] are implemented as well. Simulations based on SSDsim [18] were implemented to evaluate the proposed MBM. The detailed flash parameters can be found in Table I. For the flash microarchitecture configuration, four levels of parallelism are supported. By testing IO bandwidth, average response time, and erase counts, we evaluate the overall performance of MBM. To investigate the impact of the proposed scheme on tail latency under the different workloads, we also test the I/O latencies at the 95th, 97th, 98th, 99th, and 99.5th percentiles. For convenience, we implement TBM [15] and Sprinkler [14] as a comparison. They are introduced as follows:

- *TBM* [15]: TBM symmetrically conducts usage and recycling of the flash block addresses on the planes of a die, thus enhancing the utilization of plane-level parallelism for reads, writes and erases.
- *Sprinkler* [14]: Sprinkler relaxes parallelism dependency by scheduling I/O requests based on internal resource layout rather than the order imposed by the device-level queue. Sprinkler improves flash-level parallelism and reduces the number of transactions (i.e., improves transactional locality) by over-committing flash memory requests to specific resources.

TRASE: We choose 6 representative real workloads presented by the storage networking industry association (SNIA) [19] and MSR Cambridge [20]. The important characteristics of our traces are given in Table II.

TABLE II
WORKLOADS ANALYSIS

Workloads	**Avg.Size (Read/Write) (KB)**	**Read/Write Ratio(%) (KB)**	**Annotation**
usrlvm0	40.91/10.28	40.42/59.58	User home directories
stg-lvm0	24.92/9.19	15.19/84.81	Webstaging
web-lvm0	29.99/8.59	29.88/70.12	Web/SQLserver
src1-lvm2	19.11/32.50	25.37/74.63	Source contrl
Financial	2.38/3.85	21.96/78.04	Financial institution
MSNFS	14.88/10.60	79.57/20.43	MSN File System

VI. EXPERIMENTAL RESULTS

A. Response Time

Average response time. Figure 8 plots the average response time, normalized to that of the TBM, driven by six workloads. The MBM scheme reduces the average response time of the baseline scheme by up to 66.8% with an average of 43.6%. The significant performance improvement comes from the following facts. First, WS fully exploits plane-level parallelism with the additional block for GC. Channel_Max_Subs ensure a more even distribution of subrequests across channels, avoiding excessively high response time. Second, when executing

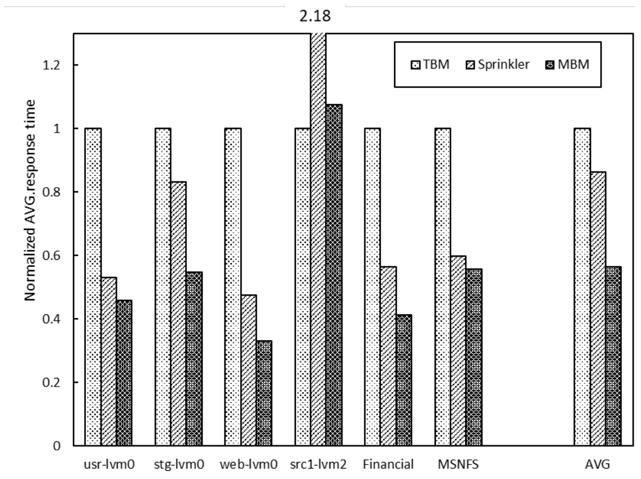


Fig. 8. The normalized average response times driven by the six workloads, normalized to that of the TBM.

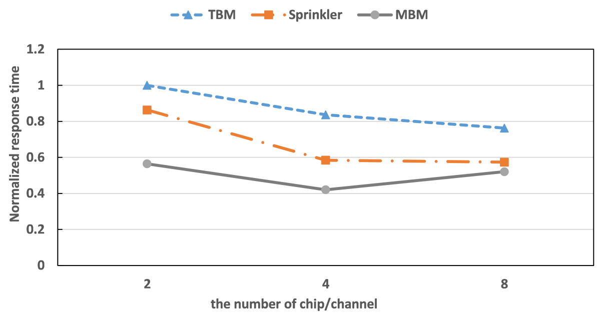


Fig. 9. The normalized average response times with different number of chips per channel, normalized to that of TBM with two chips per channel.

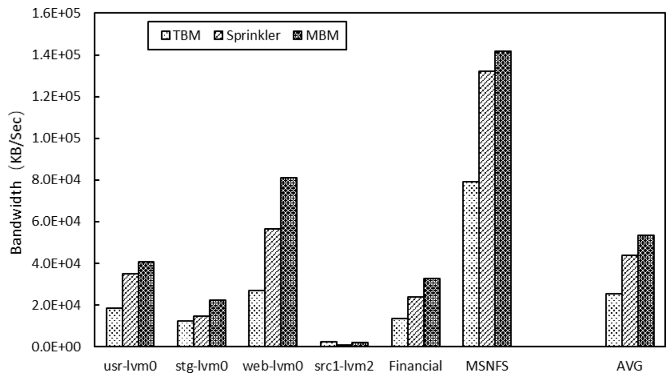


Fig. 10. The Bandwidth driven by the six workloads.

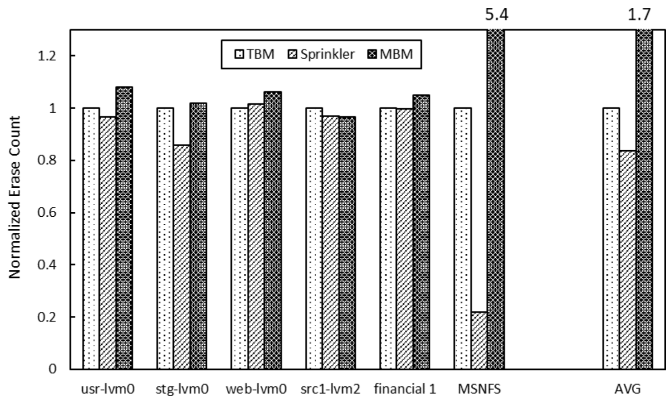


Fig. 11. The normalized erase count driven by the six workloads, normalized to that of the TBM.

read requests, RS promotes the chance of parallel read by a better data distribution strategy, improving the execution efficiency of read requests. Although TBM relieves the side-effects of out-of-place-update on the utilization of multi-plane read/write operations, it extends the response time of subrequests with unrestricted parallel reading. Sprinkler fully exploits channel-level and chip-level parallelism, but it neglects the strict restriction of the plane-level parallelism.

Unlike the other workloads, TBM outperforms Sprinkler and MBM in src1-lvm2. Sprinkler shows the worst performance with a 2.18x average response time compared to baseline. And that of the MBM increases by 7.5%. The result is related to the benchmark characteristics. Src1-lvm2 is a typical write-intensive benchmark. The average length of its write requests is the longest of the selected benchmarks. So, response time is mainly determined by program time. TBM decreases the average latency of src1-lvm2 with the best plane-parallelism because of twin block management. Sprinkler does not consider the plane-level parallelism constraint, MBM dose. But during RS, data remapping strategy for the read-intensive area makes it not always possible to keep the maximum parallelism degree when programming. So, the response time is slightly higher than that of the TBM. It is worth noting that if the decision method of the read-intensive area is further optimized, this problem will be alleviated and the average response time can be shortened.

Response time with different number of chip/channel. Moreover, to further investigate whether MBM can deliver and guarantee stable performance when the number of chips reduces, we present the average response time of different number of chips in Figure 9. The result is normalized to that of TBM with two chips per channel. We can see that MBM consistently and significantly outperforms the baseline system in different configurations. First, MBM reduces the average response time of the TBM by 43.6% and 31.6% when there are two and eight chips in a channel, respectively. Second, with chip number decreasing, the average response time of TBM and Sprinkler significantly increases. But MBM shows the opposite. It achieves almost the same performance when the chip number reduces from 8 to 2. In addition, the optimal result corresponds to the 4 chips per channel. The performance improvement of MBM comes from the fact that it fully utilizes intra-chip parallelism without sacrificing response time. However, Sprinkler does not consider the strict restriction of plane-level parallelism. TBM ignores the negative effect of the uneven distribution of requests. Thus, MBM outperforms Sprinkler and TBM no matter how the configuration changes.

B. Bandwidth

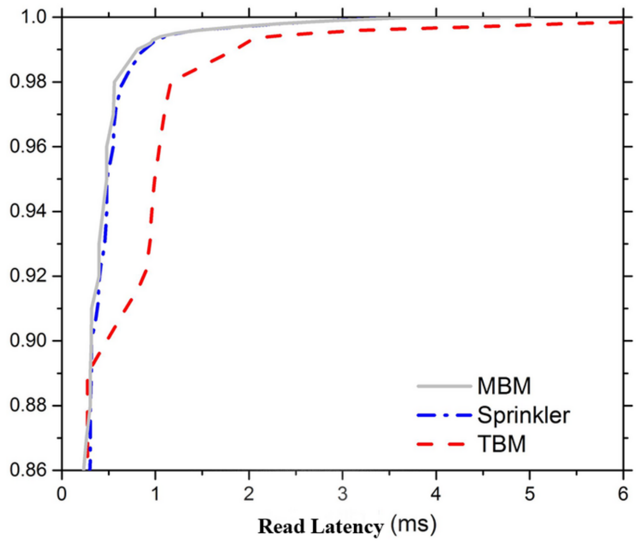
Bandwidth is also an important reflection of device performance. Figure 10 gives the bandwidth values, MBM increases the overall bandwidth by up to 3x and by 2.1x averagely. MBM is clearly the most efficient. The reasons behind MBM's superiority in the bandwidth are two-fold. First, by extends an additional block for GC, MBM sufficiently exploits plane-level parallelism compared to Sprinkler. Besides, when there are many subrequests in the queue, an SSD transaction will take out as many subrequests as possible according to Channel_Max_Subs. Moreover, Optimized data distribution in read-intensive areas also improves read parallel efficiency. Therefore, MBM shows the best bandwidth. Sprinkler also gets good improvement, this is because, in each operation, all parallel resources are utilized as much as possible. Although its plane-parallelism may be destroyed, the die- and chip-level parallelism can be better utilized. Consequently, neither TBM nor Sprinkler is productive and they are even worse than the baseline system without any optimizations. By contrast, MBM_FTL has a better balance between bandwidth and response time by exploiting read and write asymmetry, thus achieving the best results among all the schemes.

C. Erase Count

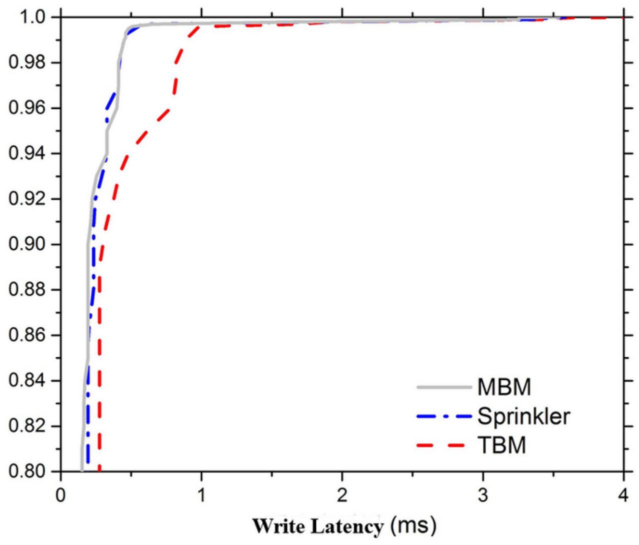
Erase Count plays an important role in measuring system overhead of proposed MBM and the other two schemes. Figure 11 presents the experiment result of the erase count. Except for MSNFS, the result shows the erase count of MBM slightly increases, but is almost the same as it of Sprinkler or TBM. Under MSNFS, the normalized result shows a large difference. This is because MSNFS has fewer writes than other benchmarks. And only dozens of GC operations are executed, while under other benchmarks are thousands. According to the experimental result of MSNFS, TBM, Sprinkler, and MBM perform GC operations 46, 15, and 247 times, respectively. As data volume increases such as the other five workloads, GC increment of MBM averages 3.5% and the maximum is 7.9%. In the read-intensive benchmarks, the increment is slightly more obvious.

D. Tail Latency

Tail latency represents the worst response time of the device. The lower tail latency indicates the much smaller performance degradations. Figure 12 and Figure 13 plots the read and write tail latency of three schemes when running MSNFS and Financial. We can see that at high percentiles, compared to Sprinkler and TBM, MBM consistently and significantly show better performance. Since tail latency is mainly caused by the blocking of a program or erase operations, the performance degradations are alleviated by P/E suspension. Moreover, the MCS strategy ensures that there are enough parallel units to respond to incoming write requests, which achieves the execution efficiency of write requests and reduces the blocking of read requests. The result of Sprinkler is similar to that of MBM under MSNFS. But in Financial, the read latency at the 98th percentile of Sprinkler is larger than that of MBM, and the write latency at the 94th percentile of Sprinkler also increases. TBM is the worst under different workloads. This reflects a serious problem with TBM. When performing GC, it forces the valid pages to be migrated into planes equally, causing the read locality corrupted. This affects both the read and write latency (update operations also need to be read first and then write), resulting in worse performance. Compared to Sprinkler, MBM completes the request in intensive case fast because RS and MCS help reduces the waiting time of program or erase operations. If the chip that needs to be read is busy, Sprinkler must wait for it completed (the program latency of a page is about 1,300us). But MBM can get the read data timely by suspending the program or erase operation.

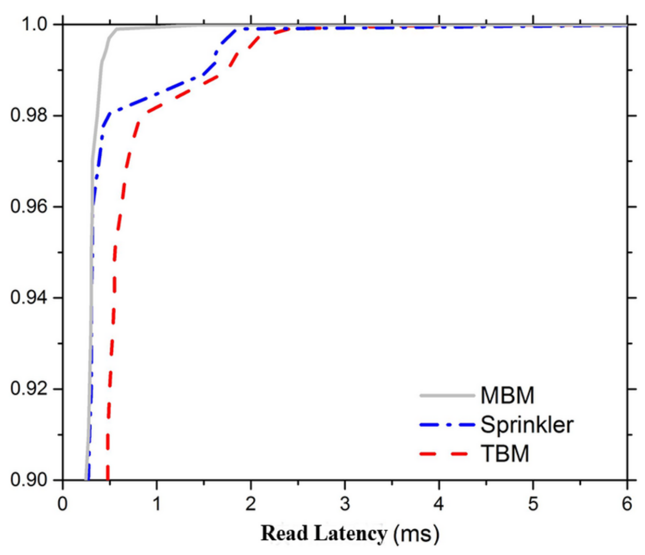


(a) MSNFS read tail latency

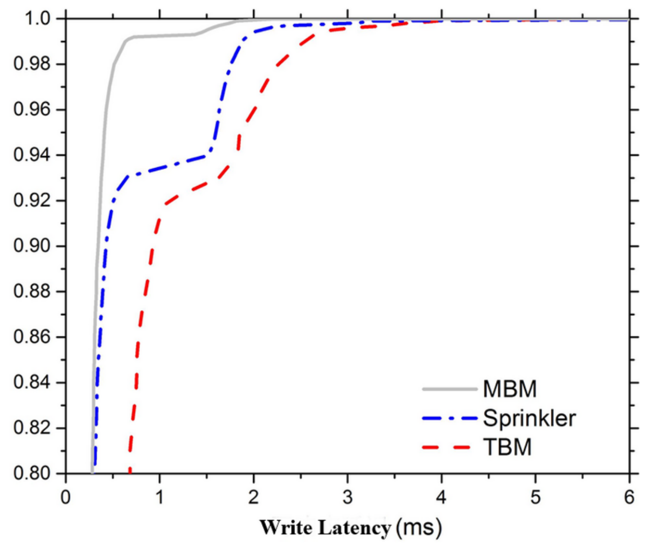


(b) MSNFS write tail latency

Fig. 12. Tail Latency:trace MSNFS



(a) Financial write tail latency



(b) Financial Write tail latency

Fig. 13. Tail Latency:trace Financial

VII. RELATED WORK

Several different methods have been proposed to improve performance by optimizing parallelism. Slacker [21] presents a slack-enabled re-ordering scheduler for subrequests issued to each flash chip. It estimates the slack of each incoming subrequest to a flash chip and allows them to jump ahead of existing subrequests with sufficient slack so as to not detrimentally impact their response times. SOML(Single-Operation-Multiple-Location) [13] can perform several small intra-chip read operations to different locations simultaneously, thereby mitigating the parallelism-related bottlenecks. P/E suspension scheme [22] suspends the on-going P/E to service pending reads and resumes the suspended P/E afterward. Having reads enjoy the highest priority. Moreover, Many recent studies generally use reordering and rescheduling of queued I/O operations to increase the chance of parallel execution at plane level [14], [23], [24]. Considering the impact of GC on device parallelism, GCPAR [25] presents a scheduling technique, I/O-parallelized GC, which leverages the idle planes during GC to service the blocked I/O requests. There have been several studies [26], [15] evaluate the priorities of four levels parallelism under various benchmarks, but none of them have optimized FTL management.

VIII. CONCLUSION

Because of the high storage capacity demands from the storage market, 3D NAND density keeps increasing. Unfortunately, high-density SSDs fail to achieve better intra-chip parallelism and channel bandwidth than their low-density counterparts. By analyzing the multi-level parallel request execution process in SSDs, we found that for improving channel bandwidth and system performance, write and read operations need to focus on different problems. Promoting plane-level parallelism is more important for write. Reducing response time and providing a more uniform distribution of data is more crucial for read. Motivated by this observation, we propose a Maximize Bandwidth Management FTL (MBM) which can perform parallelism-enhanced Write Strategy (WS) and parallelism-relaxed Read Strategy (RS) which are for write and read operations respectively. To mitigate serious performance degradation that cannot be completely avoided after applying the above strategy, a Minimizing Chip consumption Strategy (MCS) is also proposed correspondingly. Our experiments with various workloads indicate that, on average, the overall performance of MBM outperforms two state-of-the-art optimization strategies. The RS of MBM causes a little more erase counts, which are carefully controlled and evaluated. Optimizing the read-intensive area determination method in RS can further improve parallelism and reduce average response time. we will modify MBM to achieve better performance in the future work.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 61832007, Grant 61821003, Grant 61772222, and Grant U1705261, in part by the Fundamental Research Funds for the Central Universities No.2019kfyXMBZ037, and in part by National Science and Technology Major Project No. 2017ZX01032-101.

REFERENCES

[1] Y. Cai, S. Ghose, E. Haratsch, Y. Luo, and O. Mutlu. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. Proceedings of the IEEE, 105(9):1666–1704, 2017.

[2] Michael S, Nina T. Market Growth Rate Peaks After a Strong 2017; IDC Forecasts Semiconductor Revenue Growth of 7.7International Data Corporation, 2018.

[3] Jeff J, Michael S. IDC Forecasts Strong Growth for the Solid State Drive (SSD) Industry as It Transitions to 3D NAND Flash. International Data Corporation, 2017.

[4] Agrawal N, Prabhakaran V, Wobber T, et al. Design tradeoffs for SSD performance. in: Proceedings of USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, June 22 27,2008.

[5] Jung M, Kandemir M. An evaluation of different page allocation strategies on highspeed SSDs. in: Proceedings of Usenix Conference on Hot Topics in Storage & File Systems (HotStorage). Boston, MA, USA: USENIX, June 1314, 2012.

[6] Samsung. 2018. Samsung Pro 950 SSD. https://www.samsung.com/us/computing/memory-storage/solid-state-drives/ ssd-950-pro-nvme-512gb-mz-v5p512bw/. (Aug 2018).

[7] Samsung. 2018. Samsung Pro 960 SSD. http://www.samsung.com/ semiconductor/minisite/ssd/product/consumer/960pro/. (Aug 2018).

[8] Tavakkol A, Arjomand M, SarbaziAzad H. Unleashing the potentials of dynamism for page allocation strategies in SSDs. ACM SIGMETRICS Performance Evaluation Review, 2014, 42(1):551–552.

[9] Im J W, Jeong W P, et al. A 128Gb 3b/cell VNAND flash memory with 1Gb/s I/O rate. in: Proceedings of IEEE International Solid State Circuits Conference. San Francisco, CA, USA: IEEE, 2226 February, 2015, 1–3

[10] Lee S, Kim C, Kim M, et al. A 1Tb 4b/cell 64stackedWL 3D NAND flash memory with 12MB/s program throughput. in: Proceedings of2018 IEEE International SolidState Circuits Conference (ISSCC). San Francisco, CA, USA: IEEE, February 1115, 2018, 340–342

[11] Shibata N, Kanda K, Shimizu T, et al. 13.1 A 1.33 Tb 4bit/Cell 3D-Flash Memory on a 96WordLineLayer Technology. in: Proceedings of 2019 IEEE International SolidState Circuits Conference (ISSCC). San Francisco, CA, USA: IEEE, February 1721, 2019, 210–212

[12] Siau C, Kim K H, Lee S, et al. 13.5 A 512Gb 3bit/Cell 3D Flash Memory on 128WordlineLayer with 132MB/s Write Performance Featuring CircuitUnderArray Technology. in: Proceedings of2019 IEEE International SolidState Circuits Conference (ISSCC). San Francisco, CA, USA: IEEE, February 1721, 2019, 218–220

[13] Liu C Y, Kotra J B, Jung M, et al. SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs. in: Proceedings of Proceedings of the TwentyFourth International Conference on Architectural Support for Programming Languages and OperatingSystems.ACM,2019,955–969.

[14] Jung M, Kandemir M T. Sprinkler: Maximizing resource utilization in manychip solid state disks. in: Proceedings of 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2014,524–535.

[15] Tavakkol A, Arjomand M, SarbaziAzad H. Unleashing the potentials of dynamism for page allocation strategies inSSDs. in: Proceedings of ACM SIGMETRICS Performance Evaluation Review.ACM,2014,551–552.

[16] Shimpi A. Micron Announce 16nm 128Gb MLC NAND SSDs in 2014. http://www.anandtech.com/show/7147/micronannounces16nm-128gbMLCnandssdsin2014,2013.

[17] Tavakkol A, Mehrvarzy P, SarbaziAzad H. TBM: Twin Block Management Policy to Enhance the Utilization of PlaneLevel Parallelism in SSDs. IEEE Computer ArchitectureLetters,2016,15(2):1–1.

[18] Yang H, Hong J, Dan F, et al. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. in: Proceedings of International Conference on Super computing,2011.

[19] Narayanan D, et al. Write offloading: Practical power management for enterprise storage. TOS,2008,4(3):10 [139].

[20] Microsoft Enterprise Traces. http://iotta.snia.org/traces/list/BlockIO: University Science, 1989.

[21] Elyasi N, Arjomand M, Sivasubramaniam A, et al. Exploiting intra-request slack to improve SSD performance. ACM SIGARCH Computer Architecture News, 2017, 45(1):375–388.

[22] Wu G, Huang P, He X. Reducing SSD access latency via NAND flash program and erase suspension.in: Proceedings of Usenix Conference on File&Storage Technologies, 2014.

[23] M. Jung, et al., "Physically addressed queueing (PAQ): improving parallelism in solid state disks," in Proc. 39th Annu. Int. Symp. Comput. Archit., 2012, pp. 404–415.

[24] C. Park, et al., "Exploiting internal parallelism of flash-based SSDs," Comput. Archit. Lett., vol. 9, no. 1, pp. 9–12, Jan. 2010.

[25] Choi W,Jung M,Kandemir M, et al. Parallelizing garbage collection with I/O to improve flash resource utilization. in: Proceedings of Proceedings of the 27th International Symposium on HighPerformance Parallel and Distributed Computing. ACM, 2018, 243–254.

[26] Jung M,Kandemir M. An evaluation of different page allocation strategies on high speed SSDs. in: Proceedings of Usenix Conference on Hot Topics in Storage&File Systems, 2012.

[27] Yang Q, Ren J. ICASH: Intelligently coupled array of SSD and HDD. in: Proceedings of 2011 IEEE 17th International Symposium on High Performance Computer Architecture. Washington, DC, USA: IEEE, 12-16 February, 2011, 278–289.

[28] Park K T, Kang M, Kim D, et al. A zeroing celltocell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLCNAND flash memories. IEEE Journal of SolidState Circuits, 2008, 43(4):919–928.

[29] Li Y, Lee S, et al. 128Gb 3b/Cell NAND flash memory in 19nm technology with 18MB/s write rate and 400Mb/s toggle mode. in: Proceedings of2012 IEEE International SolidState Circuits Conference. San Francisco, USA: IEEE, 1923 February, 2012, 436–437.

[30] Micron Datasheet. Micron MT29FxxG08CxCAB 2D NAND Flash Memory. Boise, ID, USA, 2016. https://www.micron.com

[31] Micron Datasheet. TLC 384Gb to 1HTb Async/Sync NAND. Boise, ID, USA, 2017. https://www.micron.com