TBM: Twin Block Management Policy to Enhance the Utilization of Plane-Level Parallelism in SSDs

Arash Tavakkol, Pooyan Mehrvarzy, and Hamid Sarbazi-Azad

Abstract—The internal architecture of a SSD provides channel-, chip-, die- and plane-level parallelism levels, to concurrently perform multiple data accesses and compensate for the performance gap between a single flash chip and host interface. Although a good striping strategy can effectively exploit the first three levels, parallel I/O accesses at plane-level can be performed only for operations of the same types and page addresses. In this work, we propose the *Twin Block Management (TBM)* policy that 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. Evaluation results show that TBM improves IOPS and response time by up to 73 and 42 percent, respectively.

Index Terms—Flash memory, solid-state drive, plane-level parallelism, garbage collection

1 Introduction

IN NAND flash solid state drives (SSDs), in spite of the availability of high-performance NAND flash communication interfaces (up to 800 MB [1]), the maximum achievable I/O performance of a single flash chip (package) is restricted by the execution latency of flash operations. In particular, the flash write (program) operation is very slow and it even becomes slower by VLSI technology shrinking. Consequently, SSDs use a set of flash chips which are organized in a massively parallel architecture (see Fig. 1) to achieve high I/O performance through simultaneous execution of multiple flash operations. In this architecture, a multi-channel multi-way bus structure facilitates concurrent accesses to flash chips. Each flash chip itself is composed of a set of *dies* that share the chip communication interface and can independently execute flash operations. At the lowest level, there are multiple *planes* within a die that can operate in parallel. However, plane-level parallelism has a strict restriction that must be adhered to, i.e. same operations on the same flash memory addresses are required for simultaneous execution on the planes of a die.

Many recent studies were proposed to exploit plane-level parallelism more effectively and alleviate its inherent limitations. They generally use reordering and rescheduling of queued I/O operations to increase the chance of parallel execution at plane level [3], [5], [8]. However, the efficiency of these methods is highly sensitive to the behavior of the SSD flash management policy. Strictly speaking, an *out-of-place* update policy is used in SSD to reduce the negative impacts of the NAND flash *erase-before-write* property. To perform an update using this policy, the previous version of data is marked invalid and the new data is written into a free location. Therefore, a *logical-to-physical address mapping* scheme is used in conjunction with a *garbage collection (GC)* mechanism to manage data placement, consumption of the free memory locations, and recycling of the invalid locations. These mechanisms greatly impact the existence of queued I/O operations mapped onto different planes of a die and, at the same time, accessing identical addresses inside these planes. For instance, less queued write operations conform to 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. However, this critical influence of flash management mechanisms was not considered in previous proposals and hence their performance gain becomes negligible when random use of page addresses becomes more frequent in the long-term. In this work, we propose the *Twin Block Management (TBM)* policy for out-of-place update that is aware of plane-level addressing constraint. TBM defines new strategies for physical address assignment and GC execution in order to symmetrically conduct usage and recycling of the memory addresses on the planes of a die.

- *A. Tavakkol is with the HPCAN Lab, Computer Engineering Department, Sharif University of Technology, Tehran, Iran. E-mail: tavakkol@ce.sharif.edu.*
- *P. Mehrvarzy is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. E-mail: p.mehrvarzy@ipm.ir.*
- *H. Sarbazi-Azad is with the HPCAN Lab, Computer Engineering Department, Sharif University of Technology, and the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. E-mail: azad@ipm.ir.*

Manuscript received 8 Feb. 2015; revised 2 July 2015; accepted 13 July 2015. Date of publication 26 July 2015; date of current version 5 Jan. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2015.2461162

2 SSD Internals

Fig. 1 shows the internal architecture of a NAND flash SSD composed of four main components: 1) A host interface, that provides communication with the host system and performs I/O request queuing; 2) A controller, composed of a microprocessor and a DRAM memory, that executes a special management firmware called *Flash Translation Layer (FTL)*; 3) A flash controller that is a hardware driver to enable communication with flash chips; 4) Flash chips that provide the raw SSD storage capacity. As we mentioned previously, flash chips are organized in a hierarchy of four parallelism levels i.e. channel-, chip-, die- and plane-level. Multiple flash I/O operations can be simultaneously executed through striping over communication channels and pipelining between the set of flash chips connected to each channel. Furthermore, each die of a flash chip has its own command and address registers and hence execution of different operations can be interleaved between dies. At the lowest level, planes of a die share the same control logic. Therefore, same operations with same addresses must be available in the I/O queue for parallel plane (multi-plane) command execution. Access to the internal storage space of a plane is serial and read/write operations are performed at the unit of a *page* which typically includes 4 KB, 8 KB or a larger volume of data. Updating the content of a previously written flash page requires an *erase* operation. Due to its very slow execution, a flash erase is performed at the unit of a *block* composed of a set of 128, 256 or more pages. According to ONFI standard specification [1], multi-plane commands must access the same page offset within target blocks (referred as PAC). Besides, the flash control logic may also require the block addresses to be identical for multi-plane command execution (referred as BAC). As of 2015, all flash products require PAC but BAC is relaxed in many products.

In order to emulate the interface of a conventional block device (HDD), FTL performs following tasks: **1)** Management of the queued I/O requests and address mapping: host I/O requests are segmented into several page-size transactions each with a specific logical page address (LPA). Due to out-of-place update, LPAs must be translated into physical page addresses (PPAs). The translation procedure follows two different paths for write and read operations. For a write operation, FTL allocates a free physical page. First, a plane allocation function (PLAlloc) determines the address of target channel, flash chip, die and plane according to a predefined allocation strategy. Then, a block allocation function (BLAlloc) assigns a write-frontier within the selected plane. Inside write-frontier, pages are allocated sequentially, from first to last index, and thus PPA is determined and the (LPA, PPA) pair is stored in a mapping table for future reads. For read operations, translation is performed by searching the mapping table for LPA entry. **2)** GC: the out-of-place update policy quickly consumes free flash pages. Consequently, a GC procedure must be triggered by FTL to recycle the physical pages having invalid data. This procedure selects a victim block based on a predetermined policy, moves its valid pages into a new location, and finally triggers the execution of erase operation.

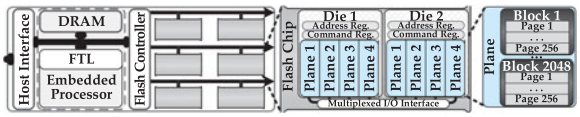


Fig. 1. The internal architecture of a NAND flash SSD.

3 Experimental Setup

In this work, we perform discrete-event trace driven simulation using SSDSim [2] that provides a detailed implementation of the architecture mentioned in Section 2. Furthermore, advanced management policies such as PAQ scheduling [5] and dynamic PLAlloc strategies [2] are implemented to better exploit internal parallelism. We use a diverse set of I/O traces from real enterprise and file-system benchmark applications. Table 1 gives details of the evaluation scenarios. The GC is performed using *Randomized Greedy Algorithm (RGA)* [6] which efficiently balances the GC cost (number of page movements) and number of comparisons required to find a candidate block. We also use two different dynamic strategies for PLAlloc function, i.e. *F* and *D* [2], [10], to better exploit all parallelism levels inside SSD. The *F* PLAlloc uses a fully dynamic resource allocation strategy which determines channel, flash chip, die and plane based on a round-robin busy-aware policy; i.e. if a resource to be allocated is busy, then the next one will be allocated in a round-robin manner. In the *D* PLAlloc, all resources are allocated using the same round-robin busy-aware strategy except for die. A mathematical formula is used to determine the die based on LPA value [10]. For BLAlloc, we use a *first-fit* approach that finds the first free block in a round-robin manner and sets it as write-frontier. By default, PAC is considered for all simulation scenarios but BAC is not. If BAC is enabled, corresponding curves are labeled with *+BAC*.

4 Long-Term SSD Behavior

We conducted a set of simulation experiments to investigate the long-term SSD behavior. To this end, each workload trace is repeatedly, and in different rounds, fed into the simulator and the replay process is continued up to a point that the volume of arrived write accesses (Vol_{write}) becomes $10 \times Cap$ (total SSD capacity). Fig. 2 illustrates the results for `proj-0` workload as a representative. In this figure, the variations of different criteria, including average response time, GC execution rate, and the percentage of read/write operations that use plane-level parallelism, are shown during the simulation process.

In the very beginning rounds, where $Vol_{write} < 1 \times Cap$, the average response time of D(+BAC) strategy is better than F(+BAC) due to excellent exploitation of plane-level parallelism for the execution of write operations. As the volume of write accesses increases in next simulation rounds ($Vol_{write} \geq 1 \times Cap$), the average response time of both strategies rises but with a higher rate for D(+BAC). In other words, when the volume of write accesses exceeds the GC execution threshold ($\approx 1 \times Cap$), then the GC is

TABLE 1
Evaluation Scenarios

	Trace	Write Ratio	Req. Size (KB) Read/Write	Inter Arrv. (ms)	Description
Workloads	tpce	8%	8 / 8.4	0.3	Microsoft TPC-E
	prn-0	89%	22.9 / 9.7	108.3	Print server 0 at MSRC
	prn-1	25%	22.5 / 11.7	53.8	Print server 1 at MSRC
	exch	72%	16.8 / 15.9	1.1	Microsoft Exchange server
	fin1	77%	2.3 / 3.8	8.2	Financial1
	prj-0	88%	17.9 / 40.9	143.2	Project directories 0 at MSRC
	post	17%	98.8 / 833	6.9	Posmark file-system benchmark

Configurations	SSD Parameters	NAND Flash Parameters [7]
	User Capacity = 240 GB	No. of Dies per Flash Chip = 2
	Spare-factor = 7%	No. of Planes per Die = 2
	Host Interface Queue Size = 32	No. of Blocks per Plane = 2048
	No. of Bus Channels = 4	Block Size = 2 MB, Page Size = 8 KB
	Channel I/O Transfer Rate = 200 MT/s	Page Read Latency = 75 µs
	No. of Flash Chip per Bus Channel = 4	Page Write Latency = 1600 µs
	Flash Chip Capacity = 16 GB	Block Erase Latency = 5 ms

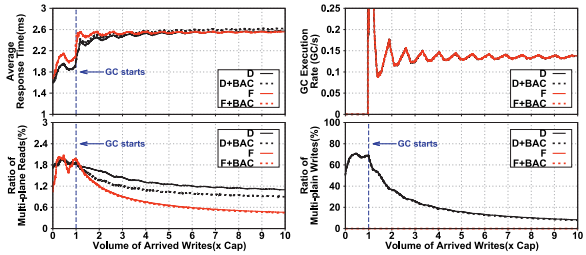


Fig. 2. The long-term SSD behavior when replaying proj-0.

triggered and normal I/O operations of the SSD are blocked by internal tasks that are required for valid page movements and block erasure. More importantly, percentage of multi-plane operations substantially decreases due to GC side-effects on address translation and, in the long-term, D(+BAC) strategy loses its superiority over F(+BAC) as its ratio of multi-plane writes is reduced by a factor of 90 percent. Besides, relaxing BAC in F and D strategies has no significant impact on overall SSD performance as we only see a less than 2 percent improvement in the long-term response time of D with respect to D+BAC.

4.1 GC Side-Effect on Multi-Plane Writes

In a *pristine SSD*, where a great portion of flash blocks are empty, free pages of the same addresses can be easily allocated for multi-plane execution of two consecutive write operations. In particular, if round-robin strategy is used for plane allocation inside a die (as used in F and D), flash writes are evenly distributed among planes of a die and hence page address assignment can be symmetrically progressed inside blocks as depicted in Fig. 3a. Furthermore, in a pristine SSD with BAC, the simple first-fit BLAlloc can select write-frontier blocks with same addresses to fully satisfy the conditions of multi-plane write. Nevertheless, when SSD free pages are exhausted and the GC is performed, the set of available free page addresses may become unequal in the planes of a die due to two GC side-effects: **I)** Blocks of different addresses may be selected for reclamation based on the GC block selection policy. Fig. 3b illustrates an example of this side-effect that causes BAC violation. **II)** GC candidate blocks may contain different number of valid pages and hence different number of free pages are consumed in the write frontier blocks during GC execution. An example of this side-effect is depicted in Fig. 3c.

4.2 GC Side-Effect on Multi-Plane Reads

As shown in Fig. 2, the ratio of read operations that exploit plane-level parallelism decreases in the long-term. In fact, the data of physical pages that participate in a multi-plane read operation (PA and PA' in Fig. 4a) may be moved to different addresses during GC execution. Fig. 4b shows an example where the block at BA is selected for reclamation and its valid pages, including the content of PA, are moved to new addresses while valid pages of BA', including the content of PA', remain in their previous locations (BAC violation). Another example is shown in Fig. 4c where the content of PA and PA' addresses are moved to two different page addresses during GC execution (PAC violation). In a nutshell, if

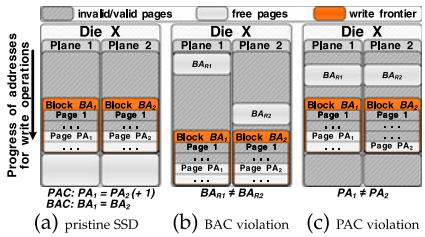
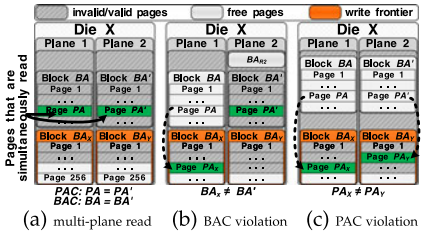


Fig. 3. GC side-effects on write address assignment.



(a) multi-plane read (b) BAC violation (c) PAC violation

Fig. 4. GC side-effects on multi-plane reads.

GC block selection and page movements are performed without address considerations, then the utilization of plane-level parallelism is considerably reduced. Note that separate execution of GC for each plane of a die decreases the chance of performing multi-plane erase operations which can reduce the negative impacts of slow erase operations.

5 Twin Block Management Policy

Here, we propose *TBM* policy to relieve the side-effects of out-of-place-update on the utilization of multi-plane read/write operations. Our policy relieves problems with both BAC and PAC and hence can be used as a general approach to overcome the addressing challenges of plane-level parallelism in the long-term. TBM also enhances the efficiency of GC execution through full utilization of plane-level parallelism for flash erase operations. To satisfy BAC, TBM modifies the FTL management mechanisms so that the same address blocks within neighboring planes, i.e. *twin blocks*, are simultaneously set as write-frontiers and recycled together during GC execution. Instead of individual block status data, TBM uses a specific data structure, called *TBM Table (TBT)*, to keep track of the twin blocks usage. Fig. 5 illustrates the TBT structure which includes a set of entries and an integer variable, namely *Current Page Index (CPI)*, that is used for write address assignment. Each entry of TBT, at an arbitrary index i, is a *TBM Unit (TBU)* and provides the usage summary information of the twin blocks at index i, including *Free Pages Count (FPC)*, *Invalid Pages Count (IPC)* and *Erase Count (EC)*. Previous studies showed how to efficiently maintain such a data structure in flash memory and just temporarily keep the currently used entries (TBUs) in DRAM [11].

5.1 TBM Modifications for Page Allocation

A prerequisite for symmetric address allocation on the planes of a die is the even distribution of write accesses among them. Therefore, TBM requires a PLAlloc strategy that provides round-robin plane allocation inside a die. Algorithm 1 shows the general scheme of such a TBM_PLAlloc function. As depicted, arbitrary allocation strategies can be used for channel, flash chip and die assignment but, within a die, plane must be allocated in a round-robin manner. The F and D strategies, used in the previous sections, satisfy such a requirement.

Algorithm 1 also explains TBM_BLAlloc function which uses CPI for symmetric address allocation in twin blocks. In the first step, TBM_PLAlloc selects a *Write-Frontier TBU (WFU)* whose twin blocks are used as the write-frontiers of their corresponding planes. CPI is set to 1 for a newly selected WFU (Line 3) since all its twin blocks are free. Address allocation starts from Page 1 of the write-frontier of Plane 1 and, due to round-robin plane allocation, continues on Page 1 of the twin write-frontiers at neighboring planes. When Page 1 of the last plane is written, CPI is incremented (Line 8) and thus address allocation proceeds for the next CPI values till the last page index of the twin write-frontiers. In this way, address allocation is symmetrically progressed on twin blocks to satisfy PAC and the probability of mapping consecutive write operations to the same addresses inside neighboring planes increases.

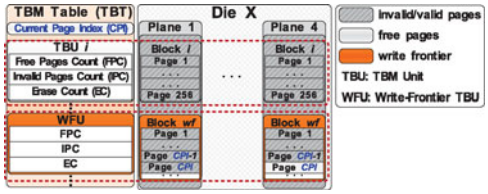


Fig. 5. The data structure used in a TBM realization.

Algorithm 1. Address Translation Functions for TBM

```
1: function TBM_PLAlloc(LPA)
2:   Channel ← AllocateChannel(LPA)        ▷ arbitrary strategy
3:   Chip ← AllocateChip(LPA, Channel)
                                           ▷ arbitrary strategy
4:   Die ← AllocateDie(LPA, Channel, Chip)
                                           ▷ arbitrary strategy
5:   Plane ← current plane of Die
6:   UPDATE current plane of Die in a round-robin manner
7:   return (Channel, Chip, Die, Plane)

1: function TBM_BLAlloc(Channel, Chip, Die, Plane)
2:   while WFU of Die has no free pages (FPC = 0) do
3:     UPDATE WFU in a round-robin manner, CPI ← 1
4:   Block ← the block of Plane determined by WFU
5:   Page ← the page of Block determined by CPI
6:   DECREMENT FPC of WFU by one
7:   if Plane is the last plane of Die then   ▷ all planes are
     written at CPI
8:     INCREMENT CPI of WFU by one
9:   return (Block, Page)
```

5.2 TBM Modifications for GC

As mentioned in Section 4, recycling of the blocks with same addresses in the planes of a die plays a key role to better exploit plane-level parallelism. Besides, the correct operation of the proposed TBM_BLAlloc function depends on the outcome of GC execution since all twin blocks of WFU must be free and available for write. Therefore, TBM_GC is designed to simultaneously perform GC for all planes of a die. As shown in Algorithm 2, TBM_GC selects a candidate TBU by searching the TBT entries based on an arbitrary selection policy (RGA or etc.). Next, the destination is set for all valid pages of the candidate TBU. If there are valid pages with same indexes inside twin blocks, then same page addresses are assigned within blocks of the WFU using TBM round-robin allocation strategy. Otherwise, the destination is set by calling TBM_BLAlloc. This helps to relieve the read PAC violation shown in Fig. 4c. Having destination assignment finished, valid pages are moved and then the erase operation is performed in parallel for all planes of the target die and twin blocks are reclaimed. To better exploit parallelism, GC can also be performed for other dies of the chip if they are near GC execution threshold (e.g. +5 percent free pages).

6 Evaluation Results

We use the simulation methodology and SSD model described in Section 3 to evaluate the effectiveness of TBM. Fig. 6 shows the long-term SSD behavior when TBM is used and `proj-0` workload is replayed. Comparing the results of Figs. 2 and 6 reveals that TBM greatly reduces the GC execution rate since more flash blocks are recycled for each GC invocation. Therefore, SSD resources are less engaged in erase operations and waiting time of the normal I/O operations decreases. On the other hand, TBM keeps the ratio of multi-plane read and write operations nearly constant during simulation. Consequently, the negative impacts of GC are mitigated and the long-term SSD response-time is improved with respect to the base FTL. Fig. 7 shows the summary of long-term simulation results for different traces. In addition to D and F

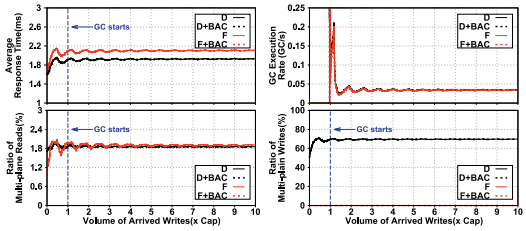


Fig. 6. The long-term SSD behavior when FTL uses TBM.

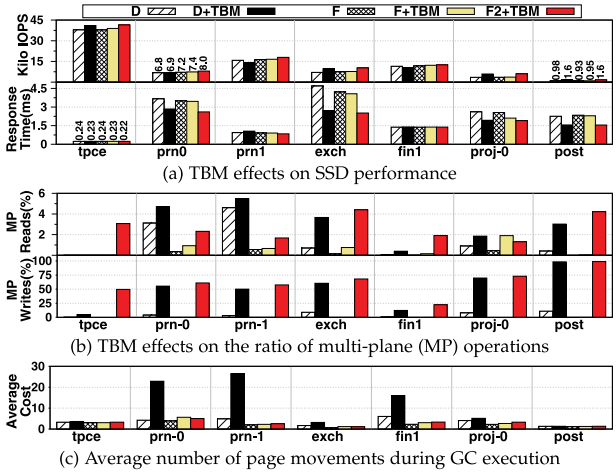
allocation strategies, simulations are also performed for a special variation of F, namely F2, which prioritizes plane-level parallelism over die-level. In fact, if there are two consecutive write operations in I/O queue, F prefers die-level parallelism for their execution but F2 favors plane-level. Since the base results of F2 are almost identical to F, they are omitted in the charts. The charts just include the scenarios without BAC, since considering/ignoring the BAC leads to negligible variation in the achieved results (0 to 2 percent). As can be seen, TBM always improves the performance of F strategy (F+TBM with respect to F) but the maximum IOPS/response time improvement rate is limited to 5 percent/17 percent as F+TBM just benefits from lower GC blocking time. However, D+TBM and F2+TBM greatly exploit plane-level parallelism and can improve IOPS/response time by up to 68 percent/42 percent and 73 percent/40 percent, respectively. Nonetheless, the D+TBM may negatively impact performance for some workloads, such as `fin1` and `prn1`. Considering the average GC cost results, it can be seen that D+TBM requires higher number of GC page movements due to larger variations in the distribution of valid/invalid pages among twin blocks. This negative impact is compensated in `prn-0` by better usage of plane-level parallelism and less GC triggers. Also, I/O request arrival pattern decreases the probability of GC interference. However, for `prn-1` and `fin-1`, extra GC page movements interfere with normal I/O operations and destroy the positive impacts of TBM; hence a performance drop of 1 to 10 percent is seen.

Algorithm 2. Garbage Collection for TBM

```
1: procedure TBM_GC(Channel, Chip, Die)
2:   CandidateTBU ← SelectCandidateBlock(TBT)
3:   for all P ∈ valid pages of the CandidateTBU do
4:     if exist P′ ∈ valid page with PageIndex_P = PageIndex_P′
       then
5:       ASSIGN Dest[P] and Dest[P′] in WFU of Die with
         identical indexes using TBM round-robin page allo-
         cation strategy
6:     else
7:       Dest[P]←TBM_BLAlloc(Channel,Chip,Die,current
         plane of Die)
8:       UPDATE current plane of Die in a round-robin
         manner
9:   for all P ∈ valid pages of the CandidateTBU do
10:    MOVE P to Dest[P]
11:  PARALLEL ERASE all blocks of the CandidateTBU
12:  UPDATE CandidateTBU: reset FPC, IPC←0, EC← EC + 1
```

7 Related Work

In order to effectively exploit SSD internal parallelism, the address mapping strategy should fairly stripe flash operations over channel-, chip-, die- and plane-level resources. In the next step, scheduler must resolve resource contention among mapped flash operations while searching for operations that can be grouped for parallel execution at die- and plane-level. Prior studies proposed dynamic address mapping strategies to enhance address striping of write operations over idle flash resources and hence achieved significant performance improvements [2], [8], [9], [10]. Other



(a) TBM effects on SSD performance

(b) TBM effects on the ratio of multi-plane (MP) operations

(c) Average number of page movements during GC execution

Fig. 7. TBM evaluation results for different traces.

studies proposed scheduling schemes for out-of-order execution or rescheduling of queued operations in order to resolve their resource contentions and better exploit intra-chip parallelism [3], [5], [8]. However, all these proposals fail to consider the long-term SSD performance when GC side-effects negatively reduce the probability of multi-plane operations. TBM, in contrast, better exploits plane-level parallelism in the long-term for LPA to PPA mapping. Consequently, it helps to boost the performance of any scheduling scheme that is aware of plane-level parallelism, including Sprinkler [3], in such a way that higher number of mapped operations conform to multi-plane addressing constraint. Recently, Jung et al. proposed HIOS, a scheduler that focuses on a different aspect of GC execution [4]. HIOS redistributes GC execution overheads across non-critical I/O requests in order to reduce channel resource contention and satisfy QoS requirements. Therefore, HIOS and TBM target orthogonal problems.

8 Conclusion

We proposed a new policy to overcome addressing constraints of plane-level parallelism to effectively exploit it in the long-term to speedup write, read and erase operations. Evaluation results show that the proposed scheme could improve IOPS and response time by a factor of up to 73 and 42 percent, respectively.

References

[1] "Open NAND flash interface specification 4.0," 2014.
[2] Y. Hu, et al., "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. Int. Conf. Supercomput.*, 2011, pp. 96–107.
[3] M. Jung, et al., "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. 20th Int. Symp. High Perform. Comput. Archit.*, 2014, pp. 524–535.
[4] M. Jung, et al., "HIOS: A host interface I/O scheduler for solid state disks," in *Proc. 41st Annu. Int. Symp. Comput. Archit.*, 2014, pp. 289–300.
[5] 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.
[6] Y. Li, et al., "Stochastic modeling of large-scale solid-state storage systems: Analysis, design tradeoffs and optimization," in *Proc. ACM Int. Conf. Meas. Model. Comput. Syst.*, 2013, pp. 179–190.
[7] Micron Technology, Inc., "MT29E256G08CMCAB," 2010.
[8] C. Park, et al., "Exploiting internal parallelism of flash-based SSDs," *Comput. Archit. Lett.*, vol. 9, no. 1, pp. 9–12, Jan. 2010.
[9] J. Shin, et al., "FTL design exploration in reconfigurable high-performance SSD for server applications," in *Proc. 23rd Int. Conf. Supercomput.*, 2009, pp. 338–349.
[10] A. Tavakkol, et al., "Unleashing the potentials of dynamism for page allocation strategies in SSDs," in *Proc. ACM Int. Conf. Meas. Model. Comput. Syst.*, 2014, pp. 551–552.
[11] C.-W. Tsao, et al., "Performance enhancement of garbage collection for flash storage devices: An efficient victim block selection design," in *Proc. 50th ACM/EDAC/IEEE Des. Autom. Conf.*, 2013, pp. 1–6.