# Decoupled SSD: Reducing Data Movement on NAND-Based Flash SSD

Jiho Kim, Myoungsoo Jung, and John Kim

**Abstract—**Modern NAND Flash memory-based Solid State Drive (SSD) is designed to support high bandwidth for I/O requests by exploiting various parallelism including multiple channels, multiple flash memory chips, and multiple planes. However, SSD system is utilized not only for general I/O requests but is also used during flash memory management processes (e.g., garbage collection). In particular, the sharing of system resources (e.g., system bus, DRAM) for I/O requests and garbage collection can cause performance degradation. In this letter, we address the system bus bottleneck and propose ***Decoupled*** SSD system that decouples the front-end (i.e. cores, system bus) with the back-end (i.e., flash memory) and provide an on-chip network to interconnect the controllers together. Our decoupled SSD enables advanced command (i.e. copy-back) to be exploited for efficient garbage collection; in particular, we propose to extend copy-back commands to enable *global* copy-back through the flash-controller interconnect to effectively decouple I/O path and garbage collection path. Our evaluations show that decoupled SSD results in up to 34.7% bandwidth improvement, for I/O traffic while achieving up to 69% speedup for garbage collection.

**Index Terms—**Solid-state drive, flash memory, copy-back, interconnection networks

## 1 Introduction

Bandwidth of NAND flash memory-based Solid State Drive (SSD) has increased with improvement in flash memory technology (e.g. ultra low-latency [2]) and high-speed interface protocol. To support high bandwidth of I/O read/write requests, internal parallelism of an SSD is exploited (e.g., multi-channels, chips, dies, etc.) and multi-plane commands [9] enable additional bandwidth through the multiple (parallel) planes. The SSD firmware (or FTL) translates high-level I/O requests (logical address) to low-level I/O requests (physical address) and sends low-level requests (commands) to the flash controller to handle the I/O operations. As a result, in addition to the flash memories, other system resources such as cores, DRAM, and system bus are also heavily utilized during I/O operations. However, in addition to I/O request, garbage collection (GC) uses the same resources, including cores, DRAM, ECC, and system-bus as well as the flash memory when valid pages are copied to the destination free blocks.

Since both I/O requests and garbage collection use the same resources, the conflicts between the two can impact overall performance and throughput. To address these conflicts, most prior work focused on the flash memory conflicts [11] because of the long latency of flash memory. However, system resources (especially system-bus) is emerging as a more critical bottleneck because of the increase in the unit of data transfer per plane (e.g., 2 or 4KB in ultra-low latency (ULL) flash [2] and 16 or 32KB page size in recent T/QLC [4]) and also the increasing number of planes – (8 or 16 planes in ULL [2] and 2 or 4 planes in recent T/QLC [4]). As a result, increasing page size and the larger number of planes not only increases flash memory parallelism (and thus bandwidth), it impacts the utilization of the system resources, in particular the system-bus, and I/O requests can be heavily impacted by garbage collection.

In this work, we identify that a fundamental cause of this performance bottleneck is the *tightly coupled* SSD system between the system resources (i.e., system bus, DRAM, etc.) and the flash memory chips. For example, when data transfer or copy is done for garbage collection, it essentially consists of chip-to-chip (or die-to-die) communication; however, the valid pages are sent to the intermediate system components (e.g., DRAM and ECC) through system-bus before arriving at their destination flash memory and creates conflict for the system resources.

To overcome the performance bottleneck, we propose a decoupled SSD system where the *front-end* (i.e., core, DRAM, system bus) are decoupled from the *back-end* or the flash memory chips. Main micro-architectural modification is the introduction of an on-chip router that interconnects the flash controllers together through an on-chip network. We assume error correction logic is integrated within the flash controller. As a result, data movement during garbage collection can be minimized (and decoupled) as the front-end resources are not utilized. In addition to the router itself, we leverage the router such that advanced commands such as copy-back can be supported. Because of error propagation, copy-back commands are limited in modern SSD but the router provides an effective mechanism to "route" traffic and support copy-back to enable *local* copy-back that includes a read (and error correction) and a write command.

We also propose to extend existing copy-back commands to *global copy-back* that enables chip-to-chip communication through the decoupled SSD (and the on-chip network). Existing copy-back commands are effectively *local* copy-back since copies occur within the same plane (die); however, copy-back in our decouple SSD system is no longer limited. As a result, the destination of the data transfer can be located on the same flash bus channel or located across different channels to increase flexibility of copy-back. In this work, the benefits of the decoupled SSD is demonstrated by exploiting the benefits of global copy-back command but we also discuss how decoupled SSD can enable more efficient management of the flash memory, including efficient support for superblock as well as wear-leveling and reliability management.

## 2 Modern SSD System

### 2.1 Garbage Collection

Garbage collection (GC) process is a critical component in SSD. It is necessary to optimize storage space in SSDs as writes are done at page granularity that is much smaller than the block granularity used for erase operations [6]. When the amount of remaining free blocks is less than a predetermined threshold value, FTL starts garbage collection by allocating a new physical page from the active free block, moving valid pages from the victim blocks to free or new blocks, and invalidates the page from the victim block. Afterwards, GC erases the victim blocks and optimizes storage space on the SSD.

Different algorithms have been proposed to select the victim blocks and the free blocks, with different trade-offs between performance [11] and reliability [10]. If the free block chosen is within the same plane as the victim block, *advanced copy-back command* can be exploited to avoid transferring data outside of the flash memory chip. However, advanced copy-back commands are difficult to exploit in modern SSDs because of error propagation since error correction cannot be done within the flash memory. In comparison, *global* free block selection chooses free block that is located in a different chip, die, and/or plane. To exploit multiple planes within a die, multi-plane advanced

• *The authors are with the School of Electrical Engineering, KAIST, Daejeon 34141, Republic of Korea. E-mail: {jihokim, jjk12}@kaist.ac.kr, mj@camelab.org.*

*Manuscript received 1 Sept. 2021; accepted 30 Sept. 2021. Date of publication 8 Oct. 2021; date of current version 28 Oct. 2021.*
*This work was supported in part by the Institute of Information & communications Technology Planning & evaluation (IITP) and the National Research Foundation of Korea (NRF) grant funded by Korea government (MSIP) under Grants IITP-20200013090022002, IITP-20200013030022003, IITP-2021-0-00524, and NRF-2021R1AC4001773.*
*(Corresponding author: John Kim.)*
*Digital Object Identifier no. 10.1109/LCA.2021.3118688*

1556-6056 © 2021 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: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:58:08 UTC from IEEE Xplore. Restrictions apply.

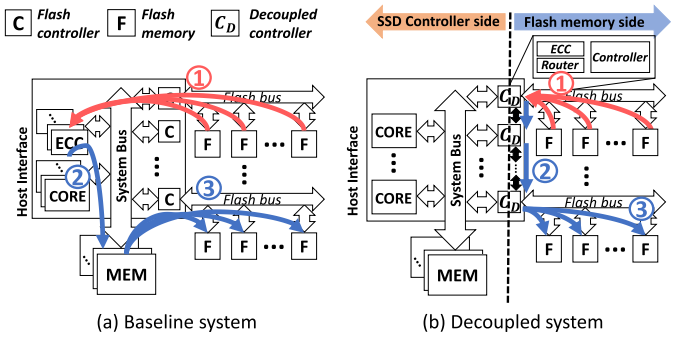



Fig. 1. (a) Execution path of garbage collection (GC) in current SSD system with heavy accesses of system resources, and (b) Execution path of GC without accessing system-bus and DRAM.

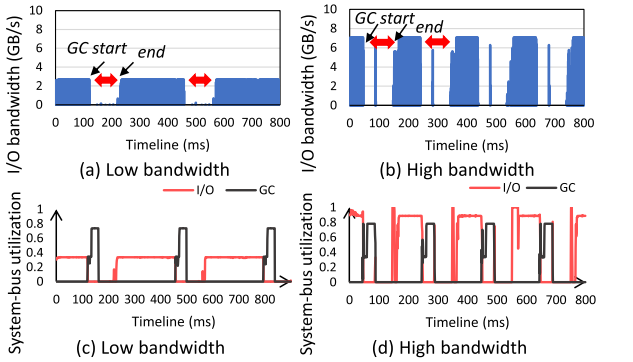



Fig. 2. (a,b) I/O bandwidth and (c,d) system-bus utilization based on simulation of a ULL device.

commands have been exploited to improve GC performance [9]. In this work, we propose a *decoupled* SSD system that can support advanced copy-back commands while providing support for error-correction and avoid utilizing system resources.

### 2.2 I/O Performance Degradation

High-level overview of data movement during garbage collection (GC) in modern SSD systems is shown in Fig. 1a. The flash controller sends read commands to the appropriate flash memory die and the valid pages are read out, goes through error detection (and possibly correction) (Fig. 1a ①), before being sent through the system bus and written out to the DRAM memory (②). Afterwards, FTL issues write operation to the flash controller and transfers the pages to the destination flash die by sending write commands to the flash memory chips (③). The performance impact of GC on the I/O requests are shown in Fig. 2(a,b) where the I/O bandwidth, defined as the amount of bandwidth provided to the I/O requests[1] is shown on the $y$-axis. The $x$-axis is time and we measure I/O bandwidth every 1 msec to observe the change in the I/O bandwidth. We simulate an SSD system based on an ULL device [2] and evaluate the impact of GC on I/O requests. We used greedy and global free block selections for our evaluation and all flash memory executes GC in parallel. The results are shown for a synthetic workload that consists of 4KB (as well as 32KB) sequential write I/O requests with an outstanding request queue depth of 64. The 4kB access is used to evaluate "low bandwidth" scenario since only one out of the eight planes are utilized. In comparison, 32kB access results in all 8 planes being accessed and creates "high bandwidth" demands.

For low bandwidth (Fig. 2a), initially, approximately 3 GB/s of I/O bandwidth is sustained but the I/O bandwidth drops after the GC is triggered. During GC, limited amount of I/O requests are processed. Similar behavior can be observed for "high bandwidth" (Fig. 2b) but the drop is more significant since higher bandwidth of the flash memory is exploited. It is well-known that GC interferes with I/O requests [6] [11]; however, as the flash memory bandwidth increases (e.g., exploiting multi-plane in modern SSD devices), the impact of system resources (such as the system bus) becomes more critical. To understand the impact, we plot the corresponding system bus utilization in Fig. 2(c,d). When GC is active, the system bus utilization for I/O requests drops; however, the drop in utilization is more significant with the "high" bandwidth simulations and the system bus becomes a point of contention between I/O and GC within the SSD. In the following section, we describe how the proposed decoupled SSD (dSSD) architecture results in GC that does not need to access the system bus – thus, improves the performance of not only GC but also the performance of I/O requests as well.

1. The system-bus bandwidth is modeled as 8GB/s and is equal to the aggregate bandwidth of all flash bus channels. The bandwidth of an individual flash chip with 1 plane (low bandwidth) is 51.2 MB/s and 8 planes are 409.6 MB/s for WRITE.

## 3 Decoupled SSD System

In this section, we describe our proposed decoupled SSD system that eliminates redundant accesses during garbage collection (GC) that enables data copy directly between flash controllers without utilizing the system bus and the core/DRAM. We also describe how enhanced *global* copy-back commands can be proposed based on the decoupled SSD architecture.

### 3.1 Decoupled SSD Architecture

A high-level overview of the proposed decoupled SSD architecture is shown in Fig. 1b. Compared to the baseline (Fig. 1a), the main hardware difference is the *decoupled* flash controller logic ($C_D$). The ECC logic is integrated within the controller and a hardware router is also added to enable controller-to-controller communication. Since all controllers are located within the same chip [7], a network-on-chip (NoC) is introduced that provides direct communication between flash memories chips across different flash channels. As a result, the flash memory-side (or the "back-end") is effectively *decoupled* from the SSD controller side (or the "front-end") through the controller and data movement within the flash memory such as copy operation is decoupled from the SSD controller side data movement. More importantly, it reduces contention for shared resources, including the system bus and the memory, when pages are copied during garbage collection while I/O requests are handled at the same time. Another key benefit from the decoupled SSD is that advanced commands (i.e., copy-back commands) that were not feasible on modern SSD systems because of error propagation is now possible through the decoupled controller.

The microarchitecture of decouple SSD in Fig. 1b is one particular implementation but there can be alternative implementations. One design decision is the NoC topology. In Fig. 1b), a 1D mesh topology is used within the NoC but alternative topologies can be leveraged, including a ring topology or a higher dimension topology, that will provides different trade-off in terms of latency/performance and cost. We also assumed one router per controller (and per flash bus channel) that were connected using a 1D mesh topology and one ECC enginer per router; however, different number of routers and/or ECC engines can be used within the NoC. For example, in modern flash controllers, there are multiple ECC units that are shared [7]. In our decoupled SSD, the ECC units are distributed across the different routers. For example, prior work [8] has an ECC unit that is shared between two flash channels. In our work, if the cost of having an ECC unit at each router is too high, two routers can share an ECC unit.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:58:08 UTC from IEEE Xplore. Restrictions apply.

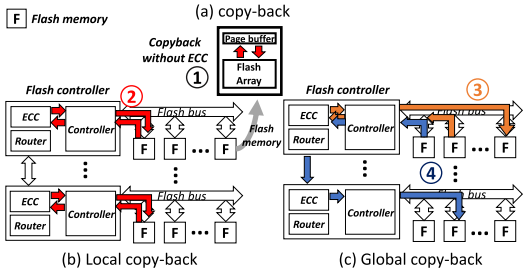



Fig. 3. (a) Local copy-back of the data within same chip either non-error corrected using copy-back command or error corrected using data transferring, (b) Global copy transfers the data from a flash chip to any flash chip either in the same channel or different channel.

### 3.2 Global Copy-Back Command

Similar to the baseline SSD, FTL in the decoupled SSD issues basic commands that are generated by the controllers (e.g., read or write commands). For I/O requests, the data path used in the decoupled SSD is the same as the baseline system. However, for any data movement within the flash memory side (i.e., page copy), the SSD front-end does not need to be accessed. Thus, the two main changes in the hardware control logic are – 1) provide support for additional "features" (e.g., copy-back) and 2) enable packetization for communication between the routers (or the flash channels).

Data movement during conventional copy-back is shown in Fig. 3a (①) where the data movement is limited to within the die – as a page is read to a page buffer inside the flash memory and then, directly written to another page (identified as the free block). However, such copy-back commands are limited in modern flash memory [1] since error correction is bypassed and errors can be propagated. In comparison, decoupled SSD enables copy-back advanced command through the integration of the router and ECC unit within the flash controller. The copy-back command from the FTL is sent to the decoupled controller, not directly to the flash memory itself. The decoupled controller then generates a sequence of read and write commands to the appropriate flash memory. An example of copy-back for decoupled SSD is shown in Fig. 3b. A read command is issued such that the page is read out from the flash. ECC is used to detect error (and perform correction as necessary) and afterwards, and write command is issued such that the data page is written or transferred back to the *same* flash memory die through the flash bus (②). We refer to such copy-back in our decoupled SSD as *local* copy-back since the source and destination are the same flash memory. The commands seen at the flash memory is similar to what is observed in the baseline SSD system when "copy" needs to be done – a read followed by a write command. However the key difference is that the read does not access system resources, including the system bus and the DRAM. For such local copy-back, the router or the "packetization" is not required for the copy-back.

In addition to local copy-back, decoupled SSD also enables *global* copy-back (Fig. 3c) where the destination is not restricted to the same flash memory die. *Global copy-back* leverages chip-to-chip (or die-to-die) communications where the destination can be a different flash memory die that is connected to the same flash bus (③). The destination can also be a flash memory chip that is connected to a different flash controller (④)) and routed through the on-chip interconnect. The main difference of global copy-back, compared to local copy-back, is the destination or the write address. If the destination remains within the same flash bus channel, the write command is simply issued to a different flash chip. However, if the destination is located on a different flash bus, the page that is read is packetized (i.e., a packet header is appended with the destination information) and the NoC properly routes the packet to the appropriate destination router/controller – before the write command is issued.

By providing *global* copy-back, decoupled SSD provides the FTL with more flexibility in terms of free block selection (or the destination of the copy) during garbage collection. The intra-chip and inter-chip reliability variance among the flash memory chips can vary significantly [10] – thus, simply providing local copy-back does not necessarily guarantee balancing of the wear-leveling across the entire SSD system; however, global copy-back enables more flexibility and multiple destination options during the write phase of copy-back enables better balancing of the number of copied pages by the garbage collection. In the garbage collection that we implement on the decouple SSD, we leverage global copy-back commands to provide a balanced endurance across the different flash memories during garbage collection.

TABLE 1
Simulation Parameters

| Simulation | Parameters |
|---|---|
| Simple-SSD | PCI-E 3.0 x8 lanes, 8 channels 8 ways 1 die, 8 planes system bus bandwidth = 8GB/s (= x1) |
| Flash (ULL) | read = 5us, write = 80us, erase = 2ms, page size = 4KB |
| Memory | flash bus freq = 1000Mhz , flash bus width = 8 bits |
| BookSim | Topology = 1D mesh, k = 8, n = 1, routing = dim order |

## 4 Evaluation

We used Simple-SSD simulator [3] in standalone mode and integrate Booksim [5] interconnect simulator to implement our proposed decoupled SSD. Detailed simulation parameters are listed in Table 1. We assumed 8 channels with 8 flash memory per channel with 8 planes. Each flash controller in the decoupled SSD has an ECC, a router, and is responsible for controlling the flash bus and flash memory. The topology of flash controller network is a 1-D mesh with minimal routing. We assumed a flash chip with one flash die for simplicity and assumed ultra-low latency parameters for flash memory [2], [3]. To generate garbage collection (GC) during I/O requests, we first initialized and invalidated all of the pages randomly. Then, we used random 4KB (for low) /128KB (for high) I/O requests with 64 I/O queue depth to fully utilize the SSD. Initially, the I/O requests are fully utilizing the SSD's internal bandwidth and we evaluate the performance when garbage collection is performed.

The results of I/O bandwidth and GC performance improvement from the decoupled SSD (dSSD) is shown in Fig. 4, with the results normalized to baseline SSD, showing the results from "low" and "high" bandwidth simulations, similar to Fig. 2 earlier. We show the results of baseline (×1) and where the system bus bandwidth is increased by a factor ×1.5 and ×2. We also compare against two different implementation of dSSD – dSSD(×1.5) and dSSD(×2) where the added *bisection* bandwidth of the NoC matches the increase in system bus bandwidth (e.g., ×1.5 and dSSD(×1.5) provide the same amount of on-chip bandwidth).

Since "low" bandwidth does not fully utilize the system-bus bandwidth even during I/O and GC contention, increasing system bus bandwidth has minimal impact on overall performance – resulting in only 4.6% and 13.6% (for GC) improvement even

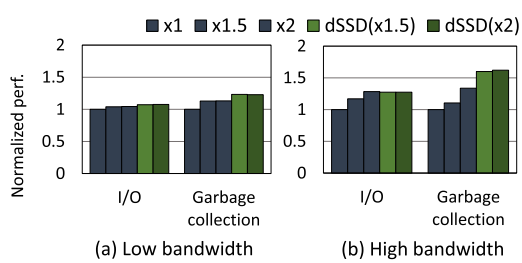



Fig. 4. Performance improvement for I/O and GC on (a) low flash memory bandwidth and (b) high flash memory utilization when increasing the system-bus bandwidth.

Authorized licensed use limited to: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:58:08 UTC from IEEE Xplore. Restrictions apply.

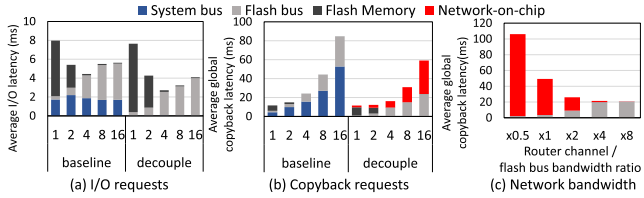



Fig. 5. Latency breakdown of (a) I/O requests, (b) copy-back commands, and (c) the impact of router channel bandwidth on copy-back commands.

when the system bus is increased by 2×. There is slightly more benefit with dSSD as performance improves to approximately 10.7% and 25.4% (for GC). However, for " high" bandwidth (Fig. 4b), increasing the system-bus bandwidth (i.e., ×1.5 and ×2) directly impacts both I/O bandwidth and GC performance. With ×2, I/O and GC improves by 28.4% and 32.3%, respectively. However, with the *same* amount of on-chip bandwidth, dSSD is able to provide further improvement in performance – especially in terms of GC performance. Compared to baseline, dSSD(×2) improve I/O and GC performance by 34.7% and 69%, respectively (and 4.9% and 27.7% compared to baseline(×2)). The decoupled SSD effectively decouples the bandwidth used by the front-end (for I/O) with the back-end (copy-back) – thus, when high on-chip bandwidth is required, it becomes more effectively to "decouple" the bandwidth or separate them to improve overall performance.

To understand the benefit of decouple SSD compared to the baseline (×1), the latency breakdown of I/O requests and copy-back requests is shown in Fig. 5. Fig. 5a shows the average latency breakdown of I/O request. With 1 plane, the I/O latency is determined mostly by the contention for the flash memory; however, as the number of planes increase the contention for flash memory (chip) is significantly reduced but the contention for the flash bus increases, for both the baseline and the decoupled SSD. However, the key difference is that the contention for system bus is removed for the decoupled SSD. The copy-back request latency breakdown for GC (Fig. 5b) shows the copy-back operation is heavily dominated by the system-bus and flash bus utilization/contention. In the decoupled system, the NoC latency dominates as the number of planes increase but since NoC is only used by the copy-back requests, the increase in latency is lower than the added latency from the system bus contention in the baseline SSD.

A key design parameter of decoupled SSD is the amount of on-chip interconnect bandwidth provided for the controller-to-controller communication. In Fig. 5c, while holding the flash bus bandwidth constant, we vary the inter-router channel bandwidth to evaluate the impact of NoC bandwidth. As the amount of router channel bandwidth increase, performance (i.e., copyback latency) is improved but effectively saturates when the amount of router channel bandwidth is 2×. Note that our proposed decouple SSD does not reduce flash memory resource conflict; however, by providing a separate network for controller-to-controller communication, both I/O request and garbage collection interference on the system bus and memory conflict are avoided and to improve performance.

*Discussions:* Garbage collection (and advanced command) was used to illustrate the benefits of the proposed decouple SSD; however, the benefits extends to improving flash memory endurance and management. For example, modern SSD's often organize physical blocks as *superblocks* to reduce the size of the mapping table in the FTL. A superblock consists of multiple physical block as shown in Fig. 6a, with each physical block located across the different channels to exploit flash channel parallelism. The block ID across multiple planes/dies and flash channel is identical to minimize the complexity of the superblock; however, blocks in different chips can have different endurance and *static* superblock mapping does not provide mapping flexibility. Thus, if one of the block fails, the entire superblock cannot be used. With the decoupled SSD architecture, the routers within the controllers can be leveraged to enable *dynamic* superblock mapping by introducing a re-mapping table within the router (Fig. 6b). The dynamic mapping does not require changes to existing FTL but rely on the decoupled SSD architecture hardware support.

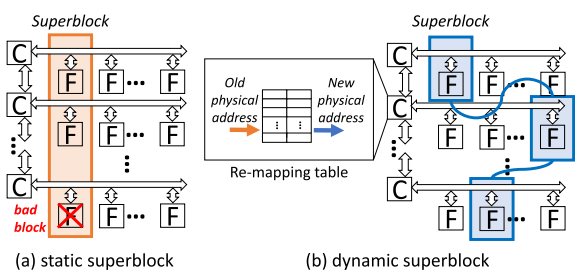



Fig. 6. (a) Example of a partial bad-block status in a (static) superblock and (b) flexible, dynamic superblock using re-mapping table in the router.

## 5 Conclusion

In this work, we proposed *decoupled SSD* that eliminates unnecessary usage of system resources of flash controller-side by introducing the on-chip network to interconnect the controllers together. The decoupled SSD enables direct communication between flash controllers to efficiently move pages during garbage collection. To support efficient communication between different flash memory, we proposed *global* copy-back command and our initial evaluation shows that decoupled SSD improves both I/O and garbage collection by up to 34.7% (and 69%), respectively.

## References

[1] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.

[2] W. Cheong *et al.*, "A flash memory controller for 15$\mu$s ultra-low-latency SSD using high-speed 3D NAND flash with 3$\mu$s read time," in *Proc. IEEE Int. Solid - State Circuits Conf.*, 2018, pp. 338–340.

[3] D. Gouk *et al.*, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *Proc 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2018, pp. 469–481.

[4] H. Huh *et al.*, "13.2 A 1tb 4b/cell 96-stacked-WL 3D NAND flash memory with 30mb/s program throughput using peripheral circuit under memory cell array technique," in *Proc. IEEE Int. Solid- State Circuits Conf.*, 2020, pp. 220–221.

[5] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2013, pp. 86–96.

[6] M. Jung *et al.*, "HIOS: A host interface i/o scheduler for solid state disks," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 289–300, 2014.

[7] Marvell, "88SS1098 SSD controller," 2018. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/storage/marvell-storage-88ss1098-product-brief-2018–03.pdf

[8] Marvell and E. Yeo, "An LDPC-enabled flash controller in 40 nm CMOS," in *Proc. Flash Memory Summit*, 2012, pp. 1–25.

[9] N. Shahidi, M. T. Kandemir, M. Arjomand, C. R. Das, M. Jung, 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. Shim *et al.*, "Exploiting process similarity of 3d flash memory for high performance SSDs," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 211–223.

[11] S. Yan *et al.*, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds," *ACM Trans. Storage*, vol. 13, no. 3, pp. 1–26, 2017.

▷ **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: Korea Advanced Inst of Science & Tech - KAIST. Downloaded on March 04,2026 at 01:58:08 UTC from IEEE Xplore. Restrictions apply.