2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO) | 978-1-6654-6272-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/MICRO56248.2022.00038

Networked SSD: Flash Memory Interconnection Network for High-Bandwidth SSD

Jiho Kim
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
jihokim@kaist.ac.kr

Seokwon Kang
Dept of Computer Science
Hanyang University
Seoul, Republic of Korea
kswon0202@gmail.com

Yongjun Park
Dept of Computer Science
Yonsei University
Seoul, Republic of Korea
yongjunpark@yonsei.ac.kr

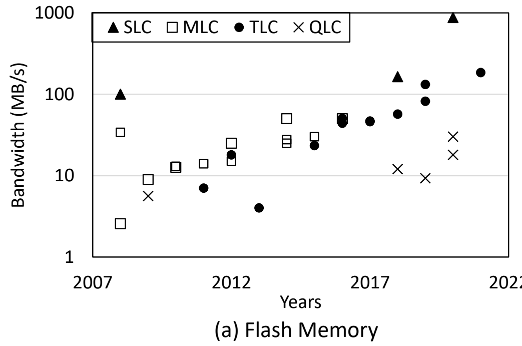
John Kim
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
jjk12@kaist.edu

***Abstract*—As the flash memory performance increases with more bandwidth, the flash memory channel or the interconnect is becoming a bigger bottleneck to enable high performance SSD system. However, the bandwidth of the flash memory interconnect is not increasing at the same rate as the flash memory. In addition, current flash memory bus is based on dedicated signaling where separate control signals are used for communication between the flash channel controller and the flash memory chip. In this work, we propose to exploit packetized communication to improve the effective flash memory interconnect bandwidth and propose packetized SSD (pSSD) system architecture. We first show how packetized communication can be exploited and the microarchitectural changes required. We then propose the Omnibus topology for flash memory interconnect to enable a *packetized network* SSD (pnSSD) among the flash memory – a 2D bus-based organization that maintains a "bus" organization for the interconnect while enabling direct communication between the flash memory chips. The pnSSD architecture enables a new type of garbage collection that we refer to as *spatial garbage collection* that significantly reduces the interference between I/O requests and garbage collection. Our detailed evaluation of pnSSD shows 82% improvement in I/O latency with no garbage collection (GC) while improving I/O latency by 9.71× when GC occurs in parallel with I/O operation, through spatial garbage collection.**

***Keywords*-Solid state drive, interconnection networks, garbage collection**

I. Introduction

Bandwidth of NAND flash memory-based Solid State Drive (SSD) has increased with improvement in flash memory technology [19] [20] [36] [6] [31] and high-speed interface protocol [10]. To support high-bandwidth of I/O requests, internal parallelism of an SSD has been exploited (e.g., multiple channels, chips, dies, etc.) and multi-plane commands [37] enable additional bandwidth through the multiple parallel planes. As the more flash memory bandwidth results in more I/O requests, the performance requirement of the Flash Translation Layer (FTL) continues to increase; in addition to the flash memories, other system resources such as cores, DRAM, and system-bus are heavily utilized during I/O operations.



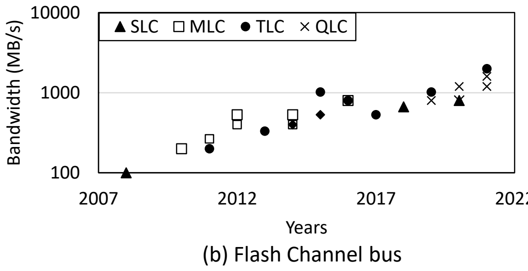


Figure 1: (a) Flash memory (write) bandwidth trend across various flash memory chips from the industry and (b) the flash memory bus bandwidth trend over the past 15 years.

Fig 1(a) illustrates how the flash memory bandwidth per chip has increased over the past 15 years where each data point represents flash products from different vendors. There has been approximately an order of magnitude increase in bandwidth every 5 years. However, bandwidth of the flash memory channels (or bus) has not increased at the same rate, as shown in Fig 1(b), as there has been an order bandwidth increase approximately every 10 years. Since there are multiple flash memory chips connected to a given channel (e.g., 8 – 16), the increase in parallelism results in the flash channel interconnect becoming a bigger bottleneck in overall system performance. In this work, ***we propose how a packetized interface can increase the effective bandwidth in the flash channel bus***.

For multi-core processors or system-on-chip, network-on-chip (NoC) architectures have been proposed [9] where instead of ad-hoc or dedicated wiring, interconnect resources can be more efficiently utilized when "packets" are routed, not "wires." In this work, we take a similar approach in the design of the flash memory channel or interconnect. In particular, modern flash memory channels are effectively designed as *dedicated* wires as different control signals are provisioned for communication between the flash channel controller and the flash memory chips while dedicated signals (or wires) are used for data (or payload) communication. However, dedicated channels can result in poor utilization of the bandwidth – for example, in the modern NV-DDR4 interface, 10 out of the 18 pins are used for data/payload while the rest of the pins are dedicated for control and cannot be utilized for data communication. In comparison, this work proposes to exploit the available bandwidth by using "packets" for communication between the SSD channel controller and the flash memory chips, to enable a *packetized SSD* (pSSD) architecture.

Flash memory interconnects are fundamentally different from both traditional network-on-chip and large-scale system interconnection networks because of the difference in the packaging constraints [8]. In particular, flash memory are commodity components where the pin constraints are particularly limited and power constrained. As a result, traditional interconnection network architectures, including topology and flow control, are not applicable for the flash memory interconnect. In this work, we propose the microarchitecture to support packetized SSD (pSSD) system, including changes at both the SSD controller and the flash memory side. In particular, we leverage existing pins that are commonly used in flash-based SSD but repurpose them to communicate packets. While the interface logic does change, the internal flash behavior does not change in pSSD.

Our proposed pSSD provides approximately 2× improvement in flash channel bandwidth and improve overall performance. However, the bus-based topology for the flash memory interconnect does not provide connectivity between the flash memory chips and limits path diversity. As a result, we partition the channel bandwidth to propose the Omnibus topology – a 2D bus topology organization that continues to maintain a bus-based organization while providing connectivity between the flash chips. By introducing vertical bus channels, the Omnibus topology not only enables connectivity between the flash channels within the same column but we exploit the bandwidth available at the flash channel controller to control both a horizontal channel and a vertical channel. Given the Omnibus organization, we demonstrate how *spatial* garbage collection can be implemented where garbage collection can be done simultaneously, in space, with the I/O request handling of the SSD. In particular, main contributions are as follows:

- We propose *packet*-based SSD (pSSD) systems to enable higher effective flash channel bandwidth by communicating "packets" instead of dedicated signals to improve performance.
- We exploit packetized interface to create flash-to-flash connectivity that enables path diversity in the host-to-flash communication and minimize flash traffic contention. In particular, we propose Omnibus topology where both a vertical and a horizontal channel bus are managed by a single flash controller.
- We propose *spatial* garbage collection that takes advantage of flash-to-flash connectivity to enable GC to occur simultaneously as I/O and minimize interference between GC and I/O.
- Our evaluations show that pSSD results in up to 82.3% improvement in overall performance while reducing the tail-latency significantly by up to 18.7×.

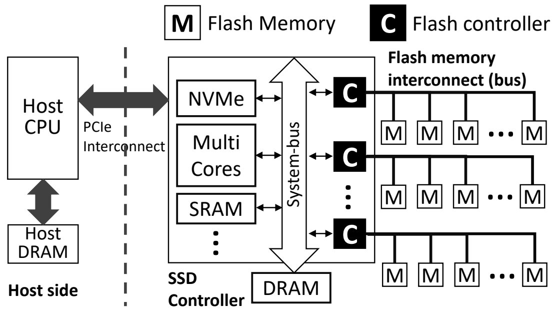


Figure 2: High-level block diagram of modern SSD architecture.

II. Background

In this section, we provide a background on modern SSD architecture and flash channel interconnect. We provide an overview of the flash channel controller and the flash interface between the controller and the flash memory itself.

A. SSD Architecture

Fig 2 shows a high-level overview block diagram of a SSD controller that consists of multiple sub-systems including NVMe host interface, multi-core subsystem that executes the FTL (Flash Translation Layer), DRAM, system-bus (e.g. AXI), and flash controller/bus with flash memory. In this work, we refer to **flash memory channel** or bus as the interconnect structure that is used to connect the flash controller and the flash memory devices. We also refer to this as **flash memory interconnection network** or flash memory interconnect since a collection of flash memory bus are used within a modern SSD architecture. The flash controller contains an ECC unit, internal page buffers, and a flash command control logic which includes a timing sequence generator for each control/data pin.

Symbols	Type	Description
CLE	Control	Command Latch Enable
ALE	Control	Address Latch Enable
RE	Control	Read Enable
RE_c	Control	Read Enable Complement
WE	Control	Write Enable
WP	Control	Write Protection
CE	Control	Chip Enable
R/B_n	Control	Ready/Busy
DQ[7:0]	Data I/O	Data Input/Outputs
DQS	Data I/O	Data Strobe
DQS_c	Data I/O	Data Strobe Complement

Table I: Flash interface signal description in Open NAND Flash Interface (ONFi) [35]

B. Flash Channel Interface

Modern SSD adopts multi-channel (or bus) architecture, and a flash memory channel is connected to multiple flash chips. Modern flash memory bus (or NAND flash interface) [35] such as NV-DDR4 operates asynchronously with multiple control and data pins as summarized in Table I. In the NV-DDR4 interface, there are 18 signals for communication but only 8 of them are used for payload or "data" itself (i.e., `DQ`) while the remaining signals are used a control signals. There are two signals `DQS, DQS_c` used as strobe signals for the data while the remaining 8 signals (`CLE, ALE, RE, RE_c, WE, WP, CE, R/B`) are used to differentiate the type of data being transmitted across `DQ` or to communicate other functionalities (e.g., write protect `WP`). For example, `CLE` (command latch enable) is asserted to signal that `DQ` contains the command information. Other control signals (e.g., `ALE` (address latch enable), `RE` (read enable), `WE` (write enable)) are used to appropriately specify the data type that is communicated through `DQ`.

The flash controller selects a flash chip target among multiple chips (or ways) using `CE` signal, and other control signals are used to communicate the flash I/O operation (e.g., Read/Write/Erase). In order to improve efficiency, command, address, and data are also sent through the `DQ` pins. However, given the large size of payload for data, most of the control signals are idle when payloads are being communicated. In this work, we propose to exploit the signal interface by communicating "packets" instead of leveraging dedicated signals to enable better sharing of the bandwidth or the pins.

C. Related Work

Interconnection Network: There have been many different topologies proposed in interconnection networks, including direct topologies (e.g., torus/mesh) and indirect topologies (e.g., fat-tree, folded-Clos) [8]. Network-on-chip (NoC) topologies have been proposed to minimize network diameters [24] [1] and recent topologies in large-scale networks have exploited high-radix switches to reduce network diameters [27] [25]. Network-on-SSD [38] was proposed to replace the flash memory bus with a conventional interconnection network topology (e.g., 2D mesh). While this work share similar goals, the key difference is that we argue that such multi-hop network is not practical in an SSD as we discuss in Sec V. Various bus topologies have been proposed for network-on-chip in multicore systems, including hierarchical bus topologies, segmented bus topologies, and various ring organizations [21] [39] However, to the best of our knowledge, no prior work has leveraged a multi-dimensional bus topology for an SSD flash interconnect system. Decouple SSD (dSSD) [23] was proposed to decouple the front-end of the SSD (i.e., flash controller) with the back-end (i.e., flash memory device) by introducing a network-on-chip to interconnect the flash controllers; however, dSSD assumes a bus structure for the flash memory interconnect.

Garbage Collection: Many prior works have been proposed to minimize I/O interference with GC [17], [29], [37], [41]. Preemptive GC [30] postpones GC when I/Os are serviced, and executes page copies for GC at a later time. Tiny-tail [41] exploits redundancy to avoid flash conflict for I/O read operations. PaGC (parallel GC) [37] maximizes internal flash bandwidth through multi-plane operations for GC. However, as the flash memory bandwidth increases, prior solutions increase flash-bus conflict even worse since more pages need to be transferred. The spatial separation of SSD for I/O and GC has been explored recently [22] on multiple SSDs; however, decoupling I/Os and GC within an SSD through a flash network has not been explored. Spatial GC proposed in this work shares some similarities to DRAM refresh management where some rows are refreshed while other rows respond to memory requests [34].

Packet-based communication: Packet-based communication has been commonly used in communication. Packetized interface has been adopted for different systems to improve communication efficiency, including on-chip networks [9], memory-semantic fabric such as GenZ [11], and CXL [7] to interconnect processors, accelerators, and memory. While GenZ and CXL provide efficient communication between the host and the I/O, they are not necessarily appropriate for internal chip-to-chip communication within an SSD because of the protocol overhead. In addition, the packaging constraints of the chip-to-chip constraints in an SSD present unique challenges, including limited pin interface and limited support for a hardware router or a switch.

III. Motivation

In this section, we describe the need for a packetized SSD (pSSD) architecture to effectively increase the flash channel bandwidth. As described earlier in Fig 1, flash memory bandwidth has continued to increase while the flash memory channel (or the interconnect between the flash memory and the flash controller) has not increased at the same rate. As a result, we discuss how a packetized interface can effectively increase the bandwidth. In addition, we show how a packetized SSD can enable flash-to-flash communication

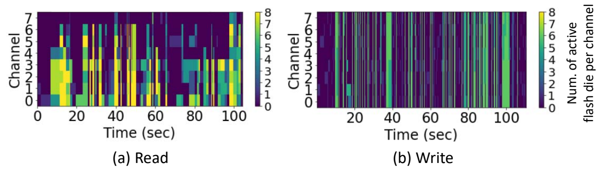


Figure 3: Analysis showing the imbalance of channel utilization for (a) read and (b) write access for an SSD system with 8 channels and 8 flash chips per channel on the Exchange-1 [28] trace. Write results in balanced usage while read accesses results in unbalanced accesses.

– providing an opportunity to minimize the interference between *internal* and *external* SSD traffic, as well as load-balancing across the different flash channels.

A. *Flash Channel Bandwidth & Packetized Interface*

The performance and capacity of an SSD have been increased by exploiting various levels of parallelism, including channel and chip parallelism, as well as parallelism within a flash chip. As parallelism improves capacity and performance, the amount of bandwidth per flash chip has also grown exponentially (Fig 1(a)). As the amount of bandwidth per chip increases and more chips are added to each flash channel, the amount of bandwidth from the flash channel has not increased at the same rate (Fig 1(b)). As a result, the flash channel bandwidth is becoming an important component in determining overall SSD performance. The flash channel bandwidth can be increased either through higher frequency or by adding additional pins (or channels). However, the scalability of both approaches are limited because of the power constraints and the pin-constraints. In this work, we explore how the *effective* bandwidth of the flash channel can be increased to improve overall performance (bandwidth) while enabling higher I/O performance for garbage collection interference by separating I/O and GC traffic.

In particular, instead of relying on a dedicated signal-based interface between the flash memory and the controller, we show how the packetized interface can effectively increase the amount of bandwidth with better utilization of the existing channels in an SSD system. Using packet-based communication is not new as it has been adopted in many domains, including large-scale systems and multicore network-on-chip noc, as well as system-on-chip architectures [2]. It is well-known that packet-based communication can improve scalability and improve performance. Thus, in this work, we explore how packetized communication can be leveraged to improve overall effective bandwidth within an SSD system for the flash memory interconnect.

B. *Flash-to-flash Connectivity*

The packetized SSD enables flash-to-flash connectivity that can provide significant benefits in the SSD system

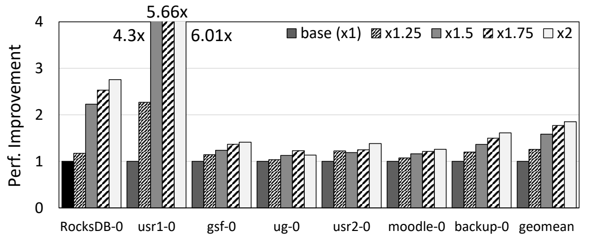


Figure 4: Performance improvement in different workloads as the flash channel bandwidth is increased by up to 2×.

for flash-to-flash communication. In particular, it allows ***potential isolation of internal and external traffic within the SSD***. In this work, we differentiate between SSD *internal* traffic and *external* traffic. We define traffic as *internal* traffic if the source and destination of the communication remain within the SSD itself (e.g., copy operations used during garbage collection) while *external* traffic is defined as when only the source or the destination (but not both) is within the SSD (e.g., I/O operations such as read or write). If these two type of traffic can be handled separately, it enables the I/O operations (external traffic) and the garbage collection (internal traffic) to be executed simultaneously while minimizing the interference between the two traffics. In Sec VI, we show how the packetized SSD with a "network" for the flash memory interconnect can provide such separation of traffic.

The flash-to-flash connectivity also enables path diversity that can be exploited when the flash channel bandwidth utilization is not well load-balanced across the channels. An example of the bandwidth imbalance is shown in Fig 3 where *x*-axis is time and *y*-axis indicates the different flash channels.[1] Only one representative workload trace (Exchange-1) is shown but other workloads follow similar trends. Within the trace, we separate out the read and the write accesses (or I/O operations). Write accesses show relatively load-balanced access across the different channels but the read accesses are shown to be imbalanced in access pattern.

The write accesses are load-balanced since the physical page addresses for write operations are determined by the FTL because of NAND flash "erase before program" characteristics – thus they are often distributed uniformly across the different channels (and different flash memory chips). In comparison, read access location is determined based on the address of the physical page where data was written and is determined by the workload characteristics. The load imbalance is not necessarily a new problem but the impact from the such imbalance was minimal when the bandwidth of flash memory itself was relatively low, compared to the flash channel bandwidth; however, as the latency/bandwidth of flash memory has improved with the

[1]Detailed evaluation setup is described in Sec VII. We assume an SSD system with 8 channels, with 8 flash memory chips per channel and the flash memory is modeled with ULL [5]

advancement of flash memory technologies (e.g., ultra-low latency [5]), the impact from the load-imbalance has become greater. Potential performance improvement from the higher flash channel bandwidth is shown in Fig 4 where we simulate the performance of various trace workloads as the flash interconnect bandwidth is increased. On average, 2× increase in bandwidth results in an 85% increase in performance; however, for some workloads with higher imbalance or higher utilization, the potential improvement can be up to 6×. In this work, we explore how the higher effective bandwidth from the packetized interface can provide such a performance improvement.

C. Challenges of pSSD

While potential benefits of packetized communication (and flash-to-flash connectivity) have been described, there are several challenges to enable the packetized SSD (pSSD) system. Some of the main challenges and opportunities can be summarized as follows:

- **Minimize overhead:** The overhead from packetized communication needs to be minimized, compared to other interconnection networks. For example, a conventional router or switch with deep buffers within flash memory is not feasible.
- **Network topology:** While packetized interface enables a network of interconnect flash memory, previously proposed network topologies are not applicable to the constraints of the pSSD system because of limited pin bandwidth of the flash memory.
- **Flash-to-flash channel utilization:** The pSSD-based system enables flash-to-flash connectivity but current systems cannot properly exploit such connectivity. We propose how *spatial* garbage collection can exploit such connectivity to provide significant improvement in tail-latency through separation of I/O and garbage collection traffic.

In the following sections, we describe our proposed packetized SSD architecture to minimize performance overhead (Sec IV), a 2D bus-based Omnibus topology organization to minimize network diameter while maintaining a similar communication interface as modern SSD flash chip (Sec IV), and exploit flash-to-flash communication to propose spatial garbage collection (Sec V).

IV. Packetized SSD Architecture

In this section, we describe our proposed packetized SSD (pSSD) architecture where we exploit *packetized* communication interface that are commonly used in packet-based communication to increase the effective bandwidth of the flash memory channels, compared to the current flash interface that relies on dedicated signals or pins. The packetized interface can increase the effective bandwidth by approximately 2× *without* introducing additional signals or increasing the signaling rate. We describe the architecture

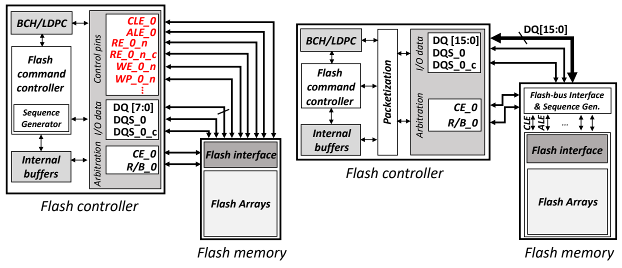


Figure 5: The block diagram of the interface for (a) signal-based conventional mechanism and (b) proposed packet-based method with modified flash controller/memory. For simplicity, only one flash memory is shown but multiple flash memory chips would share the controller and the data bus.

and the interface changes required to support the packetized interface, as well as the packet overhead; however, given the relatively large packet size in SSD (e.g., 16-64 kB page size), the overhead of packetized communication is relatively small. In this work, we define conventional flash memory interface as *dedicated* signaling as separate (dedicated) control signals/pins are used.

A. Packet-based Flash-interconnect

In this work, we observe that since a significant fraction of flash memory interconnect bandwidth is idle or not utilized when communicating payload, and propose a packet-based SSD interconnect that we refer to as packetized SSD (pSSD). As a result, most of the dedicated control signals are no longer needed but packets are communicated between the flash controller and the flash memory chip(s). The only control signals that are required in our proposed pSSD are the signals necessary for proper handshaking to determine who has access to the shared bus. Given that the interconnect is assumed to be a "bus," proper handshaking is necessary between the controller and the different flash memory chips connected to it. In our proposed pSSD, we leverage the existing `CE` (chip enable) signal to enable communication between a particular chip and the flash controller. In effect, all of the individual control signals described earlier are simply replaced with single enable control signals. Note that there is a separate `CE` for each memory chip that is connected to the bus. The flash controller also uses the `R/B` (ready or busy) control signal in our proposed packetized interface to communicate the status of each flash chip.[2]

Fig 6(a) illustrates how a *read* operation is communicated in conventional flash SSD system with multiple control pins. A flash controller first selects a flash die for the target page address through `CE_n`, and a read command (read command consists of two commands `00h` for the first command and

[2]Note that a flash memory die does not initiate data transfer, however, it is requested by the flash controller using `R/B` pin status.

30h for the second command [35]) is issued (via DQ[7:0]) by the flash controller while CLE is asserted. Then, column addresses are transmitted followed by the row address with ALE control signal asserted with the addresses. Finally, after issuing the second read command (30h), the flash memory reads a page and stores the page to an internal page register (t_R), and the page is read out by the flash controller through flash-bus for multiple cycles using read enable (RE_n) signals.

In comparison, the communication for a *read* operation in pSSD is shown in Fig 6(b). The pSSD uses two message types, control and data packets. A control packet is used to send the commands and addresses as necessary. The data packet includes page data with additional header information. Flash chip enable (CE) signal is first asserted such that the flash memory is ready to receive the packet. Then, the control packets that contain *read* command/addresses are transmitted to the destination flash memory chip. After a read flash operation is done (t_R), data packets ($D1, D2, ...$) are transmitted from flash memory to the flash controller. Difference from the signal-based page readout that uses RE_n signals, packetized SSD initiates RE_n signals to readout the internal page data through "read data transfer" command, a new flash command that read the page data with the packetization interface. The "read data transfer" command asserts RE_n signals internally and enables the flash memory to start transfer of the page data (or the payload).

B. pSSD Microarchitecture

The pSSD system requires changes to both the flash channel controller and the flash memory interface to support the packetized communication. We discuss the details of the changes required, as well as the overhead from using packets.

1) Flash Controller Microarchitecture: Conventional bus structure requires an arbitration logic that receives requests and generates the appropriate grant signal [8]. However, to reduce the complexity of such bus arbiter, we follow the same control logic as the baseline SSD flash channel bus where "arbitration" is effectively done by the flash channel controller through the hand-shaking with the CE and R/B pins. We effectively offload the timing sequence generator to the flash memory inside, and the flash controller generates a packet that contains a header to present whether it is a control or data packet.

As shown in Fig 5(b), the flash command controller remains the same; the main difference is the introduction of packetization, at the interface to introduce the appropriate packet header before driving the signals to the flash memory. By not utilizing the control signals that are used in the conventional (dedicated) signalling approach, the number of signals (or bandwidth) available for communication can be effectively increased by approximately 2×. In our packetized

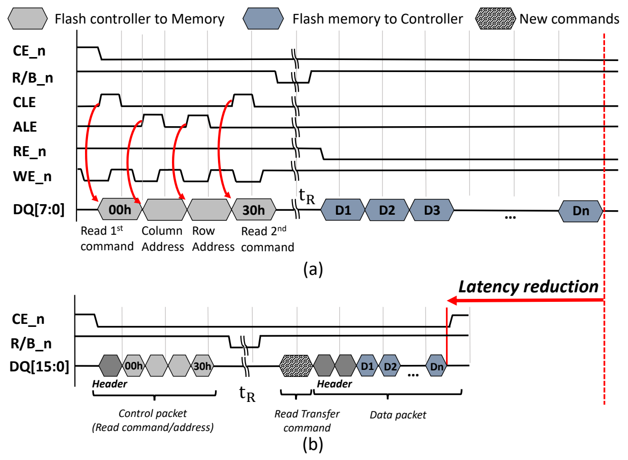


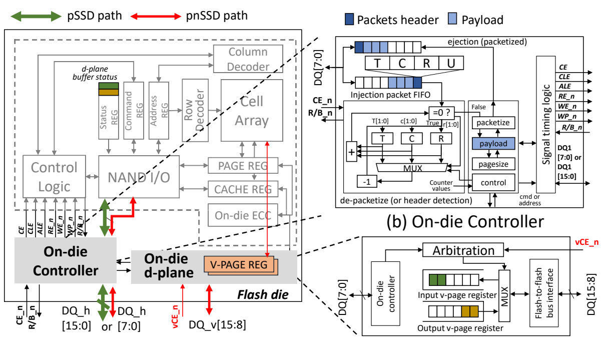
Figure 6: A timing diagram for a READ transaction for (a) conventional SSD with signal-based interface and (b) packet-based interface with pSSD.

SSD, we exploit the additional bandwidth to provide 2× improvement in the flash channel bandwidth. In Section V, we describe how this bandwidth can be partitioned to enable flash-to-flash connectivity.

2) Flash Memory Interface: A packet received by the flash memory needs to be properly interpreted. While the interface between the controller and the flash memory is modified with pSSD, the ***internal flash memory or organization is not modified*** as we introduce a controller logic between the external pins and the internal flash die. A high-level block diagram of flash memory for pSSD is shown in Fig 7(a), with a dotted-box around the baseline flash memory components that are not modified. pSSD introduces two additional hardware logic at the flash memory – an on-die controller and an on-die data-plane.

The role of the on-die controller is to interpret the packet header and generate the corresponding control signals to the flash memory – for example, if a READ packet is received, the read control signal (RE_n) will be asserted. The inputs to the on-die controller are the DQ pins as well as the handshaking pins. The on-die controller also has an internal FIFO queue that stores packets received from the flash channel controller and based on the commands received, a state-machine is used to properly generate the signals to the flash memory.

The on-die controller determines whether the packet is a control packet or a data packet using "Type" bits of the packet header as shown in Fig 8. If the packet is a control packet, the size of the packets is determined by the *T, C, R* bits that indicate the number of flits transmitted for the command itself and the column and the row addresses. One the command and the addresses are received by the on-die controller, they are effectively "forwarded" to the flash memory array cells using the same interface as the conventional



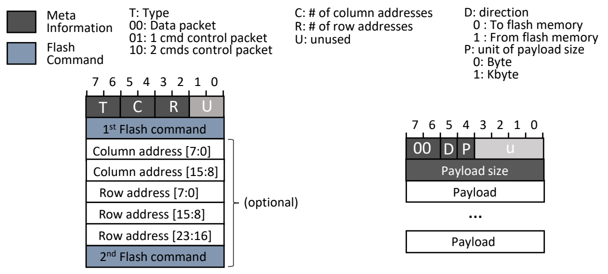
(a) Flash Memory Architecture (b) On-die Controller (c) On-die data-plane

Figure 7: (a) A high-level block diagram of the flash memory architecture for the packetized SSD, (b) the on-die controller that is added to provide support for the packetized interface, and (c) the on-die data-plane diagram that handles the packet data payloads as well as the flash-to-flash communication.

flash interface (or the dedicated control signals). As for the logic introduced to generate the control signals, the logic is very similar to the control logic that existed within the flash controller in the baseline SSD architecture. If the packet is a data packet, the on-die controller determines the size of the payload from the second flit. For read operations, the data (or payload) is read out from the page register and transmitted back to the flash controller.

In addition to the on-die controller, there is another logic added at the flash controller referred to as the on-die data-plane that includes additional buffers that we refer to as *V-page* registers. In the baseline pSSD, this logic is not needed; however, flash-to-flash connectivity is enabled with a network (Sec V), the on-die data-plane effectively serves as a "switch" for the flash-to-flash communication. Using the additional dedicated *V-page* registers, the data plane can provide a path for direct flash-to-flash communication.

3) Packet Overhead: An overview of control and data packets is summarized in Fig 8. For communication across 8-bit signals, we assume a flit (or a flow-control digit [8]) is 8 bits and a packet consists of one or more flits. For 16-bit channels, two flits are sent together simultaneously. Control packets include a different number of commands and column/row addresses depending on the flash command [35]. Data packets are also multi-flit packets that include payload size after the header flit. Both the control and data packet length is variable and the actual length is detected by the on-die controller using header information within the flash memory. However, in any packetized interface, there exists packet overhead since the unit of communication is packets, not individual wires. As shown in Fig 8, packet header needs to be the same size as a flit for both control and data packets – thus, only 6 out 8 bits are actually used and results in 25% overhead for the control packet header and 50% for the data packet header. However, the total overhead for flash-channel communication is relatively minimal because of the large



(a) Control packet format (b) Data packet format

Figure 8: (a) Control packet format that is composed of flash command and addresses, and (b) data packet format for pSSD.

packet size (e.g., 16-64 kB) compared to the packet header or the control packets that are communicated.

V. Flash-to-Flash Data Movement

Given the additional effective bandwidth from the packetized interface, the pSSD described in the previous section increased the flash channel interconnect bandwidth by 2×. An alternative approach is to keep the flash channel bandwidth the same as the baseline SSD but use the additional bandwidth to create a *network* to provide connectivity between the flash chips and created a packetized-network SSD (pnSSD). In particular, we propose to add a *vertical* flash-to-flash channel, that is exploited for parallel data-transfer and direct communication between flash chips[3]. In this section, we propose how the packetized SSD (pSSD) can be extended to provide flash-to-flash connectivity and enable data movement directly between the flash. In particular, we show how a 2D bus topology can be applied to enable flash-to-flash connectivity.

A. Limitation of Existing Network Topologies

Different topologies have been proposed for interconnection networks, including network-on-chips and large-scale systems [25] [21] [16] [24] [26]. *Indirect* networks such as Fat-tree or hierarchical topologies with intermediate switches are commonly used in high-performance computing; however, introducing an extra chip or a "router" in an SSD is not feasible. In comparison, *direct* networks such as mesh or torus networks are commonly used in network-on-chip [32] [21] [39] [33] [4]. However, multi-hop presents significant challenges in a network-on-SSD [38] as each additional hop results in a significant increase in not only latency but also cost (energy) from the I/O. In addition, the pin bandwidth (or the number of channels) available on a flash chip is rather limited, and partitioning it across multiple channels (for example, 4 channels in a 2D mesh) would significantly reduce the amount of bandwidth *per* channel,

[3]Flash chip-to-chip communication is only possible through vertical channel-bus in our implementation since we define horizontal bus to be used only for I/O handling while the vertical channels can be used for both I/O and chip-to-chip data movement.

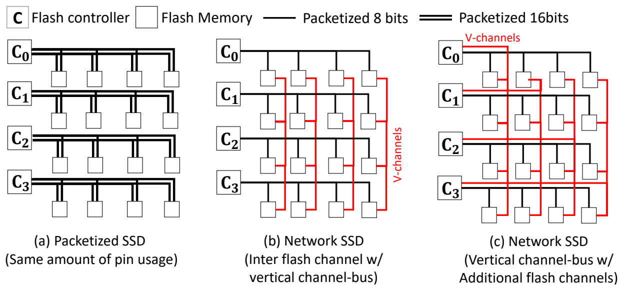

C Flash controller
Flash Memory
Packetized 8 bits
Packetized 16bits
V-channels
(a) Packetized SSD (Same amount of pin usage)
(b) Network SSD (Inter flash channel w/ vertical channel-bus)
(c) Network SSD (Vertical channel-bus w/ Additional flash channels)

Figure 9: Overview of different flash interconnects including (a) packetized SSD (pSSD) with "fat" (16-bit) channel, (b) channel sliced network with vertical bus channel to enable flash-to-flash connectivity, and (c) the proposed packetized *network* SSD (pnSSD) that provides connectivity of the flash controller to vertical channels.

even if the packetized interface is leveraged. In this work, we exploit the existing "bus" architecture that is used in modern SSD but scale it to two dimensions to create a 2D bus topology organization.

B. Omnibus Topology

In the pSSD architecture described in the previous section, the increased bandwidth from the packetized interface was leveraged to increase the flash channel bandwidth (Fig 9(a)). However, instead of dedicating the bandwidth to the *horizontal* flash channel, we partition the bandwidth to create a *vertical* bus channel (*v*-channel) as shown in Fig 9(b). As a result, all of the flash chips within the same row are connected while the flash chips within the same column are also connected. This provides direct connectivity between flash chips (or enables flash-to-flash communication) while maintaining the same total pin bandwidth compared to the baseline SSD.

The *v*-channel provides connectivity between the flash chips in the same column but has some fundamental challenges. For all SSD I/O traffic, the traffic originates from the host (and the flash controller) but by slicing the bandwidth, the bandwidth to/from the flash channel controller is reduced by $2\times$ – thus, reducing the overall I/O bandwidth by $2\times$, compared to pSSD (Fig 9(a)). In addition, usage of the vertical channels becomes a challenge. For example, most of the "control" for the horizontal bus came from the flash controller but that is not possible in Fig 9(b) as one of the flash chips in each column needs effectively act as a controller for each of the vertical channel buses (e.g., CE signals need to be asserted by some logic communicating between the flash memory chips).

As a result, we propose Omnibus [4] topology as shown in Fig 9(c). Omnibus builds on a 2D bus organization

[4]The proposed topology is analogous to a double-decker bus where there are two floors (or two "bus") but a single, shared driver, or a single controller

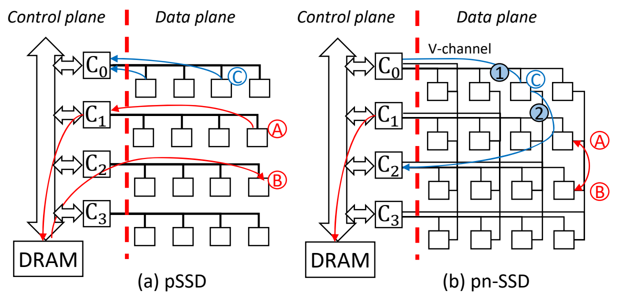


Figure 10: Block diagram showing the benefits of path diversity within the flash memory interconnect comparing (a) pSSD and (b) pnSSD with the Omnibus topology organization. In pSSD, movement from A to B has to be done through the flash controller and the internal DRAM while in pnSSD, direct connectivity is provided.

but Omnibus exploits the fact that the flash controller pin bandwidth is available with the packet interface, in Fig 9(b), to add extra connectivity. Thus, each flash controller can use the extra bandwidth to connect to one *v*-channel – and is responsible for a single *v*-channel, in addition to the horizontal flash channel bus. Unlike conventional interconnection networks where each node provides "control" (e.g., routing, arbitration, etc.), the proposed Omnibus topology enables a *split* architecture – where the control plane (or the flash channel controllers) is decoupled from the data plane (or movement of data across/between the flash chips).

C. Data Plane – Path Diversity

In this work, we refer to pSSD with flash-to-flash connectivity as packetized *network* SSD (pnSSD). Compared to the baseline, the pnSSD based on the Omnibus topology provides the *same* bandwidth from the flash channel controller as well as the flash memory chip compared to the pSSD. The main difference is that the bandwidth is partitioned – the flash chip bandwidth is partitioned across the horizontal and the vertical bus channel while the controller bandwidth is also partitioned across a single vertical and single horizontal channel. Compared to pSSD, pnSSD provides two advantages in terms of path diversity – direct connectivity for flash-to-flash communication and multiple paths when routing from the flash memory back to the flash controllers.

Examples of the potential benefits from the Omnibus organization for pnSSD are shown in Fig 10. If a page from Ⓐ needs to be copied to Ⓑ, in Fig 10, the page needs to be moved from the flash memory, back to the controller (and to the DRAM), before being copied to the destination in pSSD (Fig 10(a)). However, with pnSSD architecture, the data can be copied directly through the *v*-channels (Fig 10(b)). In addition, when data needs to be read from Ⓒ, in the baseline pSSD, there is only one path through the horizontal bus. However, with the Omnibus topology organization, the data from Ⓒ can be sent through the horizontal bus ① or it can be routed vertically before being routed horizontally as shown in ② to provide path diversity (Fig 10(b)).

The path diversity can also be exploited by *splitting* a message across the two paths. A page transfer from/to flash memory is split into two halves (or the number of paths) to utilize both the vertical and the horizontal channels. Since the network (or channel bandwidth) is partitioned, separating a packet to utilize the partitioned network is beneficial for both latency and bandwidth – and batch the bandwidth of `pSSD`.

D. Control Plane

As discussed earlier, the connectivity between the flash memory serves as the "data" plane for data movement. In comparison, the flash controller behaves as the "control" plane in pnSSD. While a network is created among the flash memory chips, each "node" in the network is mostly used for the movement of data, and common control logic (e.g., arbitration, routing logic, etc.) found in networks do not exist within the data plane. In comparison, the control logic is handled by the flash channel controllers to enable not only read/write commands but also ensure path diversity is exploited.

To enable flash-to-flash data movement, the control plane (or the flash controllers) need to properly control or manage the data plane and in particular, the *v*-channels, to ensure proper data movement between the flash memory chips. As described earlier, each flash controller is responsible for a single (different) *v*-channel. As a result, a given flash controller can play three roles within the control plane.

1) *Source* controller where the flash controller's *h*-channel is connected the source of the packet.
2) *Destination* controller where the flash controller's *h*-channel is connected the destination of the packet.
3) *Intermediate* controller where the flash controller's *h*-channel is not connected to either the source or the destination of the packet; however, the packet needs to leverage the *v*-channel of the flash controller.

An example of control plane for these three different cases is shown in Fig 11, using flash controller 0 ($C0$) as an example. [5] We assume $C0$ is responsible for the *v*-channel shown and $C0$ can be the source controller (i.e., source flash memory is connected to the *h*-channel that is connected to $C0$), the destination controller, or an *intermediate* controller such that it is not directly connected to either the source or the destination – but is responsible for the *v*-channel that connects the source to the destination. In Fig 11(a), we assume $C0$ is the source and the destination is $C1$. The control plane (through on-chip communication within the SoC controller) works with $C0$ sending a request to $C1$ to determine the status of the on-die data-plane buffer status. $C1$ checks the buffer status and once the buffer is available,

[5]For simplicity, only a single column of flash memory is shown.

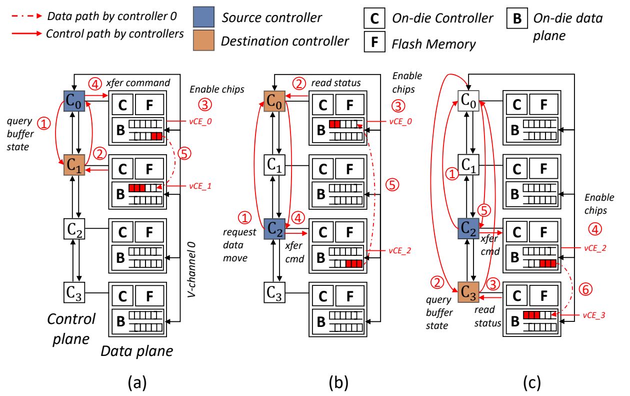


Figure 11: Illustration of how the control plane works between the flash channel controllers. For simplicity, only one column of flash memory chip is shown. Three different arbitration cases when controller $C0$ is (a) source, (b) destination, and (c) an intermediate controller.

a grant is sent back to $C0$. $C0$ then accesses the *v*-channel 0 by asserting both `vCE0` and `vCE1` (*v*-channel Chip Enable) signals and a page transfer (`xfer`) command is issued to flash memory for the direct chip-to-chip data movement.

In Fig 11(b), we assume $C0$ is the destination while $C2$ is the source. Once a request is received from $C2$, the role of $C0$ is to ensure that the chip enables (`vCEs`) are asserted after on-die buffer status is identified from the flash memory connected to $C0$. Then, the vertical channel can be leveraged through a data transfer command from $C2$. In Fig 11(c), $C0$ is responsible for a flash memory that is neither the source nor the destination but is simply an intermediate node (i.e., the source is $C2$ and destination is $C3$ but communicates through the *v*-channel that is controlled by $C0$). The request, in the control plane, starts in $C2$ and goes through $C0$ before reaching $C3$ – this ensures both source and destination chips are properly enabled for the communication within the network connected pnSSD architecture.

E. Routing/Deadlock and Scalability

While the data plane in pnSSD provides path diversity, the routing is always minimal routing and a single-hop from the controller to the flash memory. As a result, while either the horizontal or the vertical channel might be utilized, the communication packet never "holds" on to any resources (e.g., channels or intermediate buffers [8]) If the 2D bus structure is viewed as two dimension with h dimension and a v dimension, the routing algorithm is effectively a dimension-ordered routing [8] since h is always routed before v – as a result, cyclical dependency does not exist, and routing deadlock cannot occur.

In this work, we assume 8 flash-bus channels, each with 8 ways for the SSD system organization. However, for a non-square organization (e.g., 4 channels with 4 controllers but 8 ways), each controller needs to be responsible for two

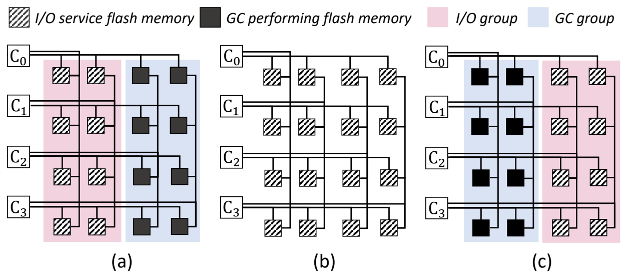


Figure 12: Illustration of how spatial GC works. When GC is first triggered, (a) the flash memories on the left are I/O group, continuing to service I/O requests while the flash memory on the right (GC group) performs GC. Once GC is finished, (b) all flash memories are used for I/O. When the next GC is triggered, (c) the GC and I/O group are swapped such that the other half of flash memories go through GC.

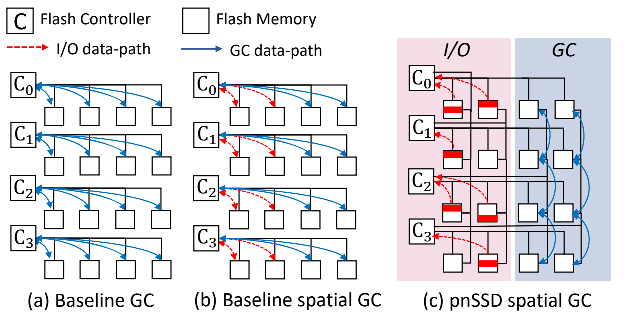


Figure 13: (a) Baseline parallel garbage collection (GC), (b) spatial GC on the baseline flash architecture without direct connectivity between the flash memory chips, and (c) spatial GC on pnSSD that has connectivity between flash memory chips.

columns – thus, a single *v*-channel bus will need to inter-connect two columns (or 2-ways) of the flash chips that are connected by the horizontal bus. If there are more channels than ways (e.g., 8 channels, 4 ways, and 8 controllers), half of the controllers only control single *h*-channel while the remaining controllers control both *h* and *v*-channels. Thus, as the size of the network increases, the Omnibus can also be scaled accordingly – as long as the bus can support the number of ways required in the system.

VI. SPATIAL GARBAGE COLLECTION

In this section, we propose *spatial* garbage collection (SpGC) that takes advantage of direct connectivity within the packetized-network SSD (pnSSD). In particular, SpGC enables garbage collection to occur simultaneously with I/O requests but more importantly, by physically separating the region of flash memory, the interference between the GC and I/O can be greatly reduced while significantly reducing the I/O tail latency.

As the capacity of SSD increases, more flash memory chips (or ways) are attached to a flash memory channel. As a result, the aggregate bandwidth of the flash memory connected to a flash memory channel can *exceed* the bandwidth of the bus. Because of the higher (internal) flash memory bandwidth, all of the flash memory connected to a flash memory channel does not need to be fully utilized to maximize flash memory channel utilization. In this section, we propose *spatial GC* that services I/Os through a subset of flash memory across the different flash memory channels while GC is executed concurrently using the remaining subset of flash memory. Note that the spatial GC can be implemented on a conventional SSD architecture; however, as we show later in Sec VII. the benefit of spatial GC is very limited because of the flash memory channel contention.

A. *I/O & GC Groups*

To support spatial GC, we define an *I/O group* as a collection of flash memory chips that services I/O requests for a period of time (or an epoch) and similarly, an *GC group* as a group flash memory that performance GC. To effectively utilize flash-bus channel bandwidth, an activation group is composed of adjacent flash memory across all of the different channels. An example of the two groups is shown Fig 12(a) for an SSD system that consists of 16 flash chips. The left side 8 flash chips are grouped together as part of the I/O group while the other half is grouped to create a GC group. As a result, while the I/O group handles I/O requests, the GC group performs GC through the flash-to-flash connectivity in parallel – thus, minimizing the interference between GC and I/O. By partitioning the flash memory into groups, the I/O operations are separated from GC but I/O is still able to fully exploit flash channel parallelism and path-diversity of pnSSD.

Initially, all flash memory is utilized by the FTL until GC is triggered. When GC begins, flash memories in the GC group start to copy valid pages to another flash memory chip located within the GC group. The FTL finds victim blocks from the flash memory within the GC group and to avoid *h*-channel contention with I/O traffic, the destination free blocks are restricted to the flash memory in the same way (or column). After GC is finished, the FTL returns to utilize all of the flash memories for the incoming I/Os (Fig 12(b)). When the next iteration of GC is triggered, FTL swaps the GC group and the I/O group (Fig 12(c)) – to uniformly increase the age (or P/E cycles) of the flash memory.

During GC with spatial GC, the interference between GC and write I/O operations are completely removed since the physical location for write operations is determined by the FTL and can be allocated to the I/O group (and not the GC group). However, interference with I/O group cannot be completely removed since if the address of the read

Simulation	Parameters
Simple-SSD organization	PCIe 4.0 x4 lanes, system-bus=8GB/s ($\times$1), DRAM=8GB/s 8 channels 8 ways 1 die, 4 planes 1024 blocks 512 pages
baseline	flash bus transfer-rate = 1000MT/b/s, width = 8 bits
pSSD	flash bus transfer-rate = 1000MT/b/s, width = 16 bits
pnSSD	# of v-channels = 8, v-channel width = 8bits
Flash Memory	read=3us, write=50us, erase=1ms, page size=16KB

Table II: Simulation parameters

Acronym	Description
baseSSD	Conventional SSD
NoSSD (pin-constraint)	Network-on-SSD [38] with 2bit channel on mesh
NoSSD (no constraint)	Network-on-SSD [38] with 8bit channel on mesh
pSSD	Packetized SSD (Sec IV)
pnSSD	pSSD with Omnibus topology (Sec V)
pnSSD (+split)	Split technique is applied on pnSSD

Table III: Description of different SSD architectures evaluated.

operation is located within the GC group, there will be interference between the I/O and GC. For simplicity, we partition the flash memory into two equal size groups for I/O and GC; however, the size of the GC group can be reduced – e.g., 1/4 of the flash memory can be GC group while 3/4 of the flash memory can be part of I/O group. This partition can lead to more frequent GC but potentially at the benefit of improved read performance.

B. Concurrent I/O and GC

The difference between baseline GC (parallel GC) and the proposed spatial GC is shown in Fig 13. When parallel GC is used, all flash memory can perform GC to simplify the GC process (Fig 13(a)); however, I/O cannot be serviced during GC. In comparison, spatial GC described earlier can be implemented on the baseline SSD architecture (Fig 13(b)). Baseline spatial GC enables the flash memory conflicts to be avoided by spatial separation – however, I/O and GC utilize the same flash memory channels and the conflict moves from the flash memory to the flash memory channels to create interference. In comparison, pnSSD with the Omnibus topology can minimize the interference between I/O and GC by leveraging the vertically connected channels (*v*-channel) (Fig 13(c)). I/O requests are serviced not only by horizontal channel but can also leverage the *v*-channels on the left side or within the I/O group. At the same time, page copies for GC are executed by the right GC side of the flash memory through *v*-channels within the GC group. As a result, the I/O requests and GC execute concurrently while minimizing interference.

VII. Evaluation

A. Methodology

To evaluate the benefits of packetized SSD, we used a SimpleSSD-standalone [12] simulator that models NVMe, FTL, and flash-bus/memory. We used representative workload traces [40] [3] and selected workloads with different ratios of read/write accesses and different access patterns. We modified the simulator to implement the Omnibus topology as described earlier in Sec V as well as the packetized interface. In addition, we modified GC victim selection and free page allocation in the FTL to model spatial GC. We used all flash memory for GC in parallel and the number of victim blocks in spatial GC is the same as the baseline.

The baseline GC that we compare against is PaGC [37]. We assume I/O and GC group consists of 32 flash memory chip. Since only half of the flash memory is used for GC with SpGC, the number of victim blocks per flash die is increased by $2\times$ to match the total number of selected victim blocks as the baseline GC. For the baseline GC, greedy policy is used to select the victim block and the free block selection is done randomly across the 64 flash memory chips. Details of other parameters used in the evaluation are summarized in Table II. We used ULL (Ultra Low-Latency) parameters [5] for the flash memory parameters and the bandwidth of NVMe, system-bus, and DRAM are provisioned to equal the total flash bus channel bandwidth to provide sufficient bandwidth for each component within the SSD controller system.

The different SSD architectures that we compare in this work are summarized in Table III, which includes the `pSSD` and `pnSSD` with the Omnibus topology, as well as the conventional baseline SSD (`baseSSD`) with conventional bus structure for the flash memory interconnect and dedicated control signaling. We also compare with NoSSD [38] where a 2D mesh network was used to interconnect the flash memory device. We evaluate two implementation of `NoSSD` – NoSSD (pin-constraint) where the pin constraints are held constant compared to all of the alternative designs compared in Table III, resulting in 2-bit data channels, as well as NoSSD (no constraint) without pin constraints such that each *channel* bandwidth is constant compared to `baseSSD` (i.e., 8-bit data channels). For these architectures, we also add `SpGC` to evaluate the impact of spatial GC. We also evaluate the performance impact from `split` that was described earlier in Sec V-C.

B. I/O Performance

We first evaluate the performance benefits of `pSSD` and `pnSSD`. To understand the general I/O performance improvement, we first simulated without triggering GC while executing workload traces [40] [3]. Fig 14 shows the average I/O latency improvement normalized to `baseSSD`. `pSSD` and `pnSSD` shows 69% and 60% improvement, compared to the `baseSSD`. The packetized SSD (`pSSD`) shows slightly better performance than simple `pnSSD` (9%) on average. Both `pSSD` and `pnSSD` provide the same amount of bandwidth to the flash controller; however, while `pSSD` uses the entire bandwidth for the *h*-channels, `pnSSD` partitions the bandwidth between the *h*-channel and the *v*-channel. `pSSD` achieves higher utilization of the flash memory channels,

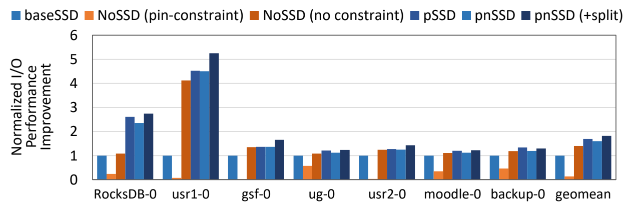


Figure 14: Normalized I/O performance (average I/O latency) improvement in workload trace when garbage collection does not occur.

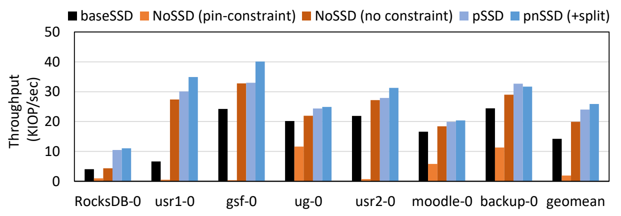


Figure 15: Throughput comparison of the different SSD architecture.

compared to pnSSD, since an *adaptive* decision needs to be made for pnSSD – determining whether to use the *h*-channel or route through the *v*-channel. In this work, we assume a greedy approach where the first available channel is used – thus, it can lead to non-optimal routing decision. It remains to be seen if an intelligent adaptive algorithm can be exploited to improve channel utilization and overall performance. The performance of pnSSD can be improved with *split* (Sec V-C) where a page is split in half and transmitted across both paths. As a result, pnSSD (+split) results in 82% performance improvement – exceeding the performance of pSSD (by 13%) through improving channel utilization of the flash channel interconnect and effectively reducing the I/O latency.

We also compare to Network-on-SSD [38] (NoSSD). Since a 2D mesh topology requires 4 bidirectional channels, the amount of bandwidth *per* channel is significantly reduced when considering the pin-constraints. As a result, NoSSD (pin-constraint) results in approximately 4× performance degradation compared to baseSSD on average. To understand the benefit of path-diversity from the mesh topology, the channel bandwidth of NoSSD is (unrealistically) increased to be the same as pnSSD (i.e., 8 bits per channel) shown as NoSSD (no-constraint) – resulting in significant performance improvement compared to NoSSD (pin-constraint) and exceeds baseSSD I/O performance by 40%. However, even though NoSSD (no-constraint) effectively increases the pin bandwidth by 4× compared to pnSSD, pnSSD is able to improve performance over NoSSD (no-constraint) by 30% since the 2D mesh topology is edge asymmetric topology – thus, the performance bottleneck are the mesh channels near the flash controllers since all I/O traffic source (or destination) is the flash controller.Fig 15 shows throughput (KIOPS) results for the trace workloads. On average, pSSD and pnSSD (+split) outperform NoSSDs by resulting in 69%, and 82% improvement in throughput, comapred to baseSSD. pnSSD (+split) shows 13.5× higher throughput compared to NoSSD (pin-constraint).

Synthetic Analysis: To further understand the performance benefit of the pSSD and pnSSD, we evaluated latency and bandwidth using synthetic workloads by increasing the number of I/O requests. An I/O request is 64KB and multi-plane commands are used to maximize flash memory bandwidth. Fig 16 shows the results for sequential and random access pattern for both read and write accesses. The *x*-axis is the number of concurrently running I/Os on SSD and it is increased up to 64 which matches the total number of flash memory devices in our evaluation.

The interleaving of accesses across the flash memory depends on the page allocation policy of the FTL. PCWD (plane, channel, way, die) is assumed for the results in Fig 16 where channel-level parallelism is prioritized. Since the allocation policy results in load-balanced access across the channels, there is no load-imbalance and thus,pSSD provides the best performance (or the lowest latency) – approximately 2× reduction in latency compared to baseSSD. pnSSD (+split) also has limited benefit, compared to pnSSD since the accesses are load-balanced. However, for the PWCD interleaving that prioritizes ways (or dies in the same channel) (Fig 17), pnSSD (+split) results in not only the same latency as pSSD, but also shows better performance when the number of concurrent I/Os is less than 32 since pnSSD can load-balance by exploiting the path-diversity. As a result, pnSSD performance is less sensitive to the access pattern (or page allocation scheme) because of its ability to load-balance. NoSSD achieves similar performance as pnSSD when the number of I/O requests are small. However, this assumed approximately 4× higher bandwidth was available for NoSSD compared to pnSSD. As the number of concurrent I/O requests increases, NoSSD suffers from network congestion near the flash controllers and results in significantly higher latency.

C. I/O and GC Interference

The benefit of the pnSSD over pSSD is in the decoupling of I/Os from the GC. Fig 18 shows I/O performance improvement when GC is performed (or triggered) while I/Os are being serviced. The benefits of spatial GC is relatively small when used with baseSSD (up to 16% improvement) or pSSD (59% for Read, 95% for Write) since the flash memory bus channel is shared between GC traffic and I/O traffic. However, pnSSD (SpGC) shows much higher performance improvement – approximately 5× increase on average since GC path is isolated from the I/O access path.

We evaluate the impact of GC interference on real workload traces in Fig 19 and compare against alternative GC

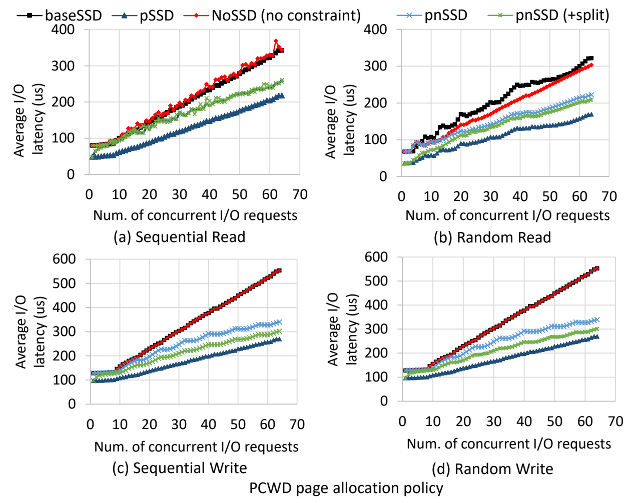


Figure 16: Sequential and random Read/Write synthetic workload performance with Plane-Channel-Way-Die (PCWD) page allocation scheme that balances I/O traffic across the different flash memory channels.

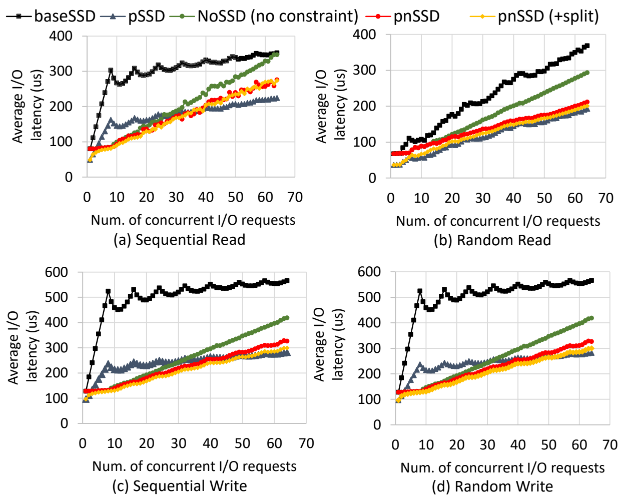


Figure 17: Sequential and random Read/Write synthetic workload performance with Plane-Way-Channel-Die (PWCD) page allocation scheme that creates more unbalanced I/O traffic across the flash memory channels.

algorithms, including parallel GC (PaGC) [37], preemptive GC [29], and spatial GC that we propose in this work. Fig 19 shows how pnSSD improves I/O performance over the baseSSD and pSSD by 9.7*times*, 5.9*times* on average, respectively. In general, preemptive GC performs well if there are "idle" times when there are no I/O requests – for such workloads, the performance benefit can be significant. However, for other workloads, preemptive GC needs to find such idle time and if it is not available, GC cannot be postponed indefinitely and thus, cause performance degradation for I/O traffic. The pnSSD architecture benefits both preemptive GC as well as SpGC. However, SpGC is able to provide better isolation between GC traffic and I/O traffic – and thus, on average, SpGC exceeds the performance of preemptive GC by 47%.

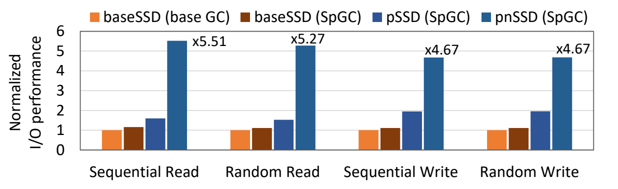


Figure 18: I/O performance improvement normalized to baseSSD with PaGC, on synthetic evaluations when GC is triggered.

Fig 20(a) shows the tail-latency results of the RocksDB-0 trace, and at 99-percentile, the pnSSD (SpGC) significantly outperforms baseline and pSSD (SpGC) by reducing tail-latency 18.7×. Even though pnSSD attempts to separate the GC traffic from the I/O traffic, there is still some interference when the read I/O destination is located within the GC group during spatial GC. As a result, such unavoidable interference results in longer latency from the flash memory conflict itself and leads to some increase in the tail latency.

We also measured GC elapsed time, and the GC time does not necessarily impact the I/O performance in pnSSD (unlike baseSSD), however, reducing the GC time is preferred to minimize flash conflict from unavoidable read I/Os. The number of page copies is doubled for spatial GC since only half of the flash memories are used for GC. However, baseSSD (SpGC), pSSD, and pnSSD results in lower GC time compared to baseSSD (Fig 20(b)). The reason for the improvement (or the reduction of) GC time for baseSSD (SpGC) is mainly from the reduced flash memory channel-bus contention for page copies itself, since half of the flash memory copies through the flash channels while the other half services I/Os. In addition, the increased effective bandwidth of the flash-bus channel reduces flash-bus contention as shown in the pSSD (SpGC). More interestingly, the pnSSD utilizes a quarter of the total channel bandwidth for GC since pnSSD only uses vertical channels on the half of the flash memory. However, the GC time is also reduced because the number of data-transfer is also reduced by half by directly communicating between flash memory (instead of communicating through the flash controller).

VIII. DISCUSSION

Cost: The two components that are modified in pnSSD are the flash channel controller and the flash on-die controller. The flash channel controller logic is simplified since the logic to generate signals (e.g, ALE, RE, etc.) is moved to the on-die controller and replaced by packet interface logic.

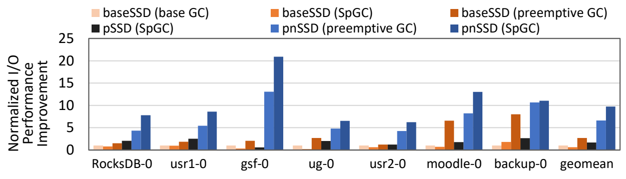


Figure 19: Average I/O performance comparison on different SSD architectures with baseline GC, preemptive GC, and spatial garbage collection.

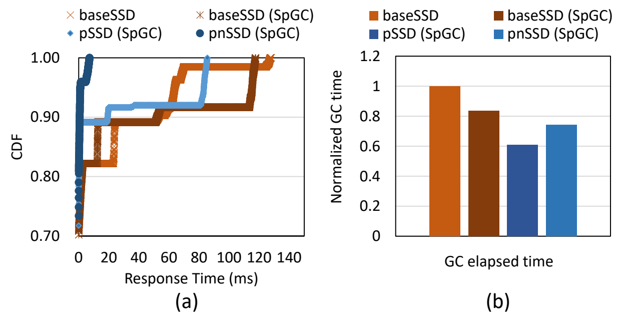


Figure 20: (a) Tail latency comparison across different SSD implementations for the RocksDB-0 trace and (b) average garbage collection (GC) execution time across all of the traces.

The main source of cost (overhead) in the flash memory side is the on-die controller and for pnSSD, the "on–die d-plane" logic that is dominated by the extra register added ($v-page$ buffer in Figure 7(c)) that requires two additional 16kB page buffers. Given that the recent flash memory die is approximately $100mm^2$ (66% for cells, 20% for peripheral circuits, and 14% for the page buffers) [18], we estimate that the overhead of the on-die controller logic can be approximately 20% increase in area, which includes the additional logic summarized earlier in Figure 7). However, recent flash cells are 3D devices such that peripheral logic is placed underneath the cells through Peri Under Cell (PUC) or Cell over Peri (COP) technique [15] – thus, the increase in the logic that we introduce from pnSSD does not necessarily impact overall flash memory size for PUC devices as the additional logic can be hidden under the cell structures [15] [36].

On-die ECC functions: When page data is directly transferred from a flash chip to a flash chip, one potential problem is error correctness. In modern SSD, the ECC engines are often located within the flash controller [42] and thus, error propagation can occur. On-die (flash chip) ECC ability has been introduced earlier to exploit internal copy-back operations [14]; however, as the flash memory became more unreliable with the multi-level cell flash, on-die ECC was no longer effective [42]. To enable flash-to-flash page copy without sending data through the ECC logic in the flash controllers, one potential solution is hierarchical ECC that implements strong ECC logic (e.g., LDPC) in the flash controllers while offloading weaker error detection logic to on-die ECC to enable a cost-efficient solution for flash-to-flash data movement – effectively realizing a hybrid ECC [13].

Impact on FTL complexity: The Omnibus topology (or pnSSD) does not impact the FTL since once the logical-to-physical address translation is done, the packets are "routed" with the pnSSD architecture. Since one of the main roles of FTL is garbage collection (GC), spatial GC (SpGC) requires some support from the FTL. In particular, the FTL needs to be aware of which group of chips are used for I/O (i.e., activation groups) and which group is used for GC. In addition, a write policy needs to be modified such that writes are restricted to the activation group with SpGC. As a result, the proposed pnSSD (and SpGC) has minimal impact on the FTL complexity.

IX. Conclusion

In this work, we proposed packetized SSD (pSSD) to effectively utilize flash channel bandwidth by converting designated signal-based interface to a packet-based communication where "packets" instead of dedicated control/wires are used for communication between the flash channel controllers and the flash memory chips. Based on pSSD, we enable flash-to-flash connectivity to create a packetized *network* SSD (pnSSD. We exploit the connectivity through a 2D bus-based topology and propose the *Omnibus* network organization for pnSSD. The pnSSD enables spatial garbage collection (SpGC) that minimizes I/Os and GC interference by separating I/O and GC data path. Our evaluations show that pnSSD results in 82% I/O performance improvement when there is no GC and improvement of 9.71× average I/O latency when there is interference between GC and I/O traffic.

Acknowledgment

We would like to thank the anonymous reviewers for their valuable comments. This work was supported in part by IITP grant funded by MSIT (No. 2020-0-01303, 2020-0-01309) and in part by NRF-2020R1A2B5B0100168713.

References

[1] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *ACM International conference on supercomputing 25th anniversary volume*, 2006, pp. 390–401.

[2] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] D. Campello, H. Lopez, R. Koller, R. Rangaswami, and L. Useche, "Non-blocking writes to files," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 151–165.

[4] Y.-Y. Chang, Y. S.-C. Huang, M. Poremba, V. Narayanan, Y. Xie, and C.-T. King, "Ts-router: On maximizing the quality-of-allocation in the on-chip network," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2013, pp. 390–399.

[5] W. Cheong, C. Yoon, S. Woo, K. Han, D. Kim, C. Lee, Y. Choi, S. Kim, D. Kang, G. Yu *et al.*, "A flash memory controller for 15μs ultra-low-latency ssd using high-speed 3d nand flash with 3μs read time," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2018, pp. 338–340.

[6] J. Cho, D. C. Kang, J. Park, S.-W. Nam, J.-H. Song, B.-K. Jung, J. Lyu, H. Lee, W.-T. Kim, H. Jeon *et al.*, "A 512gb 3b/cell 7 th-generation 3d-nand flash memory with 184mb/s write throughput and 2.0 gb/s interface," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 426–428.

[7] CXL, "Compute express link," https://www.computeexpresslink.org/download-the-specification.

[8] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* San Francisco, CA: Morgan Kaufmann, 2004.

[9] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 2001, pp. 684–689.

[10] N. Express, https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4-2019.06.10-Ratified.pdf.

[11] GENZ, "Genz," https://genzconsortium.org/.

[12] D. Gouk, M. Kwon, J. Zhang, S. Koh, W. Choi, N. S. Kim, M. Kandemir, and M. Jung, "Amber*: Enabling precise full-system simulation with detailed modeling of all ssd resources," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE, 2018, pp. 469–481.

[13] C.-C. Ho, Y.-P. Liu, Y.-H. Chang, and T.-W. Kuo, "Antiwear leveling design for ssds with hybrid ecc capability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 488–501, 2016.

[14] D. Hong, M. Kim, J. Park, M. Jung, and J. Kim, "Improving ssd performance using adaptive restricted-copyback operations," in *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA).* IEEE, 2019, pp. 1–6.

[15] H. Huh, W. Cho, J. Lee, Y. Noh, Y. Park, S. Ok, J. Kim, K. Cho, H. Lee, G. Kim *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 *2020 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2020, pp. 220–221.

[16] A. Joshi, B. Kim, and V. Stojanovic, "Designing energy-efficient low-diameter on-chip networks with equalized interconnects," in *2009 17th IEEE Symposium on High Performance Interconnects.* IEEE, 2009, pp. 3–12.

[17] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "Hios: A host interface i/o scheduler for solid state disks," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 289–300, 2014.

[18] D. Kang, M. Kim, S. C. Jeon, W. Jung, J. Park, G. Choo, D.-k. Shim, A. Kavala, S.-B. Kim, K.-M. Kang *et al.*, "13.4 a 512gb 3-bit/cell 3d 6 th-generation v-nand flash memory with 82mb/s write throughput and 1.2 gb/s interface," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2019, pp. 216–218.

[19] D.-H. Kim, H. Kim, S. Yun, Y. Song, J. Kim, S.-M. Joe, K.-H. Kang, J. Jang, H.-J. Yoon, K. Lee *et al.*, "13.1 a 1tb 4b/cell nand flash memory with t prog= 2ms, t r= 110μs and 1.2 gb/s high-speed io rate," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2020, pp. 218–220.

[20] ——, "13.1 a 1tb 4b/cell nand flash memory with t prog= 2ms, t r= 110μs and 1.2 gb/s high-speed io rate," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2020, pp. 218–220.

[21] H. Kim, G. Kim, S. Maeng, H. Yeo, and J. Kim, "Transportation-network-inspired network-on-chip," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2014, pp. 332–343.

[22] J. Kim, K. Lim, Y. Jung, S. Lee, C. Min, and S. H. Noh, "Alleviating garbage collection interference through spatial separation in all flash arrays," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 799–812.

[23] J. Kim, M. Jung, and J. Kim, "Decoupled ssd: Reducing data movement on nand-based flash ssd," *IEEE Computer Architecture Letters*, vol. 20, no. 2, pp. 150–153, 2021.

[24] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007).* IEEE, 2007, pp. 172–182.

[25] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture.* IEEE, 2008, pp. 77–88.

[26] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: a cost-efficient topology for high-radix networks," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 126–137.

[27] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high radix router," in *32nd International Symposium on Computer Architecture (ISCA'05).* IEEE, 2005, pp. 420–431.

[28] M. Kwon, J. Zhang, G. Park, W. Choi, D. Donofrio, J. Shalf, M. Kandemir, and M. Jung, "Tracetracker: Hardware/software co-evaluation for large-scale i/o workload reconstruction," in *2017 IEEE International Symposium on Workload Characterization (IISWC).* IEEE, 2017, pp. 87–96.

[29] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2011, pp. 12–21.

[30] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 3, pp. 18–es, 2007.

[31] S. Lee, C. Kim, M. Kim, S.-m. Joe, J. Jang, S. Kim, K. Lee, J. Kim, J. Park, H.-J. Lee *et al.*, "A 1tb 4b/cell 64-stacked-wl 3d nand flash memory with 12mb/s program throughput," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 340–342.

[32] Z. Li, J. San Miguel, and N. E. Jerger, "The runahead network-on-chip," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 333–344.

[33] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 196–207.

[34] J. Mukundan, H. Hunter, K.-h. Kim, J. Stuecheli, and J. F. Martínez, "Understanding and mitigating refresh overheads in high-density ddr4 dram systems," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 48–59, 2013.

[35] ONFI, "Open NAND Flash Interface Specification rev 4.2"," http://www.onfi.org/specifications, Feb 2020.

[36] J.-W. Park, D. Kim, S. Ok, J. Park, T. Kwon, H. Lee, S. Lim, S.-Y. Jung, H. Choi, T. Kang *et al.*, "A 176-stacked 512gb 3b/cell 3d-nand flash with 10.8 gb/mm 2 density with a peripheral circuit under cell array architecture," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 422–423.

[37] 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 *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 561–572.

[38] A. Tavakkol, M. Arjomand, and H. Sarbazi-Azad, "Network-on-ssd: A scalable and high-performance communication design paradigm for ssds," *IEEE Computer Architecture Letters*, vol. 12, no. 1, pp. 5–8, 2012.

[39] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian, "Towards scalable, energy-efficient, bus-based on-chip networks," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.

[40] G. Yadgar, M. Gabel, S. Jaffer, and B. Schroeder, "Ssd-based workload characteristics and their performance implications," *ACM Transactions on Storage (TOS)*, vol. 17, no. 1, pp. 1–26, 2021.

[41] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, and H. S. Gunawi, "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in nand ssds," *ACM Transactions on Storage (TOS)*, vol. 13, no. 3, pp. 1–26, 2017.

[42] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "Ldpc-in-ssd: Making advanced error correction codes work effectively in solid state drives," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 243–256.