# A Survey of SSD Simulators and Emulators

Atiyeh Gheibi-Fetrat[1*], Fatemeh Serajeh Hassani[1], Masoud Mohammadi-Lak[1], Amir Mirzaei[1], Negar Akbarzadeh[2], Mahmoud Reza Kheyrati-Fard[1], Mohammad Hosseini[1], Ahmad Javadi Nezhad[1], Arash Tavakkol[4], Jeong-A Lee[3], Hamid Sarbazi-Azad [*,2]

[1]Sharif University of Technology, Tehran, Tehran, Iran.
[2]Institute for Research in Fundamental Sciences (IPM), Tehran, Tehran, Iran.
[3]Chosun University, Gwangju, Gwangju, South Korea.
[4]ApplyBoard Inc, Kitchener, Kitchener, Canada.

*Corresponding author(s). E-mail(s): atiye.gheibi@gmail.com;
Contributing authors: fatemeh.serajeh@ce.sharif.edu; masoudmohammadilak@gmail.com; amir.mirzaei79@sharif.edu; akbarzadeh@ce.sharif.edu; mahmoud.kheyrati222@sharif.edu; smh.hosseini992@sharif.edu; ahmadjavadi17@gmail.com; arash82ir@gmail.com ; jalee@chosun.ac.kr ; azad@sharif.edu;

**Abstract**

With the ever-increasing demand of supercomputing and hyperscale data centers for high-performance and reliable storage systems, solid-state drives (SSDs) have emerged as a dominant technology. Unlike traditional hard disk drives (HDDs), SSDs utilize flash memory to achieve significantly faster data access and higher bandwidth. These advantages translate to improved system responsiveness, faster boot times, and enhanced application performance across various compute and data-intensive workloads.
However, the intricate architecture and dynamic nature of modern SSDs pose significant challenges for designers. Hardware advancements, such as the adoption of 3D NAND flash memory and complex controller algorithms, necessitate rigorous testing and optimization before real-world deployment. This is where SSD simulators and emulators become indispensable tools in the development cycle.

[*]He is now a Visiting Research Professor at Chosun University under the Brain Pool (BP) Program of the National Research Foundation (NRF) of Korea.

This survey delves into the critical role of SSD simulators and emulators, highlighting their contributions to accelerated innovation, performance optimization, predictive analysis, and future-proofing capabilities. Through a comprehensive evaluation of these tools, this work aims to provide valuable insights for researchers and developers, ultimately leading to the continued advancement of efficient, reliable, and future-proof SSD solutions.
**Keywords:** SSD, Simulator, Emulator, Hardware Platform

## 1 Introduction

The rapid adoption of solid-state drives (SSDs) in modern computing has transformed the storage landscape, offering significant advantages over traditional hard disk drives (HDDs) [1]. With their superior speed, energy efficiency, and durability, SSDs have become essential in a wide range of applications, from personal devices to large-scale data centers and high-performance computing systems [2]. As the demand for faster, more reliable storage solutions continues to grow [3], so too does the complexity of SSD technology, with advancements in flash memory, storage interfaces, and firmware designs. This growing complexity underscores the need for robust evaluation tools that can help researchers and developers design, test, and optimize SSDs under various conditions. Evaluation methods include measurement using real hardware, simulation through software models, and analytical modeling for performance predictions. Among these, simulation has emerged as the most popular approach due to its flexibility, cost-effectiveness, and ability to model a wide range of scenarios without the need for physical hardware.

Simulation has gained prominence because it enables the exploration of different SSD architectures and algorithms without the significant financial and time investments required for hardware-based measurement. Unlike analytical models, which can sometimes oversimplify complex systems, simulators allow for detailed and customizable experimentation with various workload patterns, flash memory technologies, and firmware algorithms. Furthermore, simulation offers the ability to assess long-term performance and reliability metrics, such as wear leveling and garbage collection efficiency, which are difficult to measure in real time on actual devices. This makes simulation an indispensable tool in SSD research, allowing developers to make informed decisions about design optimizations and performance improvements.

Given the intricate nature of SSD architectures—comprising flash memory cells, flash translation layers (FTLs), and complex algorithms for garbage collection, wear leveling, and error correction—it is essential to have a clear understanding of their internal workings. In the second section of this paper, we provide a comprehensive overview of these basic SSD architecture concepts, establishing a foundation for the discussion of the tools and methods used to evaluate SSD performance and reliability. Understanding these core components and how they interact within an SSD is crucial for evaluating the effectiveness of the various simulation, emulation, and hardware platforms explored later in the paper.

In the third section, we shift our focus to SSD simulators, which play a key role in the early stages of SSD development. Simulators allow researchers to model SSD architectures and algorithms, enabling them to assess performance, energy consumption, and reliability without the need for physical hardware. We discuss several well-known SSD simulators in detail, comparing their capabilities and use cases, and highlighting how they help in understanding and improving SSD design.

The fourth section covers SSD emulators, which bridge the gap between theoretical simulation and real-world testing. Emulators provide a hands-on environment where researchers can test firmware and algorithms in a virtualized SSD setting. Unlike simulators, emulators are designed to mimic the behavior of actual SSD hardware, making them invaluable for testing the performance and compatibility of different SSD configurations under real operating conditions. We present a detailed analysis of popular SSD emulators and their impact on SSD research and development.

Finally, in the fifth section, we explore hardware platforms that allow for direct testing of SSDs in real-world environments. These platforms provide crucial insights into how SSDs perform under actual workloads, offering a more holistic view of their behavior compared to simulation and emulation tools. By analyzing prominent hardware platforms, we highlight their role in pushing the boundaries of SSD performance and reliability. The paper concludes with a summary and recommendations for future work.

## 2 Basic Concepts

Solid State Drives (SSDs) are specifically engineered to harness the advantages of flash devices while mitigating their limitations. Flash memory is a non-volatile computer memory storage medium with the capability of electrically erasing and reprogramming data. This technology is constructed with a metal oxide semiconductor field-effect transistor incorporating a floating gate [4–6]. The floating gate is interposed between the metal control gate and the substrate layer. This floating gate serves to allow the flash cell to sustain one of two potential states, which are predicated on the presence or absence of trapped electrons in the floating gate [4, 5, 7, 8]. As can be shown in Figure 1, in the absence of electrons in the floating gate, the cell is in a ”1” state. Conversely, with the presence of electrons in the floating gate, the cell is deemed to be in a ”0” state.

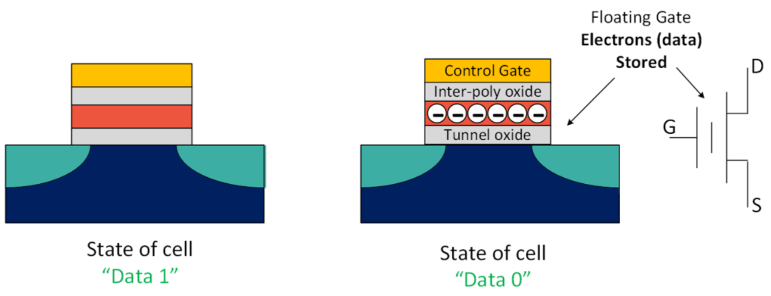

**Fig. 1** Flash Memory Cell.

The two primary forms of flash memory, namely NOR flash and NAND flash, are denoted after the NOR and NAND logic gates. Both variants incorporate the same cell design, featuring floating gate MOSFETs. Their differentiation occurs at the circuit level implementation on whether the state of the bit line or word line is pulled high or low: in NAND flash, the correlation between the bit line and the word line resembles a NAND gate, whereas, in NOR flash, it emulates a NOR gate [9]. NAND Flash SSDs have gained prominence in contemporary applications due to their enhanced density, which affords greater capacity compared to NOR SSDs [4, 5, 10].

Leveraging non-volatile flash memory, solid-state drives (SSDs) represent a significant leap forward in storage technology compared to traditional hard disk drives (HDDs) [1]. SSDs offer substantially lower latency, higher bandwidth, and greater energy efficiency. By enabling concurrent access to multiple flash memory chips, SSDs harness internal parallelism to surpass the performance limits of conventional storage solutions. This makes them an attractive option for a wide range of applications, from portable devices to high-capacity servers. Despite these advantages, the adoption of SSDs comes with certain challenges. While SSDs provide superior performance, they are often limited by factors such as shorter lifespan, higher cost per unit of storage, and lower overall storage capacity compared to HDDs [11]. Nevertheless, even with their higher upfront cost, SSDs have gained widespread acceptance, particularly in data centers, due to their lower energy consumption and enhanced performance [12–14].

The limited operational lifespan of flash memory cells stems from the frequent erase-before-write operations brought about by program and erase commands [9, 15, 16]. This challenge is chiefly addressed through the implementation of the Flash Translation Layer (FTL), which carries out address mapping, garbage collection, and wear leveling. The FTL layer fulfills a critical function in addressing the bottlenecks of flash memory, facilitating SSDs in sustaining high performance throughout their operational lifespan [9]. To better follow the functions and bottlenecks of SSDs, we give a brief explanation of the modern SSD architectures.

## 2.1 Modern SSD Architecture

As it is shown in Figure 2, SSDs are composed of an SSD controller ❸ and an array of flash memory chips ❶, each containing one or more dies ① that share a common multiplexed interface. Each die can operate independently, enhancing overall throughput. However, the shared interface for all dies may lead to potential contention among requests. Typically, a die consists of two or more planes ②, the smallest units for processing an I/O request. Each plane ③ consists of hundreds to thousands of flash blocks ④. A block is structured as a two-dimensional array made up of hundreds of rows (word line for accessing flash memory pages) of flash cells (usually 256 to 1024 rows), where each row stores consecutive data ⑤. Similar to multi-bank memory systems, the planes within a die can perform flash operations in parallel. However, these planes share a common set of data and control buses [17]. Consequently, operations can be initiated in different planes of the same die in a pipelined manner, with one operation starting each cycle. Data in a block is written at the page level, with each page typically ranging from 8 to 16 kB in NAND flash memory. Both read and write operations occur at the granularity of a page, and each block generally contains hundreds of pages [3] [8] [18] [19] [7] [20]. Blocks within each plane are assigned a unique ID within that plane, though this ID may be shared across multiple planes. Pages within a block are numbered sequentially [3].

To optimize multi-plane commands, such as read/program and erase operations, it is essential that the involved pages ⑤ or blocks ④ share identical die, block, and page addresses. Additionally, advanced commands in NAND flash memory chips, including interleaved and copy-back commands, further enhance functionality by enabling parallelism within a chip and facilitating page copying within a plane without utilizing communication channels.

To gain a deeper understanding of how modern SSDs function, we will now explore the SSD Controller and Flash Memory components in detail, beginning with the architecture of the SSD controller, followed by an examination of the flash memory architecture.

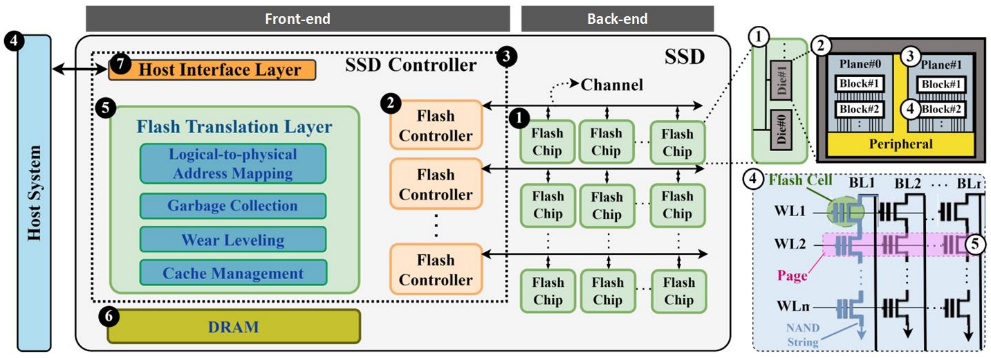

**Fig. 2** High-level overview of a solid-state drive [11].

### 2.1.1 SSD Controller

An SSD controller comprises multiple key components, including a host interface layer ❼, a processor (typically a multi-core processor in contemporary SSDs) responsible for executing the Flash Translation Layer (FTL) ❺, Dynamic Random-Access Memory (DRAM) ❻, a system bus (e.g., AXI), and flash controllers ❷ linked to numerous flash memory chips through flash channels/buses [11]. Also, some controllers include both compression and decompression features, enabling the reconstruction of the original data from the compressed version stored in flash memory [21][22].

Each of these key components plays a crucial role in ensuring the efficient operation of the SSD, from managing data flow between the host and the flash memory to handling error correction and data integrity. In the following, we will explore these components in greater detail, highlighting their specific functions and how they contribute to the overall performance and reliability of modern SSDs.

*Host Interface Layer (HIL)*: The Host Interface Layer (HIL) serves as the interface between the host system and the SSD controller. It communicates with the host system via a communication protocol over the system I/O bus. SSDs and their associated protocols continually advance to meet evolving system requirements. A key advancement has been the introduction of novel host interfaces for SSDs. Previously, many SSDs relied on the Serial Advanced Technology Attachment (SATA) protocol, originally intended for hard disk drives (HDDs). Over time, SATA has demonstrated inefficiency for SSDs due to its inability to support rapid I/O accesses and millions of I/O operations per second (IOPS) achievable with contemporary SSDs. Emerging protocols like NVMe mitigate these constraints by being specifically tailored to harness the high throughput capabilities inherent in SSDs [11].

*Flash Translation Layer (FTL)*: The primary function of the Flash Translation Layer (FTL) is to manage the mapping between logical addresses (used by the host) and physical addresses in the underlying flash memory (where data is actually stored and only accessible to the SSD controller) for each page of data [23][24]. By maintaining this indirection between the two address spaces, the FTL can remap a logical address to a different physical location (i.e., move data to a new physical address) without informing the host. When the host writes to a page or when data is moved for SSD maintenance tasks (such as garbage collection [25][26]), the old data (i.e., the previous physical location) is marked as invalid in the metadata of the physical block, and the new data is written to a page in the currently open block [3].

Over time, these page invalidations lead to block fragmentation, where most pages in a block become invalid. The FTL periodically conducts garbage collection to address this. It identifies highly fragmented blocks, migrates any remaining valid pages to a new block (aiming to fully populate it with valid pages), and then erases the entire fragmented block [25][26]. Garbage collection typically prioritizes selecting blocks with the lowest utilization (i.e., those containing the fewest valid pages) for erasure. Once garbage collection is completed and a block is erased, the block is added to a free list managed by the FTL. When the current block being written to becomes full, the SSD controller selects a new block from the free list to continue writing [3].

The FTL is also tasked with wear leveling, which ensures that all blocks within the SSD experience uniform wear out [25][26]. By evenly distributing the number of program/erase (P/E) cycles across the blocks, the SSD controller minimizes differences in wear levels, thereby prolonging the device's lifespan. Wear-leveling algorithms are activated when the current block being written to becomes full (i.e., when no more pages in the block are available), prompting the controller to select a new block from the free list. The wear-leveling algorithm determines which block is chosen from this list. A basic method is to select the block with the fewest P/E cycles to reduce wear variability across blocks, though various algorithms have been developed to optimize this process [3][27][28].

Moreover, FTL minimizes frequent accesses to flash memory by caching commonly accessed data, such as the logical-to-physical page mapping table [24], or frequently requested pages from the host, in the DRAM embedded within the SSD [11].

*Flash Controller*: The flash controller (FC) is an embedded processor within an SSD that interfaces with multiple flash chips via a shared channel. The FTL interacts with the FC to execute NAND flash operations. The flash controller encompasses an ECC unit, internal memory page buffers, command control logic for overseeing flash operations, and timing sequence generators for each control/data pin. Modern SSDs adopt a multi-channel (or multi-bus) architecture, where a flash memory channel/bus is interfaced with multiple flash chips. The flash controllers establish communication with the flash memory chips utilizing control/data and arbitration pins. During a write operation, the controller obfuscates data to forestall high bit error rates stemming from extreme data patterns. Furthermore, it employs ECC encoding to heighten reliability and performance. Subsequently, the controller transmits a write command accompanied by the physical page address to the target flash chip and then transfers the obfuscated ECC-encoded write data to the chip. For read operations, the flash controller conducts several sequential steps: issuance of a read command to the target flash chip, reception of data from the flash chip, execution of ECC decoding to rectify potential data errors, and final derandomization of the read data to restore its original form. [29] [3] [19] [7][30][31][32]. Two main tasks for the flash controller are i) error-correcting codes (ECC) , and ii) data randomization [32][33][34][3].

ECC is employed to detect and correct raw bit errors in flash memory. When the host writes a page of data, the SSD controller divides it into one or more chunks. For each chunk, the controller generates a codeword, which includes both the chunk and a correction code. The level of protection provided by ECC is determined by the coding rate, defined as the ratio of chunk size to codeword size. A higher coding rate offers weaker protection but requires less storage, representing a critical tradeoff between reliability and storage efficiency in SSDs [3]. Along with ECC information, a codeword also includes cyclic redundancy checksum (CRC) parity data [35][36]. When reading data from NAND flash memory, the ECC algorithm may occasionally indicate that it has corrected all errors, even though some errors remain. To prevent returning incorrect data to the user, the controller performs a hardware-based CRC check to verify that the data is free from errors [3][35].

Errors in flash memory are heavily influenced by the specific data values stored in memory cells [20][37][38]. Data Randomization before writing it to the flash chips is utilized by SSD controller to minimize the impact of data-dependent errors[34][39]. The primary goal of scrambling (data randomization) is to ensure the data written to the SSD contains a balanced distribution of zeroes and ones, reducing any dependence on the data values and preventing value-related error patterns. Scrambling is performed through a reversible process, allowing the controller to descramble the data during read operations. A linear feedback shift register (LFSR) is used by the controller to handle both scrambling and descrambling. An n bit LFSR generates $2^{(n-1)}$ pseudo-random bits without repetition. To ensure data can be correctly descrambled even if it is moved during maintenance tasks (such as garbage collection), the LFSR is seeded with the logical address of the page being written. This approach ensures that the logical address remains unchanged, eliminating the need for descrambling and rescrambling during data migration, which also reduces maintenance latency. The LFSR produces a pseudo-random number based on the seed, which is then XORed with the data to produce the scrambled version. Since the XOR operation is reversible, the same process is used to descramble the data when it is read back [3].

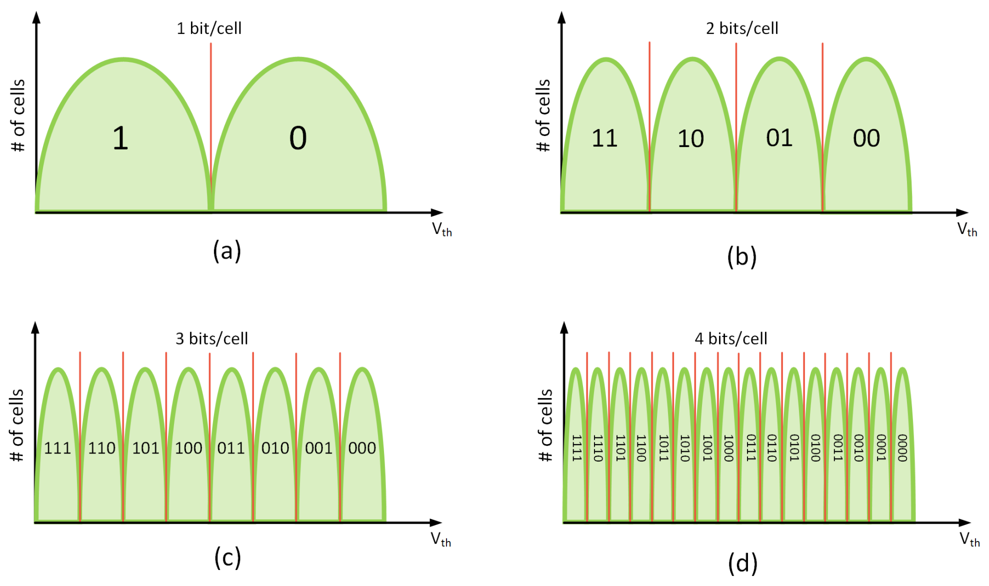

**Fig. 3** Flash cell storage capacity. (a) SLC, (b) MLC, (c) TLC, (d) QLC.

### 2.1.2 Flash Memory

As mentioned before, one or more flash chips are connected to a shared channel, each flash chip contains one or more dies and each die contains one or more planes. Each plane consists of many blocks and each block consists of many pages. Each page contains many flash cells. Flash cells are categorized by their storage capacity, denoted as single-level cell (SLC), multi-level cell (MLC), triple-level cell (TLC), or quad-level cell (QLC). In SLC NAND flash, each cell stores a 1-bit value and can be programmed to one of two distinct threshold voltage states (Figure 3.a). In contrast, MLC NAND flash stores 2 bits per cell, corresponding to four possible states (Figure 3.b), while TLC NAND flash stores 3 bits per cell, allowing for eight possible states (Figure 3.c), and QLC NAND flash stores 4 bits per cell, allowing for sixteen possible states (Figure 3.d). Each state signifies a specific value and is assigned a voltage window within the overall range of threshold voltages. Variations in programming operations result in the threshold voltage of cells programmed to the same state being initially distributed across this voltage window [3]. As the number of bits in the flash cell increases, the SSD's storage capacity also increases. However, this higher density of flash cells leads to increased latency and decreased endurance [40] [29] [3] [7] [20] [41] [42] [43] [44][45] [46] [11].

The programming of a "0" is achieved by introducing electrons into the floating gate during the programming operation. This is carried out using the process of Fowler-Nordheim tunneling to inject electrons into the floating gate. Fowler-Nordheim (FN) tunneling occurs when a significant voltage, for instance, 18 V, is applied between the control gate and the substrate. The process of reading data from a flash memory cell is contingent upon a threshold voltage. In a Single-Level Cell (SLC) flash memory configuration, the threshold voltage for interpreting a "1" should be lower than the read voltage, while the threshold voltage for a "0" should be higher than the read voltage. During a page read operation, the read voltage is utilized to access the control gate of the flash cell in question. Subsequently, the presence or absence of electric current flow through the transistor is used to discern the stored information. If an electric current is detected, the cell contains the data "1"; conversely, if no current is present, it holds the data "0". During the erase operation, electrons are removed from the floating gate to alter the cell's state to "1". This is achieved by applying a substantial negative voltage to the substrate while connecting the control gate to the ground. Consequently, electrons move away from the floating gate through the FN tunneling. The erase process is carried out in a block-by-block manner to prevent unintended changes from "0" to "1" in individual flash cells. As a result, before writing new data, empty blocks must be prepared (erase-before-write). It is important to note that the erase process takes longer compared to read and write operations, leading to a significant performance bottleneck for NAND flash memories. To mitigate this extended erase latency, efforts are made to conceal its impact on overall performance [3] [7] [20] [41] [42].

Numerous emerging types of devices are frequently optimized to efficiently handle their primary workloads, necessitating a customized communication model between the host and the device. Therefore, the research community started to introduce simulation tools that reliably model these new features and protocols. There is a variety of simulators, emulators, and also hardware platforms that model different generations of SSDs and their protocols . In this paper, we provide an overview of the cutting-edge infrastructures that simulate, emulate, or implement SSDs at both the software and hardware levels. The main purpose of this paper is to familiarize researchers with these infrastructures and to aid the advancement of storage devices by exploring their limitations, as well as the potential of innovative solutions in shaping the future of SSDs.

## 3 Simulators

The role of simulators in the study and development of SSD technology is crucial for understanding and predicting the behavior of these storage systems under a wide range of conditions. Simulators provide researchers and engineers with the ability to model SSD architectures and simulate their performance, reliability, and energy consumption without the need for physical prototypes. These tools are instrumental in experimenting with various algorithms, such as garbage collection, wear leveling, and error correction techniques, and in assessing their effectiveness in improving SSD performance and longevity. By simulating different workloads, hardware configurations, and firmware strategies, simulators offer invaluable insights into the optimization of SSDs, enabling the development of more efficient, reliable, and cost-effective storage solutions. In this section, we will explore some of the most well-known simulators used in SSD research, providing a detailed comparison of their features and capabilities.

### 3.1 MQSim

MQSim [2], introduced in 2018 at the 16th USENIX Conference on File and Storage Technologies, is a simulation framework designed for studying multi-queue SSD devices. Developed by ETH Zürich and Carnegie Mellon University, MQSim improves upon earlier simulators by accurately modeling both traditional SATA SSDs and modern types like NVMe. It supports diverse workloads and detailed modeling of SSD protocols, ensuring precise performance simulations across various interfaces and architectures. With comprehensive models of conventional and modern host interfaces, MQSim accurately simulates SSD behavior and request latency. Its modular design allows users to modify individual components without altering the overall framework, ensuring adaptability to future SSD developments. By simulating features like multi-queue request handling and advanced maintenance algorithms, MQSim provides performance results that closely match those of real SSDs, making it an invaluable tool for researchers. Key features of MQSim include support for standalone programming mode, offering a dedicated environment for simulating SSD behaviors in isolation. The simulator supports integration with full-system simulators, enabling detailed simulation of memory-related behaviors. It also incorporates advanced storage features like transaction scheduling and incremental step pulse programming for comprehensive flash memory simulations. MQSim offers configurable cache settings, including fully associative cache configurations, allowing for the testing of various caching strategies. Also, it can simulate both hybrid and page-level flash translation layers for in-depth analysis of flash memory behavior and management [47].

MQSim excels in steady-state simulation, accurately and quickly modeling steady-state SSD behavior, which is crucial for realistic performance evaluation. The simulator captures the full end-to-end latency of I/O requests, offering a comprehensive understanding of SSD performance. Additionally, MQSim simulates various workloads, uncovering issues like inter-flow interference and performance impacts not captured by existing simulators. As it has mentioned above, the simulator reports performance results that closely match those of real state-of-the-art SSDs, with an average error margin of only 11%. However, some weaknesses exist. The detailed and comprehensive modeling in MQSim can make it complex to set up and use. The accuracy and depth of the simulations can also be computationally intensive, demanding significant processing power and memory. Additionally, MQSim's focus on SSDs limits its suitability for simulating other types of storage devices or broader storage systems without modification.

MQSim-E [48] is an extension of MQSim that enhances the capabilities of MQSim by incorporating essential features needed in enterprise environments. Among the key functionalities of MQSim-E are the accurate modeling of the NVMe interface, which is vital for modern enterprise SSDs, and the inclusion of advanced SSD maintenance algorithms, such as dynamic address allocation, fine-grained address mapping, and token-based garbage collection. Additionally, MQSim-E offers a more realistic perspective of performance over time.

### 3.2 SimpleSSD version 1.x

SimpleSSD [49], introduced in 2017 by Jung et al., is a sophisticated SSD simulator designed to provide high-fidelity modeling of both hardware and software characteristics of SSDs. This simulator supports comprehensive system-level performance evaluations and a variety of workloads. SimpleSSD employs a layered approach to simulate SSD internal processes, featuring a host interface layer (HIL) for managing I/O requests, a flash translation layer (FTL) for address translation, and a parallelism abstraction layer (PAL) for request parallelism. This architecture delivers detailed performance metrics and integrates with full-system simulators like gem5, facilitating CPU performance evaluations and exploration of different SSD architectures.

The framework supports both standalone and full system modes, enhancing SSD design flexibility. It is compatible with Atomic ARM CPU models and aligns with modern storage standards through NVMe support. SimpleSSD includes DRAM support within its computation complex and offers advanced features like transaction scheduling, super page/block management, and incremental step pulse programming for detailed flash memory simulations. Configurable cache settings optimize data retrieval and storage efficiency, while power models for NAND enable accurate power consumption simulations. The framework also supports dynamic firmware execution and comprehensive queue management for realistic data transfer emulation [2][47]. The SimpleSSD's layered architecture allows for detailed simulation of SSD internals, offering researchers opportunities to explore new strategies and algorithms. However, the detailed modeling might be overwhelming for new users or those with limited technical knowledge, and full-system simulations at the cycle level can require significant computational resources and long runtimes.

## 3.3 SimpleSSD version 2.x (Amber)

Amber [47] is an advanced SSD simulation framework developed by the Computer Architecture and Memory Systems Lab at Yonsei University, in collaboration with Pennsylvania State University and the University of Illinois Urbana-Champaign. Released in 2018, it integrates both computation and storage components within an SSD, supporting a wide array of storage interface protocols and data transfer emulation. Amber is capable of evaluating diverse workloads with support for multiple SSD interface protocols, including SATA, NVMe, UFS, and OCSSD. Amber models embedded CPU cores, DRAMs, and various flash technologies, enabling full-system simulation and data transfer emulation. The framework includes a comprehensive firmware stack, covering DRAM cache logic and flash firmware such as FTL and HIL, and supports standard protocols. By modifying the host system's DMA engines and system buses in a popular simulator like gem5, Amber can capture dynamic performance and power details of embedded cores, DRAMs, firmware, and flash during various OS system and hardware platform executions. It supports standalone programming and full system modes, enhancing flexibility in SSD design and operation, and is compatible with various ARM CPU models, including Atomic, Timing, Minor, High-performance In-Order, DerivO3, and O3-v7a. Amber's computation complex includes support for both CPU and DRAM, ensuring comprehensive simulation capabilities for embedded systems. Its advanced storage complex features, including transaction scheduling, super page/block management, and incremental step pulse programming, enable detailed flash memory simulations. Amber supports configurable cache settings, including readahead and fully associative cache configurations, optimizing data retrieval and storage efficiency. Additionally, it can simulate both hybrid and page-level flash translation layers, allowing for detailed analysis of flash memory behavior. Amber includes power models for CPU, DRAM, and NAND, facilitating accurate energy consumption and efficiency analysis. The simulation framework also supports energy and dynamic firmware execution dynamics, essential for assessing performance under real-world conditions. Finally, Amber incorporates comprehensive queue management, crucial for emulating realistic data transfer and I/O operations.

Amber boasts several strengths and weaknesses. Its comprehensive modeling capabilities encompass detailed simulations of SSD resources such as embedded CPU cores, DRAMs, and various flash technologies, resulting in precise performance, power consumption, and firmware execution assessments. Amber's full-system simulation integrates both computation and storage complexes, enabling an evaluation of system component interactions and their performance impact. Amber is versatile for simulating a range of storage applications and hardware configurations. The simulator provides realistic data transfer emulation, closely mimicking real SSD behavior under various I/O operations. Additionally, Amber dynamically measures power consumption during firmware executions, accounting for power usage by embedded cores, internal DRAMs, and flash devices. Its tight integration with the gem5 simulator allows for detailed CPU timing and full-system environment simulations, facilitating the evaluation of system-level challenges across different operating systems and hardware platforms. However, Amber's detailed modeling approach introduces complexity and a steeper learning curve for users unfamiliar with SSD architecture and simulation tools. The simulator's detailed modeling and full-system simulation capabilities can lead to slower performance compared to more simplified simulators. Moreover, the resource-intensive nature of its simulations requires significant computational power and memory, which may not be feasible for all users. Finally, Amber's focus on detailed SSD modeling and integration with gem5 may limit its applicability to scenarios where less detailed but faster simulations are sufficient.

## 3.4 VSSIM

VSSIM [50], a virtual machine-based SSD simulator developed by researchers at Hanyang University in 2013, provides a robust platform for analyzing SSD performance under various workloads. The simulator offers a comprehensive tool for evaluating different storage interfaces and their operational characteristics. VSSIM's architecture includes an SSD model built on QEMU/KVM, featuring components like an FTL module, I/O emulator, SSD monitor, and Latency Manager. These components facilitate accurate emulation of SSD operations. The FTL module manages block status and allocation, while the I/O emulator handles NAND operations, including data transfers and flash memory tasks. The SSD monitors processes and visualizes real-time SSD operation data, and the Latency Manager introduces precise delays to emulate NAND operations accurately, making VSSIM an essential tool for both academic research and practical SSD development.

VSSIM is a robust SSD simulator with several notable strengths. It offers a highly detailed and modular model, allowing for the simulation of various hardware and software components, and making it adaptable for diverse research and development needs. Its real-time execution feature enables users to observe the performance of the host system with a given SSD design dynamically. VSSIM accurately models the behavior of physical SSDs, with validation against commercial SSDs like the Intel X25M, showing a performance offset of just 3% for sequential I/O. However, VSSIM has some weaknesses. It is less accurate in simulating random I/O performance compared to sequential I/O, which can affect the precision of performance studies involving random data access patterns. Additionally, as a virtual machine-based simulator, it relies on the QEMU/KVM environment, which might limit its applicability in certain contexts or require additional setup and configuration efforts. Furthermore, running VSSIM in real-time with detailed simulations can be resource-intensive, potentially requiring significant computational power and memory.

## 3.5 SSDSim

SSDSim [51] is a sophisticated SSD simulator introduced in 2011 by Yang Hu and his research team. This simulator is crafted to examine and analyze the internal behaviors of SSDs, focusing on performance and endurance impacts under diverse conditions. It utilizes real-world workloads typical of high-performance computing environments, such as financial data processing and web search to ensure its accuracy against hardware prototypes. The simulator models the internal architecture of SSDs in detail and parallelism at the channel, chip, die, and plane levels, as well as the FTL. It accurately simulates advanced commands and allocation strategies, providing metrics like waiting time, processing time, response time, and buffer hit counts. Operating in a standalone programming mode, SSDSim offers a focused environment for isolated simulation of SSD behaviors. The platform supports fully associative cache configurations. SSDSim can also simulate hybrid and page-level flash translation layers, facilitating a comprehensive analysis of flash memory behavior and management [2].

SSDSim boasts several strengths, including its high-fidelity simulation capabilities that accurately model SSD behavior by considering software processing costs, a significant component of SSD response time. This ensures precise performance metrics. The simulator's results have been validated against real SSD hardware prototypes, showing minimal deviation (2-2.9%) in average response times, which attests to its reliability. However, SSDSim's detailed accounting for software processing costs, while increasing accuracy, also requires more computational resources and time compared to simpler simulators.

## 3.6 FlashSim

FlashSim [52] is a comprehensive simulator for NAND Flash-based SSDs, developed to address the need for performance evaluation in this emerging storage technology. Released in 2009 by the Department of Computer Science and Engineering at Pennsylvania State University, FlashSim is integrated with the DiskSim simulator, enabling hybrid simulations of SSDs and HDDs. It supports various workload types and evaluates different FTL schemes. The simulator operates using an event-driven, object-oriented approach in C++, allowing for modularity and easy extensibility. It simulates SSD operations by creating event objects that represent different hardware components, including packages, dies, planes, blocks, and pages, as well as software components like the FTL and RAM buffers. FlashSim supports standalone programming mode, providing a specialized environment for simulating isolated SSD behaviors. It includes sophisticated storage features such as super page/block management, enhancing its capability to simulate complex SSD operations. Additionally, FlashSim can simulate both hybrid and page-level flash translation layers, offering a detailed analysis of flash memory management and behavior [2][47].

FlashSim boasts several strengths that make it a robust SSD simulator. FlashSim's accuracy is ensured through validation against real SSD devices for behavioral similarity. Additionally, FlashSim includes energy consumption analysis using different FTL schemes with real traces. However, there are some weaknesses. The initial versions were constrained by a simplified hardware model, though this has been improved in later iterations. As a single-threaded program, FlashSim's performance may be limited in simulating complex, multi-threaded SSD operations. Lastly, despite its design for extensibility, adding new features still demands significant effort due to the complexity of accurately simulating SSD behavior.

## 3.7 WiscSim

WiscSim [53] is a comprehensive SSD simulator developed by a research group at the University of Wisconsin–Madison to facilitate detailed analysis of SSD behavior under various workloads. Published in 2017, WiscSim supports advanced functionalities like Native Command Queuing (NCQ), multiple mapping and page allocation schemes, garbage collection, and wear-leveling. This simulator was designed to provide insights into the internal metrics of SSDs rather than just end-to-end performance, making it a powerful tool for understanding SSD operations. The workloads tested on WiscSim include popular applications like LevelDB, RocksDB, SQLite, and Varmail, which simulate diverse usage patterns and access behaviors essential for comprehensive SSD performance evaluation. WiscSim operates as a discrete-event simulator, modeling the behavior of SSDs at a granular level by processing events like read, write, and erase operations, allowing researchers to observe the impact of different workloads and configurations on SSD performance over time.

WiscSim offers several strengths, such as providing detailed simulation capabilities that offer granular insights into internal SSD metrics, including cache miss ratios and garbage collection efficiency, beyond just end-to-end performance. Additionally, WiscSim is well-tested with over 350 unit tests, including end-to-end data integrity checks, ensuring reliability and accuracy in simulation results. It is also capable of testing various workloads covering a wide range of real-world applications and access patterns. However, WiscSim's detailed internal metrics and advanced features can introduce performance overhead, resulting in slower simulations compared to more simplified models. Moreover, while it is highly detailed for SSD analysis, its specialization limits its usefulness for simulating other types of storage devices without significant modifications.

## 3.8 NANDFlashSim

NANDFlashSim [54] is a high-fidelity, microarchitecture-aware NAND flash memory simulator introduced by a group of researchers to provide an accurate and detailed simulation of NAND flash memory behavior in 2016. This simulator was designed to evaluate the performance of various workloads, including write-intensive, read-intensive, and real-world applications such as online transactions and search engines. NANDFlashSim supports different types of SSDs. The simulator employs a highly reconfigurable and detailed timing model for various NAND flash memory systems, using multistage operations and command chains to handle fine-grained NAND transactions. This enables memory system designers to closely study NAND flash performance and optimization points at a cycle level, providing valuable insights into performance enhancement and system design.

Its strengths include high configurability, allowing for the simulation of various NAND operations and system configurations, and awareness of intrinsic latency variations in NAND flash operations. It offers cycle-level simulation with detailed timing models for precise performance evaluations and optimizations. Additionally, NANDFlashSim handles parallelism by simulating complex operations like multistage operations and command chains to manage fine-grained NAND transactions. However, the simulator's complexity can pose challenges for users without a thorough understanding, and simulations indicate increased resource contention as the number of dies grows, which limits performance improvements.

## 3.9 FlashStorageSim

FlashStorageSim [55], introduced in 2017 by a team from Samsung Semiconductor, Inc., is a sophisticated performance modeling simulator designed for SSD architectures used in data center servers. It extends the capabilities of NVDIMMSim by adding detailed models for SSD controllers, flash devices, and host interfaces like SATA, PCIe, and DDR. The simulator focuses on evaluating SSD performance under different workloads such as Fio, LinkBench, and YCSB. FlashStorageSim operates by accurately modeling the SSD's internal architecture, including channel interleaving and way pipelining, and it implements host logic and FTL functions such as Garbage Collection and Wear Leveling. Additionally, it features advanced mechanisms like fast-forwarding and cycle granularity adjustment to significantly enhance simulation speed, achieving up to 7X faster performance compared to traditional simulators.

FlashStorageSim includes detailed flash management features like garbage collection and wear leveling, which are crucial for realistic SSD performance evaluation. Additionally, its accuracy has been validated against real SSD performance data, ensuring reliable simulation of real-world conditions. However, the simulator has some weaknesses. The trade-offs in accuracy for speed, while beneficial for faster simulations, might not be suitable for all types of analysis. Furthermore, its detailed and comprehensive nature can make setup and configuration complex and time-consuming.

## 3.10 EagleTree

EagleTree [56] is an open-source simulation framework for SSD-based applications, developed to explore the complex design space of the complete I/O stack, including applications, operating systems, SSD controllers, and flash chip arrays. Introduced in 2013 by the EagleTree team, this simulator allows for extensive experimentation with cross-layer designs in a controlled virtual environment. The workloads supported by EagleTree range across various SSD operations, providing insights into hardware bottlenecks, OS scheduling strategies, and the impact of different mapping and garbage collection strategies. EagleTree operates by simulating the entire system in virtual time, enabling hundreds of experiments to be conducted efficiently. It allows users to manipulate SSD internals, configure hardware parameters, and explore various scheduling and mapping strategies to evaluate their impact on performance.

EagleTree, an advanced SSD simulator, boasts several key strengths. Its cross-layer design exploration feature enables experimentation with integrated designs, promoting optimized and efficient SSD-based applications. As an open-source tool, it is freely available, encouraging community contributions and broad accessibility for researchers and practitioners. Users can easily configure parameters such as flash chip types, SSD geometry, and advanced command support, enhancing its versatility. However, some weaknesses include the need for validation against real system components, a potentially complex setup for those lacking technical expertise, and performance discrepancies between simulated and real-world SSDs.

## 3.11 SRsim

SRsim [57], published in 2015 by a team from the Korea Advanced Institute of Science and Technology (KAIST) and the Scientific Data Research Center KISTI, is an open-source simulator designed to model the behavior of SSD-based RAID systems. It aims to assist researchers in exploring and experimenting with various SSD arrays and RAID configurations. SRsim supports different RAID levels, including RAID 0, RAID 5, and RAID 6, and can process a range of general storage traces. The operational process of SRsim involves several key steps. Initially, it scans the input trace files and converts logical addresses to distribute data blocks across the SSD array. It then generates stripes with data blocks stored in different SSDs and assigns unique stripe IDs to each stripe based on the RAID configuration. The striping manager handles read, write, and update operations on data blocks and parities in the SSD array. Finally, the simulator processes I/O requests in parallel, ensuring efficient and accurate simulation of SSD-based RAID environments.

SRsim offers several strengths and a few weaknesses. SRsim provides accurate performance simulations of SSD arrays and RAID environments, which aids in evaluating performance and reliability. It efficiently handles parallel I/O processing, mimicking real-world SSD behavior, and is compatible with general storage traces, making it interoperable with other simulators like FlashSim. Additionally, it includes a comprehensive RAID controller with advanced features like address translation, striping management, and parity generation. However, the setup can be complex, requiring a detailed understanding of SSD and RAID principles for accurate simulation. SRsim focuses exclusively on SSD-based RAID, limiting its applicability to other types of storage systems. The detailed modeling of SSD internals and RAID operations may introduce performance overhead, and the accuracy of simulations heavily relies on the quality and characteristics of the input trace files.

## 3.12 SSDModel

This simulation environment [17] is a trace-driven system, adapted from the DiskSim simulator developed by the CMU Parallel Data Lab, specifically designed for SSD behavior analysis. Developed in 2008 by a research group from University of Wisconsin-Madison and Microsoft Research presented at the USENIX Annual Technical Conference, this simulator processes various workloads, including TPC-C, Exchange, IOzone, and Postmark, to evaluate SSD performance. The simulator operates by replaying traces of real system workloads to emulate SSD operations, providing detailed performance metrics and insights into how different SSD configurations handle various data management tasks. This allows for an in-depth analysis of design tradeoffs and their impacts on performance and longevity.

SSDModel offers a powerful platform for detailed analysis, allowing users to comprehensively evaluate various SSD configurations and their performance under different workloads. Additionally, SSDModel delivers detailed performance metrics, including data placement, parallelism, write ordering, and workload management, aiding in the understanding of SSD behavior and design tradeoffs. However, the simulator's complexity may pose challenges in setup and usage, requiring significant expertise to customize and accurately interpret results.

## 3.13 SSDExplorer

SSDExplorer [58] is a sophisticated virtual platform designed for fine-grained design space exploration of SSDs. Published in 2015 by a collaborative team from the University of Ferrara and Politecnico di Torino, the tool enables detailed simulation and analysis of various SSD architectures. It facilitates the evaluation of performance and reliability trade-offs under different workloads, offering valuable insights for optimizing SSD design while minimizing the risk of overdesign. The simulator operates by modeling the behavior of SSD components and their interactions with workloads, allowing users to modify parameters such as interface types, workload characteristics, and flash management algorithms to observe their impact on overall SSD performance and efficiency.

SSDExplorer offers several advantages, including detailed simulation capabilities that allow for fine-grained analysis of various SSD architectures and versatile support for multiple SSD types. It enables workload customization, allowing users to modify workload characteristics and SSD parameters to simulate different scenarios, which is invaluable for optimizing SSD design and minimizing overdesign. However, the tool's complexity can be a drawback, as its detailed simulations require significant computational resources and time.

## 3.14 FTLSim

FTLSim [59] is an SSD simulator developed in 2012 by Peter Desnoyers at Northeastern University, designed to model SSD performance, especially in terms of write amplification and garbage collection efficiency. This simulator accommodates real-world workloads, providing both uniform and non-uniform traffic models to predict SSD behavior. FTLSim supports various SSD configurations, including fully page-mapped SSDs, and simulates cleaning mechanisms like LRU and greedy algorithms. It evaluates SSD operations, such as page writes, block erasures, and internal copying, to produce performance metrics like write amplification. Through its analytic models, FTLSim enables performance assessment under diverse conditions, facilitating optimization for interfaces and workloads, including random and sequential write patterns.

While FTLSim offers precise closed-form solutions for write amplification and supports diverse real-world workloads, it has limitations. It lacks wear-leveling support, a key aspect for SSD longevity, and simplifies garbage collection processes, which might overlook the full intricacies of SSD architectures. Additionally, although it models page-mapped FTLs well, it lacks depth in handling hybrid block/page-mapped systems and is best suited for static or controlled workloads, limiting its application for dynamic real-world scenarios.

To address some of these limitations, FTLSim-WL [60], an extension of FTLSim, was introduced in 2022 by researchers at Syracuse University and Florida International University. FTLSim-WL expands FTLSim's capabilities by adding wear-leveling evaluation, focusing on both static and dynamic wear-leveling algorithms. It models SSD performance for various workloads, from synthetic random writes to real-world I/O traces, offering a closer examination of SSD endurance and performance degradation over time. FTLSim-WL includes features like pre-conditioning the SSD to a steady state using sequential and random writes, enhancing simulation accuracy before wear and performance evaluations.

FTLSim-WL provides detailed insights into wear leveling, accurately tracking write amplification and the distribution of erase counts, and is highly customizable with adjustable workload parameters. However, it primarily emphasizes wear-leveling impacts, leaving aspects like latency and throughput less examined. FTLSim-WL can be time-intensive, especially for long-term lifespan analysis, and is designed specifically for fully page-mapped SSDs, limiting its applicability to other SSD architectures. Together, FTLSim and FTLSim-WL serve as robust tools for modeling SSD behavior, with FTLSim-WL offering additional focus on SSD endurance.

## 3.15 Other Simulators

Several other simulators, though less widely known, also contribute valuable insights into SSD behavior. SSD-Extension [61], developed by Microsoft Research in 2009, builds on the DiskSim tool [62] from CMU to enable SSD-specific simulations. While commonly used in academic and industrial settings, it focuses primarily on modeling the flash translation layer (FTL) without simulating the detailed hardware and software aspects of SSDs. As a standalone simulator, SSD-Extension is well-suited for targeted experiments in page mapping and storage management, though its limitations necessitate more advanced tools for comprehensive system-level simulation.

NVMain [63], introduced in 2012, takes a different approach by offering an architectural-level simulator for non-volatile memories. Designed for hybrid and emerging memory technologies, NVMain enables system-level simulations, considering various aspects like memory organization, energy efficiency, and performance across different workloads. Its ability to simulate different bus models and memory configurations makes it particularly useful for research into the interaction between DRAM and non-volatile memories, offering insights into how memory controllers and interconnects influence system performance.

NVMeSim [64], developed in 2016, focuses on simulating the Non-Volatile Memory Express (NVMe) protocol, particularly in systems using PCIe interfaces. It offers detailed modeling of host controllers, queue management, and data transfer mechanisms, making it a valuable tool for evaluating high-performance SSDs. NVMeSim's ability to handle various non-volatile memory technologies and storage stacks gives it a unique role in testing SSD performance under real-world conditions, particularly for high-speed storage solutions.

The table 1 presents a comprehensive feature comparison of well-known SSD simulators discussed in this section. This comparison highlights each simulator's capabilities in three primary areas: linear modeling, component modeling, and additional specialized abilities. Linear modeling capabilities, such as timing and request processing, are crucial for performance evaluations, while component modeling features focus on internal SSD operations like garbage collection and wear-leveling. The "Abilities" section outlines advanced functionalities, including support for real-world deployments and modern protocols like NVMe. Together, these dimensions provide a detailed view of each simulator's strengths and limitations, enabling a clearer understanding of which tools are most suitable for specific SSD research and development needs.

**Table 1** Feature Comparison of various Simulators.

| Feature | Linear Modeling | | | | | | | Component Modeling | | | | | Abilities | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Device | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| MQSim | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| SimpleSSD 1.x | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Amber | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| VSSIM | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| SSDSim | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| FlashSim | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| WiscSim | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| NANDFlashSim | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| FlashStorageSim | - | - | - | - | - | ✓ | ✗ | ✓ | - | - | ✓ | - | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| EagleTree | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗** | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| SRsim | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| SSDModel | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| SSDExplorer | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| FTLSim-WL | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |

1- NVM R/W Timing
2- Multi-Queue-Aware request processing in FTL
3- FTL Proc. (FTL request processing overhead)
4- Cache Access Latency
5- Host-to-Device & Device-to-Host Transfer Latency
6- Page-Level Address Mapping
7- Hybrid-Address Mapping
8- Garbage Collection
9- Write Cache
10- FTL Transaction Scheduling Unit (TSU)
11- FTL Wear-Leveling Unit
12- NVM Transfer
13- Deployable in Real Environments
14- NVMe Multi-Queue Support
15- NVMe Interface Modification
16- Low-Latency Device Support
17- Kernel Bypassing with SPDK
18- PCI peer-to-peer DMA Support
19- NVMe-oF Target Offloading
* Similar operation happens in FIL (Flash Interface Layer) based on paper.
** Can be added to the simulator. Doesn't exist by default.

## 4 Emulators

Emulators serve as a bridge between theoretical models and real-world SSD hardware by mimicking the behavior of SSDs at a high level of accuracy. Unlike simulators, which primarily focus on abstract modeling, emulators provide a more hands-on approach by allowing users to interact with virtualized SSD environments as if they were physical devices. This makes them particularly useful for testing firmware, validating algorithms, and conducting performance evaluations under realistic operating conditions. By reproducing the functional characteristics of SSDs, emulators enable researchers and developers to experiment with different configurations, troubleshoot issues, and optimize performance without the need for physical hardware. In this section, we will examine several prominent SSD emulators, mentioning their strengths and limitations in various research and development contexts.

### 4.1 NVMeVirt

NVMeVirt [65] is a software-defined virtual NVMe device simulator developed by a research team from Ajou University and Seoul National University, presented in 2023 at the 21st USENIX Conference on File and Storage Technologies. It supports various types of storage devices, including conventional SSDs, NVM SSDs, ZNS, and KVSSD. NVMeVirt bridges the gap between the host I/O stack and virtual NVMe devices, offering flexibility in defining NVMe device types with custom features. It emulates device performance for real workloads, such as database engines, using a detailed performance model that reflects real device characteristics. This allows users to predict application performance on future storage devices with different latency and bandwidth parameters. NVMeVirt provides a comprehensive platform for storage research and development by supporting advanced features like PCI peer-to-peer DMA and NVMe-oF target offloading. The simulator works by intercepting and processing NVMe commands from the host and then translating these commands to the virtual NVMe device's characteristics. It employs a kernel module that integrates with the Linux I/O stack to handle I/O requests efficiently. Performance models simulate the behavior of different SSD types under various workloads, based on real device measurements, ensuring accurate emulation of latency, throughput, and other performance metrics. NVMeVirt can replicate complex storage environments, allowing developers to test and optimize their software for different storage configurations without access to physical hardware.

NVMeVirt's strengths include its flexibility in allowing users to define any NVMe device type with custom features, making it adaptable to various storage needs. It aids in understanding the performance characteristics of applications and operating systems under real workloads and facilitates rapid prototyping and development of new NVMe devices. However, the complexity of implementing and using NVMeVirt requires a good understanding of NVMe protocols and storage device internals. While designed to minimize overhead, software-based emulation might still introduce performance overhead compared to real hardware. Additionally, as a Linux kernel module, NVMeVirt requires a Linux environment, potentially limiting its use in other operating systems.

## 4.2 Confzns

ConfZNS [66] is an advanced emulator designed to explore the design space of Zoned Namespace (ZNS) SSDs. Developed in 2023 by researchers from Dankook University and Syracuse University, ConfZNS offers a precise and configurable environment to emulate ZNS SSDs, providing insights into their performance and latency characteristics under various configurations. It allows users to experiment with different workloads and zone mappings, enhancing the understanding of how these devices handle parallelism and isolation. ConfZNS has been validated against real ZNS SSDs, demonstrating a high accuracy in performance emulation, making it a valuable tool for both hardware and software researchers. The emulator functions by simulating diverse internal architectures and resource allocations of ZNS SSDs, capturing the timing characteristics and resource contention effects through a detailed latency model, and providing full-stack exploration from the internal device structure to application software.

Users of ConfZNS can easily configure various parameters, such as zone size, zone reset time, and the mapping of channels and ways to zones, making it highly customizable. As an open-source tool available on GitHub, it allows researchers to modify and extend both hardware and software aspects. ConfZNS's extensive configurability and depth can make it complex for users unfamiliar with SSD architectures and emulation techniques. While it supports various SSD types, its primary focus on Zoned Namespace SSDs might limit its applicability for studies centered on other SSD types.

## 4.3 ZNS+

ZNS+ emulator [67] is a sophisticated tool designed to enhance the performance and efficiency of Zoned Namespace SSDs (ZNS). Developed in 2021 by researchers from Sungkyunkwan University and Samsung Electronics, ZNS+ introduces an advanced interface that supports in-storage zone compaction to alleviate the overhead associated with segment compaction in log-structured file systems (LFS). By enabling the host to offload data copy operations to the SSD, ZNS+ significantly improves file system performance, achieving up to 2.91X better efficiency compared to conventional ZNS-based storage systems.

ZNS+ offers several strengths that make it a compelling choice for SSD simulation. More that performance enhancement, reducing the host overhead, and efficient in-storage zone compaction which were mentioned above, the inclusion of a sparse sequential overwrite feature further improves write operations, boosting overall system performance. However, these benefits come with certain challenges. The advanced features of ZNS+ may introduce additional complexity in management and optimization, and its full potential is best realized with log-structured file systems, which may limit its compatibility with other file systems. Additionally, as a relatively new technology, ZNS+ may encounter compatibility issues with existing infrastructure and software not designed to support its features.

## 4.4 FEMU

FEMU [68] is a QEMU-based flash emulator introduced in 2018 to address the limitations of existing SSD emulators by offering a scalable, accurate, and cost-effective solution. Developed by researchers from the University of Chicago, Parallel Machines, and CNEX Labs, FEMU supports various workloads and configurations, including OCSSD. FEMU operates by emulating the SSD at the block device level, allowing it to interact with the guest OS as if it were a real SSD. It achieves this by using techniques such as exitless interrupts and optimized I/O threads to minimize latency and maximize scalability, effectively supporting up to 32 parallel channels or chips. This versatility allows FEMU to emulate different types of SSD interfaces and operations effectively, providing a robust platform for both internal SSD and split-level research.

FEMU is extensible, supporting internal-SSD, kernel-level, and split-level research, which makes it a valuable tool for diverse research applications. The emulator delivers relatively accurate results with a variance of 0.5-38%, serving as an effective drop-in replacement for Open Channel SSDs, especially for kernel prototyping and other research scenarios where real devices are unavailable. Additionally, FEMU supports multiple workloads, making it suitable for comprehensive studies across various storage solutions. However, users should be aware that FEMU's complex setup requires a deep understanding of QEMU and SSD architecture, and despite its optimizations, there may still be some performance overheads when compared to actual hardware. Furthermore, the emulator's performance and scalability can be influenced by the capabilities of the host system, such as CPU speed and memory.

## 4.5 FlexDrive

FlexDrive [69], introduced in 2016 by a team at Samsung Semiconductor, Inc., is a comprehensive and customizable emulation framework designed to evaluate and optimize NVMe-based SSDs. It supports a variety of workloads, including database benchmarks like Pgbench and the Aerospike Certification Tool (ACT), enabling detailed performance assessments. The emulator operates by accurately controlling key storage performance metrics such as latency and bandwidth, allowing for precise simulation of different SSD behaviors under various conditions.

FlexDrive offers significant advantages, including high flexibility with extensive reconfigurability through its latency and bandwidth controls, allowing for precise SSD performance management. It supports real application testing, providing practical insights into performance and compatibility. FlexDrive is more cost-effective and easier to implement and operate than hardware emulators like FPGAs. Its modular design enhances versatility, making it adaptable to various SSD interfaces such as NVMe/PCIe. Additionally, FlexDrive achieves near-native host machine speed with minimal overhead, making it efficient for high-speed NVMe operations. It also supports scalability studies, making data center capacity planning and optimization valuable. However, some challenges include potential CPU overhead due to spin locks in the latency model, complex configuration requirements that may be challenging for less experienced users, and kernel dependency that could lead to compatibility issues with new versions.

## 4.6 FSSD-EM

The FSSD-EM (FPGA-based SSD emulator with energy modeling) [70] is an advanced emulation platform designed to accurately mimic real SSD behaviors, developed in 2023 by a research group utilizing the Xilinx Virtex UltraScale+ FPGA VCU118 Evaluation Board. FSSD-EM connects to the host machine via a PCIe Gen3 x4 interface, ensuring high-speed data transfer and efficient performance evaluation. The primary aim of FSSD-EM is to facilitate research and development by enabling detailed analysis and optimization of SSD-centric applications under realistic workload conditions. The operation of the FSSD-EM involves three key components: the Host Interface Layer (HIL), the SSD Emulator Controller, and the Flash Emulator. The HIL interacts with the host machine, receiving read and write requests and generating corresponding SSD requests with page granularity, utilizing a Direct Memory Access (DMA) engine for faster data transfer. The SSD Emulator Controller manages the Flash Translation Layer (FTL) and emulates the embedded DRAM cache purely in hardware logic. The Flash Emulator simulates the access time of a NAND storage device using a timing model on the FPGA, with DDR memory serving as the storage backend. This design allows the emulator to operate under the same conditions as real SSDs, enhancing the accuracy and reliability of the emulation.

FSSD-EM includes an energy analysis module, offering valuable insights into the energy consumption of different SSD architectures and workloads. As an open-source project, FSSD-EM is designed to benefit both the industry and research communities by enabling extensive SSD-related studies. However, FSSD-EM also has some weaknesses. Its complexity, requiring detailed configuration and parameter adjustments, may be challenging for new users. Despite improvements, it still exhibits higher error rates for certain operations, such as write requests, due to varying erasing policies and garbage collection algorithms. The FPGA-based nature of the emulator demands significant computational resources and expertise in hardware programming, which may limit its accessibility.

### 4.7 Emulating Realistic Flash Device Errors with High Fidelity

This emulation framework [71], was developed in 2016 by a team from New Mexico State University, and the University of Utah. This framework is designed to emulate the erroneous states of SSDs precisely, enabling a thorough evaluation of the storage software stack's resilience to failures. The framework works by first modeling device behaviors reported in previous studies and creating a database of realistic error patterns. It then manipulates I/O commands at the driver level to emulate these errors with minimal disturbance to the target software, using a modified iSCSI driver to decouple the storage software from the block device.

This SSD emulator offers several strengths, including high-fidelity emulation that accurately reproduces SSD error behaviors, providing realistic scenarios for testing. It minimizes system disturbance by decoupling the storage software from the block device using the iSCSI protocol, reducing interference with the target system. However, there are some weaknesses, such as the complex setup that requires detailed knowledge of SSD behaviors and the iSCSI protocol for accurate emulation and integration. The emulation process might introduce performance overhead, potentially affecting the target system during testing.

## 5 Hardware Platforms

Hardware platforms play a pivotal role in the evaluation and development of SSD technology by providing tangible environments for testing and validating SSD architectures, algorithms, and performance metrics. Unlike simulators and emulators, hardware platforms offer direct interaction with real or prototype SSD hardware, allowing for more accurate and comprehensive experimentation. These platforms enable researchers and engineers to test SSDs under actual operating conditions, revealing insights that simulations or emulations may overlook, such as thermal effects, wear behavior, and real-world latency issues. Additionally, hardware platforms are critical for firmware development, benchmarking, and optimizing SSD components at both the hardware and software levels. In this section, we will review key hardware platforms used in SSD research and development to highlight their significance in advancing SSD technology.

### 5.1 OpenSSD Platforms (Jasmine & Cosmos/Cosmos+)

The OpenSSD project [72][73][74], established in 2011, provides an open-source platform for developing and testing SSD technologies. Initially, the Jasmine OpenSSD platform was introduced, allowing modification of firmware but not hardware. Jasmine is a reference implementation of SATA SSD based on the Indilinx (now OCZ Storage Solutions) Barefoot controller, which is an ARM-based SSD controller used in numerous high-performance SSD devices. In 2014, the Cosmos OpenSSD was released, enabling both hardware and software modifications (through the use of an FPGA). Later this platform was updated to Cosmos+ which has an updated controller. Cosmos is a PCIe-based SSD device based on the HYU Tiger3 controller. Cosmos+ is very similar to Cosmos but uses the HYU Tiger4 controller instead. Both Tiger3 and

Tiger4 are developed by the ENC (Embedded and Network Computing) Lab. led by Prof. Yong Ho Song at Hanyang University, Korea. The controller is synthesized in an FPGA placed on the board and could be easily modified as well. The project aims to facilitate research and development by providing fully accessible source codes for both hardware and software, allowing users to build and customize their SSD controllers. The OpenSSD platforms support various workloads, making them suitable for diverse applications and research needs. Jasmine utilizes the SATA interface and Cosmos and Cosmos+ both implement a subset of the NVMe interface.

OpenSSD platforms offer a range of strengths, including high customizability, enabling users to modify both hardware and software for extensive experimentation and development. Their open-source nature ensures fully accessible source codes for hardware and firmware, fostering transparency and collaboration in research. The platforms are ideal for advanced research in SSD technologies, such as firmware development and performance evaluation. They act as a fully functioning SSD device in the system, making them versatile for different applications, and enhances debugging, reducing SSD firmware test time and simplifying the development process. Additionally, These platforms serve as excellent educational tools for learning about SSD architecture and firmware development.

## 5.2 BlueSSD

BlueSSD [75], introduced in 2010, is an open platform for cross-layer experiments on NAND flash-based SSD architectures, developed by a collaborative group from Seoul National University, MIT, and UCSD. This platform provides a comprehensive environment for designing, implementing, and evaluating hardware and software components efficiently. The platform operates using an FPGA-based system, allowing users to test and modify both hardware and software components. It includes a software framework for developing flash firmware and employs the Bluespec programming language for hardware design, ensuring precise interface definitions. The architecture of BlueSSD enables detailed exploration of SSD performance, garbage collection processes, and FTL behaviors, making it a powerful tool for SSD research and development. However, the simulator's complexity, requiring knowledge of both hardware and software design, can be challenging for some users. It necessitates specific FPGA boards and hardware components, which may not be readily available to all researchers. Additionally, its reliance on the Bluespec language might limit accessibility for those unfamiliar with it.

## 5.3 OpenExpress

OpenExpress [76] is an innovative open-source NVMe host accelerator IP designed for integration with FPGA boards, enabling the development of customizable NVMe devices without software intervention. Developed by the CAMELab research group from Korea Advanced Institute of Science and Technology (KAIST) and introduced in 2020, this platform automates NVMe control logic in hardware to facilitate high-performance storage system research. OpenExpress supports various workloads, demonstrating superior performance compared to Optane SSDs, particularly in read-intensive tasks. The emulator within OpenExpress operates by fully automating the NVMe control logic, eliminating the need for software intervention during concurrent NVMe requests. It employs hardware modules for queue dispatching, data transferring, and completion handling. These modules fetch and decode incoming I/O requests, manage DMA for both host and backend memories, and handle NVMe completion queues. This hardware-centric approach allows OpenExpress to deliver a maximum bandwidth of around 7GB/s, significantly enhancing the performance of backend memory modules over PCIe.

Its hardware automation eliminates the need for software intervention in processing NVMe requests, reducing latency and increasing efficiency. As an open-source platform, it provides an accessible and versatile tool for academic and non-commercial research. It supports scalable data submission by handling multiple DRAM channels and modules as a storage backend, making it capable of managing rich outstanding NVMe commands and queue management. As a cost-effective alternative to proprietary NVMe IP cores, OpenExpress makes advanced NVMe research more accessible. However, its hardware-centric approach may require significant expertise in FPGA programming and hardware design. Despite its optimization for high performance, FPGAs are generally slower than custom ASICs, potentially limiting peak performance. The initial setup and configuration on an FPGA board can be time-consuming and challenging for new users.

## 5.4 FlashBench

FlashBench [77] is a comprehensive development environment introduced in 2012 by a team from Seoul National University, designed to facilitate the rapid creation and testing of flash-based storage devices. It supports multiple workloads through various design levels, enabling seamless hardware and software integration. The platform allows for the simulation of SSD performance, including the evaluation of algorithms, system architecture, and hardware components under different operational scenarios. FlashBench is particularly useful for testing the behavior of SSDs under diverse workloads, offering insights into performance bottlenecks and optimization opportunities.

FlashBench offers several advantages as a comprehensive environment for developing and testing flash-based storage devices. This capability is crucial for evaluating performanceunder different conditions. Additionally, its seamless integration between hardware and software enhances the efficiency of the development and testing processes, enabling in-depth analysis to identify performance bottlenecks and optimization opportunities. However, FlashBench also has some disadvantages, such as its complexity. Furthermore, its resource-intensive nature requires significant computational power and time for simulating complex workloads, and its focus on flash-based storage devices may limit its applicability to other storage technologies.

## 6 Conclusion

In this survey, we explored various SSD simulators, emulators, and hardware platforms that are instrumental in advancing SSD research and development. Each of these tools serves a unique purpose: simulators allow for the fast modeling and testing of SSD architectures, emulators offer a practical environment for validating firmware and algorithms, and hardware platforms provide a direct interface with physical devices for real-world performance evaluation. Together, these tools offer researchers and developers a comprehensive suite of options for understanding and optimizing SSD behavior under diverse conditions.

As SSD technology continues to evolve, the interplay between simulation, emulation, and hardware testing will remain vital in addressing challenges such as improving efficiency, reliability, and longevity. Future research should focus on integrating these tools more seamlessly and developing hybrid approaches that maximize their potential, ultimately leading to the creation of more advanced, high-performance SSD solutions.

## References

[1] Cho, S., Kim, B., Cho, H., Seo, G., Mutlu, O., Kim, M., Park, J.: Aero: Adaptive erase operation for improving lifetime and performance of modern nand flash-based ssds. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, pp. 101–118 (2024)

[2] Tavakkol, A., Gómez-Luna, J., Sadrosadati, M., Ghose, S., Mutlu, O.: {MQSim}: A framework for enabling realistic studies of modern {Multi-Queue}{SSD} devices. In: 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 49–66 (2018)

[3] Cai, Y., Ghose, S., Haratsch, E.F., Luo, Y., Mutlu, O.: Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. Proceedings of the IEEE **105**(9), 1666–1704 (2017)

[4] Chang, Y.-H., Lin, J.-H., Hsieh, J.-W., Kuo, T.-W.: A strategy to emulate nor flash with nand flash. ACM Transactions on Storage (TOS) **6**(2), 1–23 (2010)

[5] Bez, R., Camerlenghi, E., Modelli, A., Visconti, A.: Introduction to flash memory. Proceedings of the IEEE **91**(4), 489–502 (2003)

[6] Alsalibi, A.I., Mittal, S., Al-Betar, M.A., Sumari, P.B.: A survey of techniques for architecting slc/mlc/tlc hybrid flash memory–based ssds. Concurrency and Computation: Practice and Experience **30**(13), 4420 (2018)

[7] Cai, Y., Ghose, S., Luo, Y., Mai, K., Mutlu, O., Haratsch, E.F.: Vulnerabilities in mlc nand flash memory programming: Experimental analysis, exploits, and mitigation techniques. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 49–60 (2017). IEEE

[8] Cai, Y., Luo, Y., Haratsch, E.F., Mai, K., Mutlu, O.: Data retention in mlc nand flash memory: Characterization, optimization, and recovery. In: 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), pp. 551–563 (2015). IEEE

[9] Jin, Y., Lee, B.: A comprehensive survey of issues in solid state drives. Advances in computers **114**, 1–69 (2019)

[10] Lim, S.-P., Lee, S.-W., Moon, B.: Faster ftl for enterprise-class flash memory ssds. In: 2010 International Workshop on Storage Network Architecture and Parallel I/Os, pp. 3–12 (2010). IEEE

[11] Nadig, R., Sadrosadati, M., Mao, H., Ghiasi, N.M., Tavakkol, A., Park, J., Sarbazi-Azad, H., Luna, J.G., Mutlu, O.: Venice: Improving solid-state drive parallelism at low cost via conflict-free accesses. In: Proceedings of the 50th Annual International Symposium on Computer Architecture, pp. 1–16 (2023)

[12] Meza, J., Wu, Q., Kumar, S., Mutlu, O.: A large-scale study of flash memory failures in the field. ACM SIGMETRICS Performance Evaluation Review **43**(1), 177–190 (2015)

[13] Schroeder, B., Lagisetty, R., Merchant, A.: Flash reliability in production: The expected and the unexpected. In: 14th USENIX Conference on File and Storage Technologies (FAST 16), pp. 67–80 (2016)

[14] Narayanan, I., Wang, D., Jeon, M., Sharma, B., Caulfield, L., Sivasubramaniam, A., Cutler, B., Liu, J., Khessib, B., Vaid, K.: Ssd failures in datacenters: What? when? and why? In: Proceedings of the 9th ACM International on Systems and Storage Conference, pp. 1–11 (2016)

[15] Mielke, N., Belgal, H.P., Fazio, A., Meng, Q., Righos, N.: Recovery effects in the distributed cycling of flash memories. In: 2006 IEEE International Reliability Physics Symposium Proceedings, pp. 29–35 (2006). IEEE

[16] Modelli, A., Visconti, A., Bez, R.: Advanced flash memory reliability. In: 2004 International Conference on Integrated Circuit Design and Technology (IEEE Cat. No. 04EX866), pp. 211–218 (2004). IEEE

[17] Agrawal, N.: Design tradeoffs for ssd performance. In: USENIX ATC (2008)

[18] Cai, Y., Haratsch, E.F., Mutlu, O., Mai, K.: Threshold voltage distribution in mlc nand flash memory: Characterization, analysis, and modeling. In: 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1285–1290 (2013). IEEE

[19] Park, J., Azizi, R., Oliveira, G.F., Sadrosadati, M., Nadig, R., Novo, D., Gómez-Luna, J., Kim, M., Mutlu, O.: Flash-cosmos: In-flash bulk bitwise operations using

inherent computation capability of nand flash memory. In: 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 937–955 (2022). IEEE

[20] Cai, Y., Haratsch, E.F., Mutlu, O., Mai, K.: Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In: 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 521–526 (2012). IEEE

[21] Zuck, A., Toledo, S., Sotnikov, D., Harnik, D.: Compression and {SSDs}: Where and how? In: 2nd Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 14) (2014)

[22] Li, J., Zhao, K., Zhang, X., Ma, J., Zhao, M., Zhang, T.: How much can data compressibility help to improve {NAND} flash memory lifetime? In: 13th USENIX Conference on File and Storage Technologies (FAST 15), pp. 227–240 (2015)

[23] Chung, T.-S., Park, D.-J., Park, S., Lee, D.-H., Lee, S.-W., Song, H.-J.: A survey of flash translation layer. Journal of Systems Architecture **55**(5-6), 332–343 (2009)

[24] Gupta, A., Kim, Y., Urgaonkar, B.: Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. Acm Sigplan Notices **44**(3), 229–240 (2009)

[25] Chang, L.-P., Kuo, T.-W., Lo, S.-W.: Real-time garbage collection for flash-memory storage systems of real-time embedded systems. ACM Transactions on Embedded Computing Systems (TECS) **3**(4), 837–863 (2004)

[26] Yang, M.-C., Chang, Y.-M., Tsao, C.-W., Huang, P.-C., Chang, Y.-H., Kuo, T.-W.: Garbage collection and wear leveling for flash memory: Past and future. In: 2014 International Conference on Smart Computing, pp. 66–73 (2014). IEEE

[27] Gal, E., Toledo, S.: Algorithms and data structures for flash memories. ACM Computing Surveys (CSUR) **37**(2), 138–163 (2005)

[28] Chang, L.-P.: On efficient wear leveling for large-scale flash-memory storage systems. In: Proceedings of the 2007 ACM Symposium on Applied Computing, pp. 1126–1130 (2007)

[29] Micheloni, R., Crippa, L., Marelli, A., Silvagni, A.: Nand ddr interface. Inside NAND Flash Memories, 161–196 (2010)

[30] Zhao, K., Zhao, W., Sun, H., Zhang, X., Zheng, N., Zhang, T.: {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), pp. 243–256 (2013)

[31] Tanakamaru, S., Yanagihara, Y., Takeuchi, K.: Error-prediction ldpc and error-recovery schemes for highly reliable solid-state drives (ssds). IEEE Journal of Solid-State Circuits **48**(11), 2920–2933 (2013)

[32] Park, J., Kim, M., Chun, M., Orosa, L., Kim, J., Mutlu, O.: Reducing solid-state drive read latency by optimizing read-retry. In: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 702–716 (2021)

[33] Cai, Y., Ghose, S., Haratsch, E.F., Luo, Y., Mutlu, O.: Reliability issues in flash-memory-based solid-state drives: Experimental analysis, mitigation, recovery. Inside Solid State Drives (SSDs), 233–341 (2018)

[34] Cha, J., Kang, S.: Data randomization scheme for endurance enhancement and interference mitigation of multilevel flash memory devices. Etri Journal **35**(1), 166–169 (2013)

[35] Rollins, D., *et al.*: A comparison of client and enterprise ssd data path protection. Micron Technology, Inc **2**, 1–3 (2011)

[36] Peterson, W.W., Brown, D.T.: Cyclic codes for error detection. Proceedings of the IRE **49**(1), 228–235 (1961)

[37] Cai, Y., Mutlu, O., Haratsch, E.F., Mai, K.: Program interference in mlc nand flash memory: Characterization, modeling, and mitigation. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 123–130 (2013). IEEE

[38] Cai, Y., Yalcin, G., Mutlu, O., Haratsch, E.F., Unsal, O., Cristal, A., Mai, K.: Neighbor-cell assisted error correction for mlc nand flash memories. ACM SIGMETRICS Performance Evaluation Review **42**(1), 491–504 (2014)

[39] Kim, C., Ryu, J., Lee, T., Kim, H., Lim, J., Jeong, J., Seo, S., Jeon, H., Kim, B., Lee, I., *et al.*: A 21 nm high performance 64 gb mlc nand flash memory with 400 mb/s asynchronous toggle ddr interface. IEEE Journal of Solid-State Circuits **47**(4), 981–989 (2012)

[40] Dirik, C., Jacob, B.: The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. ACM SIGARCH Computer Architecture News **37**(3), 279–289 (2009)

[41] Mielke, N.R., Frickey, R.E., Kalastirsky, I., Quan, M., Ustinov, D., Vasudevan, V.J.: Reliability of solid-state drives based on nand flash memory. Proceedings of the IEEE **105**(9), 1725–1750 (2017)

[42] Grupp, L.M., Davis, J.D., Swanson, S.: The bleak future of nand flash memory. In: FAST, vol. 7, pp. 10–2 (2012)

[43] Mohan, V., Siddiqua, T., Gurumurthi, S., Stan, M.R.: How i learned to stop worrying and love flash endurance. In: 2nd Workshop on Hot Topics in Storage and File Systems (HotStorage 10) (2010)

[44] Boboila, S., Desnoyers, P.: Write endurance in flash drives: Measurements and analysis. In: FAST, pp. 115–128 (2010)

[45] Jimenez, X., Novo, D., Ienne, P.: Wear unleveling: Improving {NAND} flash lifetime by balancing page endurance. In: 12th USENIX Conference on File and Storage Technologies (FAST 14), pp. 47–59 (2014)

[46] Margaglia, F., Brinkmann, A.: Improving mlc flash performance and endurance with extended p/e cycles. In: 2015 31st Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–12 (2015). IEEE

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

[48] Lee, D., Hong, D., Choi, W., Kim, J.: Mqsim-e: An enterprise ssd simulator. IEEE Computer Architecture Letters **21**(1), 13–16 (2022)

[49] Jung, M., Zhang, J., Abulila, A., Kwon, M., Shahidi, N., Shalf, J., Kim, N.S., Kandemir, M.: Simplessd: Modeling solid state drives for holistic system simulation. IEEE Computer Architecture Letters **17**(1), 37–41 (2017)

[50] Yoo, J., Won, Y., Hwang, J., Kang, S., Choi, J., Yoon, S., Cha, J.: Vssim: Virtual machine based ssd simulator. In: 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–14 (2013). IEEE

[51] Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In: Proceedings of the International Conference on Supercomputing, pp. 96–107 (2011)

[52] Kim, Y., Tauras, B., Gupta, A., Urgaonkar, B.: Flashsim: A simulator for nand flash-based solid-state drives. In: 2009 First International Conference on Advances in System Simulation, pp. 125–131 (2009). IEEE

[53] He, J., Kannan, S., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: The unwritten contract of solid state drives. In: Proceedings of the Twelfth European Conference on Computer Systems, pp. 127–144 (2017)

[54] Jung, M., Choi, W., Gao, S., Wilson III, E.H., Donofrio, D., Shalf, J., Kandemir, M.T.: Nandflashsim: High-fidelity, microarchitecture-aware nand flash memory simulation. ACM Transactions on Storage (TOS) **12**(2), 1–32 (2016)

[55] Malladi, K.T., Chang, M.-T., Niu, D., Zheng, H.: Flashstoragesim: Performance modeling for ssd architectures. In: 2017 International Conference on Networking, Architecture, and Storage (NAS), pp. 1–2 (2017). IEEE

[56] Dayan, N., Svendsen, M.K., Bjorling, M., Bonnet, P., Bouganim, L.: Eagletree: Exploring the design space of ssd-based algorithms. arXiv preprint arXiv:1401.6360 (2014)

[57] Ahn, H., Lee, Y., Lee, K.-H.: Srsim: A simulator for ssd-based raid. In: International Conference: Beyond Databases, Architectures and Structures, pp. 610–620 (2015). Springer

[58] Zuolo, L., Zambelli, C., Micheloni, R., Indaco, M., Di Carlo, S., Prinetto, P., Bertozzi, D., Olivo, P.: Ssdexplorer: A virtual platform for performance/reliability-oriented fine-grained design space exploration of solid state drives. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **34**(10), 1627–1638 (2015)

[59] Desnoyers, P.: Analytic modeling of ssd write performance. In: Proceedings of the 5th Annual International Systems and Storage Conference, pp. 1–10 (2012)

[60] Jiao, Z., Bhimani, J., Kim, B.S.: Wear leveling in ssds considered harmful. In: Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems, pp. 72–78 (2022)

[61] Prabhakaran, V., Wobber, T.: Ssd extension for disksim simulation environment. Microsoft Reseach (2009)

[62] Bucy, J.S., Ganger, G.R., *et al.*: The DiskSim Simulation Environment Version 3.0 Reference Manual. School of Computer Science, Carnegie Mellon University, ??? (2003)

[63] Poremba, M., Xie, Y.: Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In: 2012 IEEE Computer Society Annual Symposium on VLSI, pp. 392–397 (2012). IEEE

[64] Choi, W., Zhang, J., Gao, S., Lee, J., Jung, M., Kandemir, M.: An in-depth study of next generation interface for emerging non-volatile memories. In: 2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA), pp. 1–6 (2016). IEEE

[65] Kim, S.-H., Shim, J., Lee, E., Jeong, S., Kang, I., Kim, J.-S.: {NVMeVirt}: A versatile software-defined virtual {NVMe} device. In: 21st USENIX Conference on File and Storage Technologies (FAST 23), pp. 379–394 (2023)

[66] Song, I., Oh, M., Kim, B.S.J., Yoo, S., Lee, J., Choi, J.: Confzns: A novel emulator for exploring design space of zns ssds. In: Proceedings of the 16th ACM

International Conference on Systems and Storage, pp. 71–82 (2023)

[67] Han, K., Gwak, H., Shin, D., Hwang, J.: Zns+: Advanced zoned namespace interface for supporting in-storage zone compaction. In: 15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21), pp. 147–162 (2021)

[68] Li, H., Hao, M., Tong, M.H., Sundararaman, S., Bjørling, M., Gunawi, H.S.: The {CASE} of {FEMU}: Cheap, accurate, scalable and extensible flash emulator. In: 16th USENIX Conference on File and Storage Technologies (FAST 18), pp. 83–90 (2018)

[69] Malladi, K.T., Awasthi, M., Zheng, H.: Flexdrive: A framework to explore nvme storage solutions. In: 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1115–1122 (2016). IEEE

[70] Yu, L.: Fssd-em: Fpga-based ssd emulator with energy modeling. PhD thesis, University of Illinois at Urbana-Champaign (2023)

[71] Wang, S., Cao, J., Murillo, D.V., Shi, Y., Zheng, M.: Emulating realistic flash device errors with high fidelity. In: 2016 IEEE International Conference on Networking, Architecture and Storage (NAS), pp. 1–2 (2016). IEEE

[72] Kwak, J., Lee, S., Park, K., Jeong, J., Song, Y.H.: Cosmos+ openssd: Rapid prototype for flash storage systems. ACM Transactions on Storage (TOS) **16**(3), 1–35 (2020)

[73] Jung, T., Lee, Y., Shin, I.: Openssd platform simulator to reduce ssd firmware test time. Life Science Journal **11**(7) (2014)

[74] Software, S., SW STAR Lab. project., A.L.: The OpenSSD Project Platforms. http://www.openssd-project.org/platforms/ Accessed 2024-07-29

[75] Lee, S., Fleming, K., Park, J., Ha, K., Caulfield, A., Swanson, S., Kim, J., *et al.*: Bluessd: An open platform for cross-layer experiments for nand flash-based ssds. In: WARP-5th Annual Workshop on Architectural Research Prototyping (2010)

[76] Jung, M.: {OpenExpress}: Fully hardware automated open research framework for future fast {NVMe} devices. In: 2020 USENIX Annual Technical Conference (USENIX ATC 20), pp. 649–656 (2020)

[77] Lee, S., Park, J., Kim, J.: Flashbench: A workbench for a rapid development of flash-based storage devices. In: 2012 23rd IEEE International Symposium on Rapid System Prototyping (RSP), pp. 163–169 (2012). IEEE