Universitá degli studi di Ferrara

dottorato di ricerca in scienze dell'ingegneria

Ciclo XXVIII

Coordinatore: Prof. Stefano Trillo
Settore Scientifico Disciplinare: ING-INF/01

Solid State Drives: design challenges for optimum performance-reliability trade-off

Relatore
Prof. Olivo Piero

Dottorando
Zuolo Lorenzo

Correlatore
Dott. Zambelli Cristian

Anni 2012/2015

Universitá degli studi di Ferrara

dottorato di ricerca in scienze dell'ingegneria

Ciclo XXVIII

Coordinatore: Prof. Stefano Trillo
Settore Scientifico Disciplinare: ING-INF/01

Solid State Drives: design challenges for optimum performance-reliability trade-off

Relatore
Prof. Olivo Piero

Dottorando
Zuolo Lorenzo

Correlatore
Dott. Zambelli Cristian

Anni 2012/2015

Alla mia famiglia e in particolar modo a mio padre,
per aver creduto in me anche durante i momenti di dubbio e sconforto.

Al Prof. Piero Olivo e al Dott. Cristian Zambelli,
per avermi sempre sostenuto e aiutato durante il lavoro di ricerca.

All Ing. Rino Micheloni,
per avermi aiutato a contestualizzare questo lavoro nel panorama industriale.

Al Prof. Davide Bertozzi,
per avermi insegnato a meravigliarmi e a stupirmi sempre.

Introduction

Solid State Drives (SSDs) are one of the electronic systems with the higher development rate in the last decade: they are widely used in hyperscale systems such as cloud computing and big data servers, where performance is a constraint, as well as in consumer electronics by replacing traditional Hard Disk Drives (HDD).

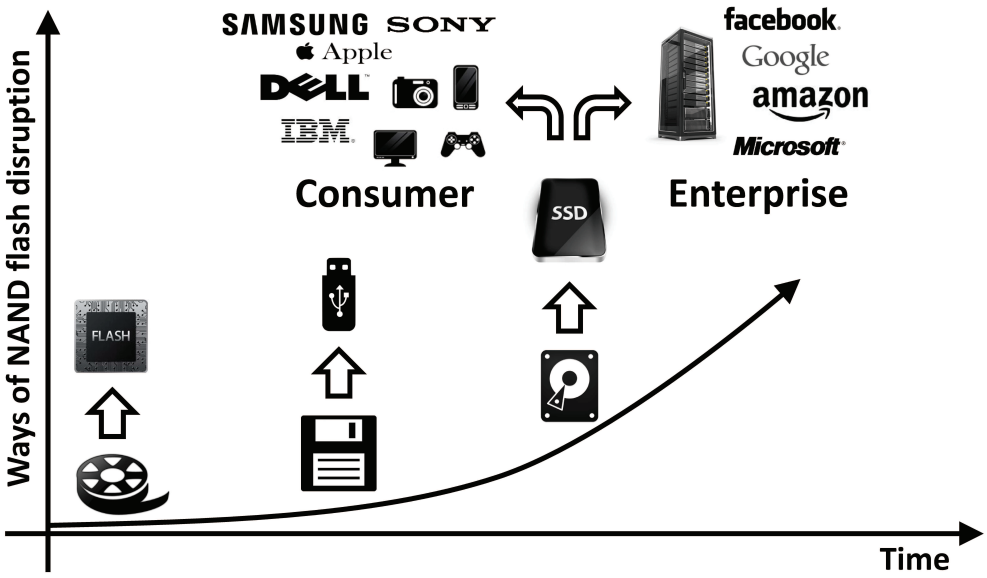


Figure 1: Ways of NAND flash memories disruption.

SSDs' design, in the last 5 years, faced an extraordinary evolution caused by the continuous development of NAND Flash memories representing their storage medium. With this respect, as shown in Fig. 1, NAND flash memories have completely transformed the way information is processed and stored. Starting as film and tape replacement for cameras and voice recorders, NAND flash memories rapidly superseded traditional magnetic storage supports and now they represent an obliged choice for high-performance storage solutions. The availability of NAND flash-based SSDs materialized, in the last 5 years,

as an astonishing proliferation of global-scaled Corporations whose commercial strength is tightly coupled to an innovative SSD design methodology thought for big data centers and cloud computing.
The previous developing strategy of SSDs, in fact, was based on a full compatibility with HDD and therefore the SSD's performance optimization was focused on that of the Flash Translation Layer (FTL), the firmware managing the basic memory operations. This approach was made possible by the use of sufficiently reliable Single Levels Cells (SLC) NAND memories, storing a single bit per cell in the traditional 0/1 digital paradigm, with a low read error probability, thus requiring the design of simple engines for Error Correction Codes (ECC). The SATA protocol interfacing the memory system and the host was sufficient to guarantee the requested Quality of Service (QoS), that is the ability of keeping a sustained performance over time within a defined threshold.
As a whole, the SSD architecture optimization and the development of dedicated CAD tools for the SSD design space explorations were FTL-oriented, in a Top-Down approach.

In the last few years, on the contrary, the need for SSDs with higher and higher storage capacities and performance joined to the availability of high density NAND Flash memories able to store 2 or even 3 bits in a single cell, moved the design paradigm from a Top-Down to a Bottom-Up approach where the performance and the reliability of the storage media dictate the design constraints.
NAND Flash memories with scaled technologies, in fact, suffer from several physical mechanisms able to impact on their reliability figures such as Endurance, that is the minimum number of Program/Erase (P/E) cycles that the memory can withstand before leading to a failure, Data Retention, denoting the ability of a memory to keep a stored information over time with no biases applied, and the immunity from Read Disturbs, representing the stress suffered by a cell when reading neighbor cells.

P/E operations in NAND flash cells rely on charge transport through thin oxides via Fowler-Nordheim tunneling into/from a storage layer. Electron tunneling is responsible for a slow, but continuous, oxide wear out causing undesired charge flowing into/from the storage layer. As the number of P/E cycles increases, these effects strongly impact on the writing operations. To counter endurance effects, sophisticated (but slow and power hungry) algorithms are adopted to tightly control the amount of charge transferred into/from the storage layer. However, the relentless oxide degradation strongly affects the ability of keeping unaltered the charge content into the storage layers for extremely long times, a mandatory requirement to fulfill the non-volatile paradigm.

These reliability issues become more and more significant in Multi-Level Cells (MLC) and Triple-Level Cells (TLC) storing 2 and 3 bits per cell, respectively, where the undesired transfer of a very small amount of charge into/from the storage layer may alter significantly the memory information content.

The basic parameter characterizing the NAND Flash memory reliability is the Raw Bit Error Rate (RBER), representing the percentage of erroneous bits retrieved during a read operation. The knowledge of this parameter, whose value increases with technology scaling, with the number of bits that a cell can store, with the number of P/E cycles, with the time elapsed between two successive read operations, with the number of repeated read operations on the same memory location, is now the driver for architectural and software design of present SSDs.

MLC and TLC NAND Flash memories require the availability of an ECC scheme able to correct the relatively high number of errors detected when reading the memory. The selection of the ECC code and the design of the correction engine represent the key point for present SSDs' design since they must be carefully calibrated with respect to the figures of merit of the selected non-volatile memories: a too simple ECC scheme may not be able to guarantee a suitable reliability, whereas a too complex ECC scheme may reduce severely the read bandwidth because of the time required for error correction, with a consequent impact also on the system power consumption. On the basis of the selected ECC code and of the designed ECC engine, the optimal error reduction algorithms may be identified.

The joint selection of the appropriate NAND Flash memories and the design of the adequate ECC scheme represent the key point to guarantee a high QoS for the SSD to be designed.

Once the ECC scheme has been designed, the Bottom-Up design flow rises to the memory controller, representing the interface towards the ECC engine and the memory storage system. The read bandwidth provided by the ECC block must be guaranteed by the controller, to avoid that the design efforts devoted to optimize the ECC scheme vanish. With this respect, the SSD controller must be designed in order to manage a sufficient amount of commands to fully exploit the bandwidth of the underlying storage system. Similarly, also the interface towards the host must be able to guarantee the expected read bandwidth. For this reason, SATA protocol is no longer able to deal with the performance made available the other blocks in the SSD architecture (MLC NAND Flash memories, ECC engine and SSD controller).

On the basis of this Bottom-Up SSD's design flow, from an accurate knowledge of the performance and limits of the selected NAND memories to the design of a suitable ECC engine and, successively to that of the controller and of the host interface, also CAD tools for SSD design must follow this

Bottom-Up vision while relaxing the efforts previously devoted to the FTL design.

In this thesis, thanks to a dedicated tool for SSDs design space exploration which follows the previously mentioned Bottom-Up approach, several aspects related to the design and the optimization of an SSD architecture will be presented. With this respect, it will be shown how an SSD works, that is on the basis of the present technical and economic revolution driven by cloud storage and big data centers. Readers will get all the elements to understand:

- how to accurately simulate an SSD in terms of bandwidth and latency
- how to efficiently design a NAND flash-based SSD and why its performance rapidly decrease with use and time
- why the SSD design has to be changed when emerging non-volatile memories are considered as the storage medium

Finally, the thesis will speculate on future research opportunities made available by this work focusing on the simulation of the SSD power consumption and on the study of flexible an high-performing storage architectures constructed from SSDs and Multi Purpose Processing Arrays (MPPA) systems.

Contents

Chapter 1

SSDExplorer: a virtual platform for SSD simulation

Solid State Drives (SSDs) are becoming popular, driven by the restless growth of high performance and cloud computing [1]. The development of a SSD architecture implies the analysis of important trade-offs that, if properly understood, may contribute to tighten the SSD design space, thus reducing the prototyping effort. Although SSD hardware prototyping platforms may capture realistic storage system behaviors, they suffer from an intrinsic lack of flexibility with respect to the available SSD design choices [2].

To tackle this challenge and to identify optimal design points meeting target performance goals under given cost constraints, the SSD research community increasingly relies on sophisticated software tools that enable modeling and simulation of SSD platforms. Among them, two categories of tools have been proposed: disk emulation tools [3] in virtual environments [4], and pure software simulation tools [5]. The former category uses functional simulation to obtain fast performance evaluation of the SSD in a host environment. This comes at the cost of constrained design space exploration capabilities due to the use of abstract simulation models. The latter category exploits trace driven simulators obtaining a steady-state performance analysis of the disk. However, these tools often overlook the macroscopic performance/reliability implications of some key component parameters or subtle microarchitecture-level effects. In fact, microarchitectural details are typically abstracted, thus preventing the analysis of the SSD performance with respect to its sub-components and their interaction efficiency.

In both categories of tools the common underlying assumption is that the SSD microarchitectural details that determine the disk behavior have been already defined. In these tools, the modeling framework pursues other goals rather than a fine-grained design space exploration (FGDSE) of the

microarchitecture, namely performance quantification of a known architecture, full system simulation, and flash translation layer (FTL) validation. These approaches are in general unsuitable to meet the requirements of a SSD designer, whose primary need is not the capability to perform functional simulation, but rather to quantify the efficiency of microarchitectural design choices to cope with long term concerns such as the disk reliability and wearout-induced performance drop.

Currently there is a gap in the landscape of simulation frameworks for SSD devices specifically targeting FGDSE. Bridging this gap is mandatory to avoid the over-design of a SSD architecture when trying to meet a target I/O performance requirement, to perform a pure reliability assessment or aiming at a trade-off between the two.

To achieve this goal in this thesis it is proposed SSDExplorer, a software framework, which complements modeling and simulation capabilities of state-of-the-art tools targeting the following main innovations:

- *Modeling of all components of the SSD architecture*, thus broadening and easing the design space exploration capabilities provided by competing tools. Moreover, a careful choice of the most suitable modeling abstraction for each SSD component is provided.

- *Accounting for the performance implications of the FTL without requiring its full implementation.* This is achieved, for high-end SSD controllers, by supporting the Write Amplification Factor (WAF) abstraction [6]. As a consequence, one of the goals of SSDExplorer is to deliver a fast path for accurate I/O performance quantification.

- *Delivering unprecedented insights into the performance equalization within the SSD architecture.* This includes accurate performance breakdowns and identification of microarchitectural bottlenecks, or unexploited parallelisms. Analysis of the interaction efficiency among subcomponents of a SSD is an essential requirement for microarchitecture design. SSDExplorer enables to search for resource-aware design points able to meet target I/O performance requirements, hence reducing the risk of costly overdesign.

- Accuracy of the simulated SSD architectures providing results validated with a mature commercial platform (i.e., OCZ vertex 120GB [7]) and a state-of-the-art enterprise platform [8].

These features enable a strategic analysis of the SSDs that will be demonstrated in this paper. Different disk architectures can be compared among

each other in an attempt to identify the design point that minimizes resource consumption while meeting target performance/reliability constraints.

1.1 Related Works

Understanding the behavior of SSDs to reproduce their functionality with dedicated frameworks is a growing challenge in the research community. Currently, publications mainly focus on disk *emulation* [3] and disk *trace-driven* simulation software [5, 9, 10, 11, 12, 13, 14].

Yoo et al. [3], propose a disk *emulation* strategy based on a reconfigurable framework able to deal with a real SSD. One of the key contributions of this work is the ability to track the real performance of a host system through a dynamic manager built around a QEMU virtual platform [4]. However, to achieve fast performance estimations, several components (i.e., the processor, the NAND Flash arrays, etc.) are described at a high abstraction level. Performance fluctuations experienced by these blocks are therefore lost, thus strongly reducing the performance estimation accuracy.

Moving to SSD *trace-driven* simulation tools, the open-source frameworks proposed in [5, 9] allow SSD performance and power consumption evaluation. Attempts to improve them in order to achieve real performance matching have also been proposed in [13, 14]. However, these tools are still highly abstracted, thus providing an insufficient level of simulation accuracy and realistic components description to perform real FGDSE. Moreover, since the aforementioned classes of frameworks do not model all the internal blocks of a SSD, they are able to accurately track the behavior of a disk only starting from a set of predefined and statically assigned timings (i.e., the channel switch delay, the I/O bus latencies, the command scheduler delay, etc.). An additional attempt to modify one of those tools in order to incorporate detailed NAND and ECC timings has been provided in [15]. Although the accuracy of the obtained results in that particular case study is high, those tools still lack the possibility to evaluate micro-architectural effects on the SSD performance like commands pipelining or suspension or uncommon queuing mechanisms, which are visible only if a cycle-accuracy paradigm is pursued.

To overcome this weakness, several cycle-accurate SSD simulators have been developed. Lee et al. [10] exploit a clock precision simulation for hardware components description. However, it does not allow a full modeling of all the components building a SSD, thus hiding some of the bottlenecks affecting the architecture. Other methods for fast simulation have been proposed in [11, 12], but they also suffer from accuracy loss due to the lack of a complete architectural modeling.

Table 1.1: Comparison between SSDExplorer and other SSD characterization approaches.

Reconfigurable parameters	**SSDExplorer Platform**	**Emulation Platforms**	**Trace-driven Platforms**	**Hardware Platforms**
Real FTL	√	√	√	√
WAF FTL	√	No	No	No
Host Interface performance	√	√	No	√
Real workload	√	√	No	√
Different Host Interfaces	√	No	√	No
Accurate DDR timings	√	No	No	No
Multi DRAM buffer	√	No	No	No
Configurable Channel Nº	√	√	√	No
Configurable Target Nº	√	√	√	No
NAND architecture	√	√	√	No
Accurate NAND timings	√	No	No	√
NAND Reliability	√	No	No	√
ECC Model	√	No	No	√
Interconnect model	√	No	No	√
Core model	√	No	No	√
Real firmware execution	√	No	No	√
Multi Core	√	No	No	No
Model refinement	√	No	No	No
Simulation Speed	Variable	High	High	Fixed

Hardware platform prototypes have been proposed as well in [2] and [16]. They enable a precise SSD behavior investigation, although their fixed architecture severely limits exploration of different design solutions. To this extent, the sole internal firmware modification is allowed.

What is really missing in all previous works is a clear exploration of the performance correlation between the host interface capabilities and the non-volatile memory subsystem involving all intermediate architectural blocks. Currently, the performance equalization of those chained components is overshadowed. Furthermore, since these modeling approaches are not oriented in the direction of a reliability projection of different SSD architectures, their usage is merely limited to a coarse functional simulation that may be strongly unaligned with the industrial needs.

To summarize, Table. 1.1 shows the main characteristics of SSDExplorer in the context of previous works in this field, by comparing relevant features of the available simulation frameworks. As it can be seen, SSDExplorer introduces detailed timings and behavioral models of the critical architectural blocks (i.e., DRAMs, NAND Flash memories, ECC sub-system, etc.) that are mandatory for an accurate performance/reliability evaluation. On the contrary, available state-of-the-art emulation and trace-driven platforms do not account for those models.

1.2 SSDExplorer at a glance

1.2.1 Modeling Strategy

One of the key concepts that drove the development of SSDExplorer has been the possibility to experience a unified, reconfigurable and multi-abstraction simulation environment. To achieve this goal, each block of SSDExplorer has been written and integrated using the SystemC modeling and simulation environment [17]. SystemC allows designers to cover, using a single description language, several model refinement layers ranging from the Timed Functional level up to the Register Transfer Level (RTL). Thanks to this feature, if a specific block must be thoroughly investigated, a more accurate model can be easily developed and plugged into the simulation environment without changing any other component. However, it is worth highlighting that, since the simulation speed offered by the SystemC scales inversely to the description level, the abstraction of each model must be wisely selected depending on the simulator goals in order to the maximize the simulation efficiency. Although a similar strategy was successfully described for other applications [18], this approach is devised for the first time in a FGDSE-dedicated SSD

simulation tool.

Therefore, SSDExplorer has been designed in order to: *i)* select the most suitable modeling style for each SSD component to accurately quantify the performance; *ii)* tolerate lack of precise implementations of specific HW/SW components without affecting the overall accuracy by providing suitable modeling abstractions. From this perspective, in fact, a detailed implementation of all SSD components might not be available when its architecture design space is explored. As a consequence of these considerations, it has been chosen to model with a high accuracy all HW/SW components that logically belong to the SSD control path (i.e., all the blocks involved in the command manipulation process), whereas components belonging to the datapath (i.e., components exploited during the data transfer phase) have been modeled in terms of the introduced processing/storage delays. This approach is successful both in improving the simulation speed while still capturing subtle micro architectural effects affecting SSD performance metrics, and in providing a backbone for FTL functional simulation. Finally, communication among each model domain is provided through tailored wrappers able to translate logical signals to state variables without impacting on the simulation framework.

1.2.2 Models exploited in SSDExplorer

Fig. 1.1 shows the SSD architecture template simulated by SSDExplorer. Three domains can be identified based on the selected modeling abstraction: *Pin-Accurate Cycle-Accurate* (PA-CA), *Transaction-Level Cycle-Accurate* (TL-CA), and *Parametric Time Delay* (PTD) models. It has been followed the terminology in [19] to specify abstraction layers.

Pin-Accurate Cycle-Accurate models

The key components that take part to the management of the data flow are the CPU, the system interconnect, and the channel/target controller. All these components are involved in the real execution of the SSD FTL (if available) or of its abstracted behavior. To this extent, a cycle-accurate design abstraction is used for modeling these components to accurately capture commands handled by the SSD and their timings. In such a way, the firmware overhead in terms of overall performance drop can also be easily and accurately evaluated.

Cycle-accurate models effectively capture the complete functionality, structure, communication, and timing of a micro-architectural block. The communication between components is modeled in a PA-CA manner: the hardware signals connecting a component to a bus are explicitly modeled, and

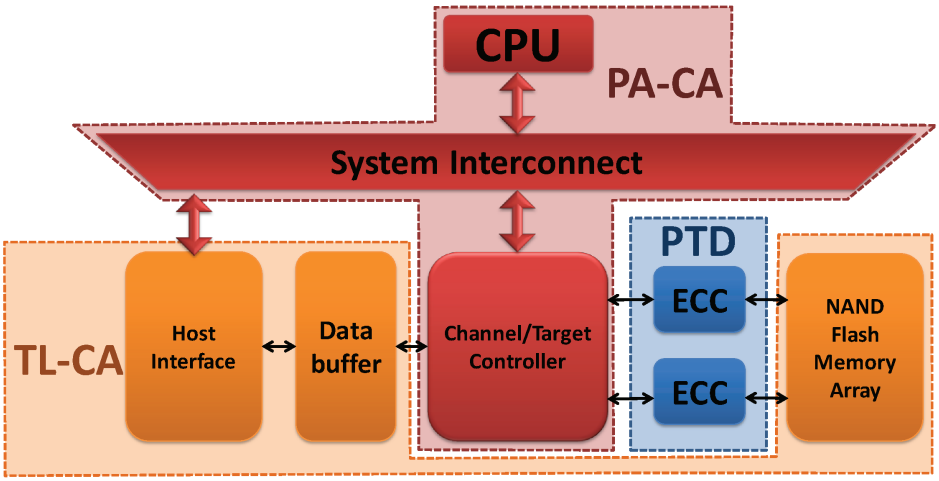


Figure 1.1: Default architecture template modeled by the simulator.

all the timing- and protocol-induced delays are captured accurately for the bus architecture. The components are modeled in a cycle-accurate manner as well: data processing inside a component is scheduled on a cycle by cycle basis. Cycle-accurate models are close in structure to RTL models, but still benefit from the simulation speed-up provided by a high-level modeling and simulation language.

In SSD Explorer, the pin- and cycle-accuracy are enforced for modeling the control path, since subtle effects in command handling and/or component interactions may cause performance deviations that should be highlighted by a FGDSE tool. Any end user may plug its SystemC models here, reflecting in-house IP cores. In the early design phases, however, more relaxed bus functional models can be used, limiting cycle accuracy to the bus interfaces and to the bus architecture itself. This option reduces the simulator capability in capturing FTL execution overhead, yet not limiting the disk performance estimation using either coarse or abstracted FTL models.

1. CPU: SSDExplorer can implement any CPU, including both custom IP cores and advanced multicore architectures given the availability of its PA-CA model, its proper Instruction Set Simulator, or the actual processor command back-trace. With this respect, in fact, thanks to the provided CPU-wrapper users can first test and design with dedicated simulators a proper CPU on which the FTL can be executed, and than replay on SSDExplorer the corresponding set of operations. Thanks to this feature, the programming model offered by the tool is flexible supporting the inter-operability with state-of-the-art processors simulators such as Gem5 [20] and Open Virtual Platforms [21] and enabling future modeling and

analysis activities on the processor role in the SSDs.

2. System Interconnect: SSDExplorer can include the most relevant communication interfaces used in SSD platforms such as: AMBA AHB, Multi-Layer AMBA AHB, AMBA AXI (single address lane, multiple data lanes) and OCP. Custom system interconnects can also be plugged into the simulator provided the availability of their PA-CA models.

3. Channel/Target Controller: to perform read/write operations on the NAND Flash memory arrays, it is mandatory to introduce a controller deputed to formatting commands issued by the CPU with a proper protocol. The Open NAND Flash Interface (ONFI) [22] standard has been exploited for the NAND memory arrays. From an architectural point of view, the channel/target controller is composed of five macro blocks: a slave program port on the system interconnect, a Push-Pull DMA controller, a SRAM cache buffer, an ONFI port and a command translator. The microarchitecture described in [23] has been chosen to mimic realistic functionalities of a channel/target controller in industry-relevant designs. SSDExplorer can be configured with a flexible number of channels and targets.

Transaction-Level Cycle-Accurate models

The host interface, the DRAM buffers and the NAND Flash memory arrays have been described by selectively abstracting the modeling style. The main idea is to avoid modeling all pins of the communication interface between the data path components, as well as the signals that constitute the bus and the external system interface. Communications instead go through channels, on top of which read and write transactions are carried out. At the same time, computation primitives are still scheduled at each clock cycle. This is to allow both sudden command requests concurrently with the execution of another transaction (e.g. erase suspend in NAND Flash memories during garbage collection) and to preserve timing accuracy in the wrappers bridging these models with pin- and cycle-accurate ones. Nevertheless, the burden to preserve this cycle accuracy is not heavy. In fact, there are memory dominated-components on the data path, whose performance mainly depends on properly capturing timing parameters of the memories rather than on the modeling of complex computation tasks. Moreover, since NAND Flash memory components could be inactive for long times, (i.e., for random workloads which do not spread among all the SSD channels), to increase the simulation efficiency the processes used for memory simulation are spawned only

on demand. Basically, when the command scheduler detects that an operation must be issued to one or more NAND Flash targets belonging to a single channel, it spawns the corresponding process responsible for the management of the targets of that channel. If the channel is idle, no process is created. Upon process spawning, only the finite state machines of the addressed targets are updated, while the other ones remain idle. By combining the dynamic process management with the selective process spawning, it is possible mitigating the impact of the memory subsystem on the simulation speed.

1. Host Interface:this component manages the communication protocol with the host, providing commands and data to the SSD. Two types of interfaces are implemented in SSDExplorer: Serial Advanced Technology Attachment (SATA) and PCI Express (PCIE). Both interfaces include a command/data trace player, which parses a file containing the operations to be performed and triggers operations for the other SSD components accordingly. The features of the available interfaces are:

   - SATA: all SATA protocol layers [24] and operation timings have been accurately modeled following the SATA protocol timing directives provided in [25]. Native Command Queuing support has been implemented featuring arbitrary queue length up to 32 commands.
   - PCIE: this interface allows boosting sequential and random operations throughput, and it is currently exploited in enterprise SSDs [1]. Fast operations are achieved through the NVMe (Non Volatile Memory Express [26]) protocol that significantly reduces packetization latencies with respect to standard SATA interfaces [27]. All PCIE configurations (i.e., from generation 1 up to generation 3 with variable lane numbers) can be modeled, thus ensuring accurate latency matching.

   To ease the interchange between different host interfaces, a common control architecture based on a fabric interconnect slave port and an external DMA controller [28] able to transfer data from the host interface to the data buffers and vice versa is available in SSDExplorer.

2. DRAM Buffer: this component is used either as a temporary storage buffer for read/write data or as a buffer [29] for the address mapping operations given a dedicated firmware that runs on SSDExplorer. A cycle accurate DRAM model is required to capture realistic behaviors (i.e., column pre-charging, refresh operations, detailed command timings, etc.). The data buffers of SSDExplorer are modeled with a SystemC customized

version of the simulator proposed in [30]. The number of available buffers in a SSD architecture is upper bounded by the number of channels served by the disk controller. In SSDExplorer the user can freely change this number, as well as the bandwidth of the memory interface, the DRAM functionality, etc., acting upon a simple text configuration file, which abstracts internal modeling details. The DDR, DDR2 and DDR3 protocols are supported as DRAM interface.

3. NAND flash memory array: the fundamental component of a SSD is the non-volatile storage memory array. NAND Flash devices are hierarchically organized in dies, planes, blocks, and pages. Program and read operations work on a page basis, whereas the erase operation is performed blockwise, thus inhibiting the in-place data update. Due to the internal architecture of NAND Flash devices, large fluctuations in memory timings arise depending on the chosen operation, thus introducing a significant amount of performance variability. To accurately take into account all these effects, a cycle accurate NAND Flash simulator has been exploited [31]. Furthermore, to take into account the realistic behavior of the memory, an error injection engine (i.e., a BER simulator) has been included to reflect the effects of different error patterns on the other components of the NAND Flash/ECC subsystems. It is possible to embody different NAND Flash technologies (i.e., single-, multi- and triple-level cell) in SSDExplorer.

Parametric Time Delay model

The microarchitectural blocks related to the Error Correction Codes (ECC) have been modeled using a parametric time delay abstraction level. These devices, on the one hand, strictly depend on the design choices of SSD vendors, on the other hand, their behavior and impact on SSD I/O performance can be easily abstracted by means of well-defined quality metrics. [32]. In other words, the behavior inside PTD models does not need to be scheduled at every cycle boundary (i.e., to be cycle accurate). Instead, computation primitives inside a component can be grouped together and the schedule can be relaxed so that time is incremented in chunks. As an example, the correction time of 5 errors in a NAND Flash page corresponds to an effective wait time of 10 ns that cannot be interrupted by any other command. This allows a reduction in the detail captured inside components, with benefits on both modeling time and simulation speed. At the same time, communication events can still be scheduled in a cycle accurate manner. However, even if this choice enables accurate I/O performance characterization, it prevents

functional simulation when such components are instantiated. Nevertheless, at an early design stage, when the internal SSD architecture is defined, functional simulation is actually not required, since priority is given to the delivery of a target I/O performance with a matched and resource-aware architecture configuration. At later design stages, PTD models can be replaced by more refined models, even restoring the functional simulation capability of SSDExplorer.

SSDExplorer embodies two configurable PTD ECC models: a Bose, Chaudhuri, Hocquenghem (BCH) engine and a Low Density Parity Check (LDPC) ECC. These blocks are composed by a fixed high-speed encoder and a multi-machine decoder. The delays of both the encoder and the decoder stages can be configured to mimic the behavior of a single- or a multi-threaded version as shown in [32]. To explore different ECC architectures, the internal parallelism of each machine can be configured by the user.

Finally, it is worth pointing out that in state-of-the-art SSD simulators the presence of ECC is usually neglected. However, an accurate calculation of the SSD performance must take into account the latency introduced by the encoding and decoding phases of an ECC especially when performance-reliability trade-offs related to NAND Flash memories must be analyzed [33].

1.3 FTL simulation

1.3.1 Realistic FTL

Realistic FTL development, testing and simulation are one of the main features that users require during the early steps of the firmware developing phase. These features, in fact, allow understanding how management algorithms such as Garbage Collection (GC) and Wear Leveling (WL) have to be designed to maximize both the performance and the reliability of NAND flash memories. However, since many different implementations could be available, it is mandatory to offer a flexible framework for both the processor simulation/design and the FTL execution. In SSDExplorer these problems have been addressed exploiting two well established open-source simulators: Gem5 [20] and Open-Virtual-Platform (OVP)[21]. These tools allow users to define a custom system on chip architecture which can embody simple single core systems or more complex multi/many core platforms. Moreover, thanks to the standard programming model provided by these simulators, specific FTL implementations can be easily developed, tested, and simulated thus capturing the actual behavior of the SSD's firmwares on top of a specific processor architecture.

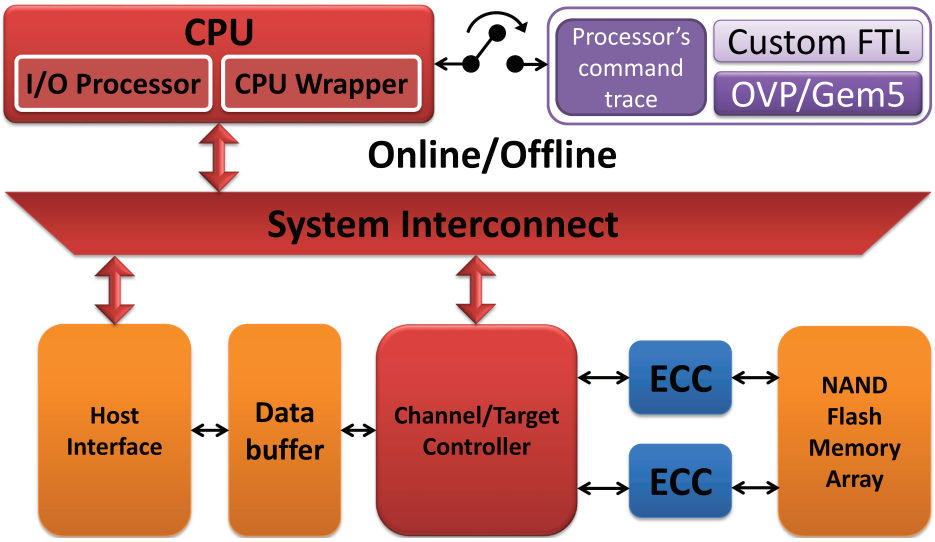


Figure 1.2: "Online-Offline" simulation mode when a realistic FTL is considered.

All the aforementioned features, however, do not come for free. In fact, concerning the simulation speed of Gem5 and OVP it is necessary to highlight that simulating complex architecture such as nowadays SSD processors, could take a lot of time. As a consequence, embodying these platforms inside SSDExplorer does not represents the best solution when a fast performance assessment is required. To overcome this problem, it has been developed a specific wrapper (sketched in Fig. 1.2) inside the CPU model of SSDExplorer which can be dynamically attached/detached to the processor's simulator. This approach is called "Online-Offline" simulation mode and relives SSDExplorer to embody complex processor models which could burden on its actual simulation speed. Basically, the processor's simulator produce separately from the SSDExplorer framework a specific command trace during the FTL execution ("Offline" simulation). This command trace is composed by two parts: *i)* the list of read, write, and erase operations that have to be executed by NAND flash memories representing the actual FTL status; *ii)* the processing time taken by the processor to produce each I/O. All the collected data are then read and playback by the SSDExplorer CPU wrapper ("Online" simulation) which introduces the right amount of delay and send the I/Os to the I/O processor which is connected to the Channel/Target controller and the underlying memory array. This approach allows simulating a infinite number of possible FTL algorithms since, SSDExplorer behaves as a delay generator which is completely agnostic with respect the actual FTL state or configuration. Moreover, thanks the "Online-Offline" simulation paradigm

users can assess the performance of the SSD only when needed or when a specific FTL state has to be studied.

1.3.2 WAF model

The FGDSE of a wide range of SSD architectures has the drawback of requiring a custom FTL tailored for each configuration (some examples of custom FTLs are provided in [34, 35, 36, 37, 38]). Moreover, during the early steps of the SSD design, a complete FTL implementation could not be available since many architectural details such as the processor's architecture could not be provided. This calls for an estimation of the impact of software management algorithms usually exploited by a SSD (e.g., GC, WL, etc.) without requiring their actual implementation. This problem has been tackled in [6] by introducing a lightweight algorithm able to evaluate the blocking time of the GC impact in terms of WAF, under the assumption that others FTL functionalities are handled by the CPU without imposing a significant performance drop. A standard WAF model [6] is able to calculate the number of additional writes produced by the GC operation with respect to the actual number of writes issued by a host system starting from a pool of few parameters such as: the total number of blocks in the disk, the over-provisioning factor and the GC activation threshold. In this way, it is possible to quickly explore the SSD FTL behavior and to assess its efficiency through the computed WAF value. In fact, as WAF increases, the computed blocking time of the modeled GC management procedures increases as well, thus heavily affecting the overall disk performance. In SSDExplorer, thanks to the CPU wrapper and the "Online-Offline" simulation mode described in Section 1.3.1, it is possible to inject inside the simulator also a command trace produced by a WAF model. Clearly, as it can be seen in Fig. 1.3, since this algorithm is not intended to be also a processor simulator, the processor's command trace will include only read, write and erase operations for NAND flash memories, whereas the actual processing time of each command will be neglected.

1.3.3 WAF model VS realistic FTL

The effectiveness of the WAF model must be compared with a real page-level FTL to locate the operative range in which it can be reliably adopted. To this purpose the template architecture of Fig. 1.4 has been used, configured with the values reported in Table. 1.2.

The two main actors are the SSD controller (hereafter intended to consider the host interface, the CPU, the interconnect system and the channel/target controller) whose key parameter is the CPU frequency, and the NAND

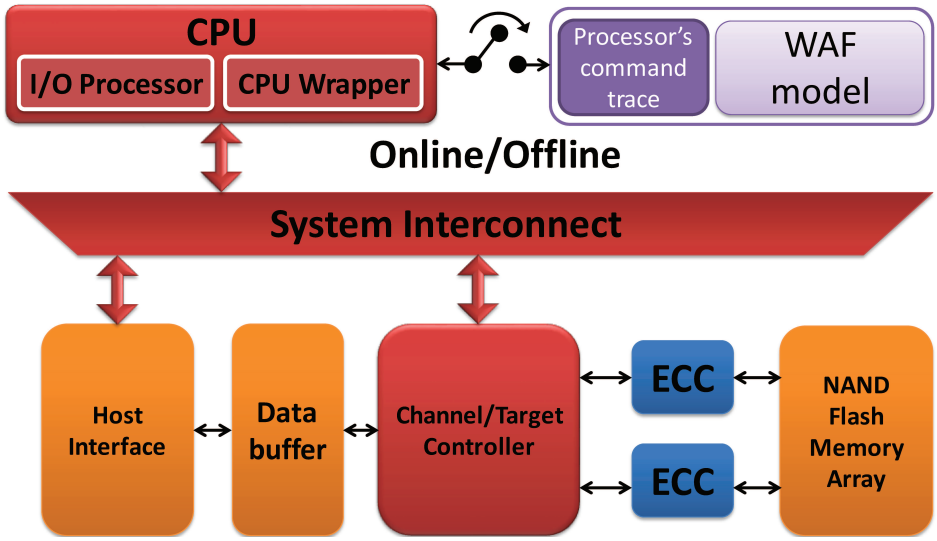


Figure 1.3: "Online-Offline" simulation mode when a WAF model is considered.

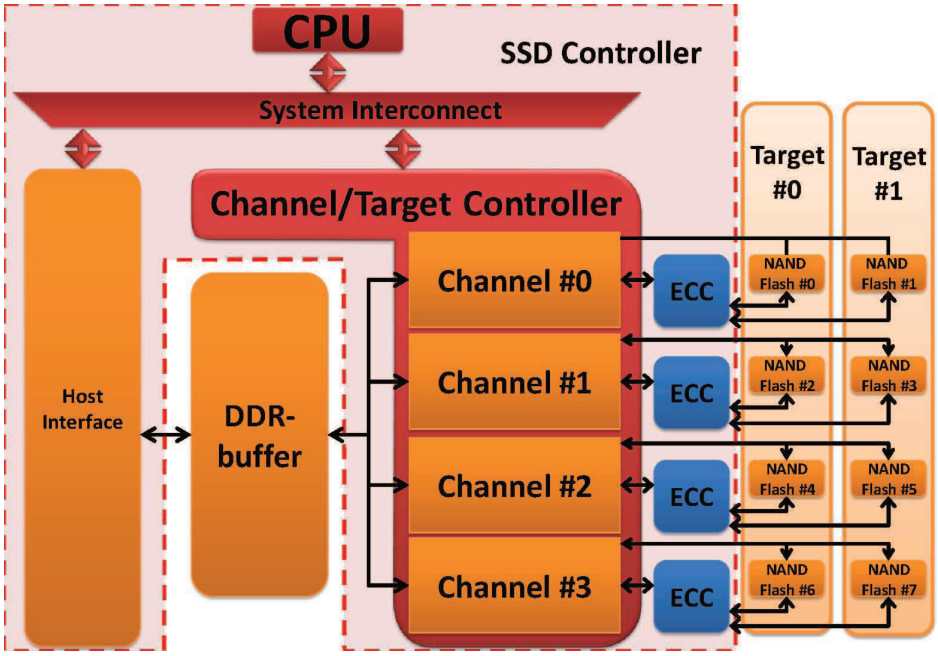


Figure 1.4: A SSD template architecture used for FTL and WAF algorithms evaluations

memory subsystem. Since each die and plane of a NAND Flash memory is composed of a repetitive cluster of pages and blocks, it is possible to shrink

Table 1.2: SSDExplorer configuration for WAF accuracy assessment.

Parameter	Architecture
Host Interface	SATA II
DRAM-Buffer	1
Mock-up DRAM-size	64 kBytes
Channels	4
Targets	2
NAND Flash Dies	4
NAND Flash Planes	2
NAND Flash Blocks	16
NAND Flash Pages	4
NAND Flash Page Size	4096 Bytes
Caching	No
FTL-LOG2PHY mapping	page-associative
FTL-GC algorithm	Greedy
FTL-GC threshold	30%
FTL-GC reserved blocks	1
FTL-WL policy	opportunistic
Over-provisioning	20%

the actual memory size by reducing the effective cluster capacity. In such a way longer processes like a full disk fill can be easily simulated without impacting the framework accuracy. The only constraint related to this mocking up approach is to maintain the memory architecture in terms of dies and planes because these parameters heavily modify the memory performance.

Both the FTL and its WAF model have been tested among different SSD controller frequencies dealing with different workloads. Fig. 1.5 shows the performance achieved with a read workload: since read transactions do not require software manipulations, the FTL execution time becomes marginal, therefore not affecting the overall bandwidth. When a write workload is considered, Fig. 1.6 shows a discrepancy between the WAF and the real FTL implementation which vanishes as the SSD controller frequency increases. The discrepancy at low frequency is mainly caused by two factors: the first is the difference in the WAF values computed by the model and by the FTL, which are 1.20 and 1.16, respectively; the second is strictly related to a non-negligible additional execution time spent to execute the GC victim-blocks identification process of the firmware. By increasing the SSD controller speed, the FTL components contribution not related to the GC on the SSD perfor-

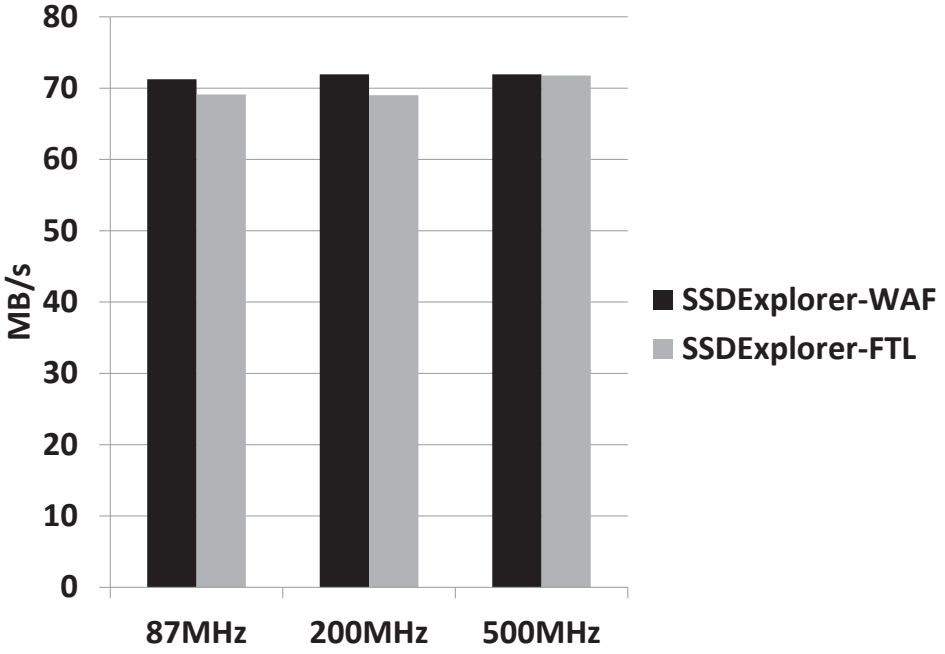


Figure 1.5: Sequential read bandwidth achieved by SSDExplorer using the WAF abstraction model and a real FTL.

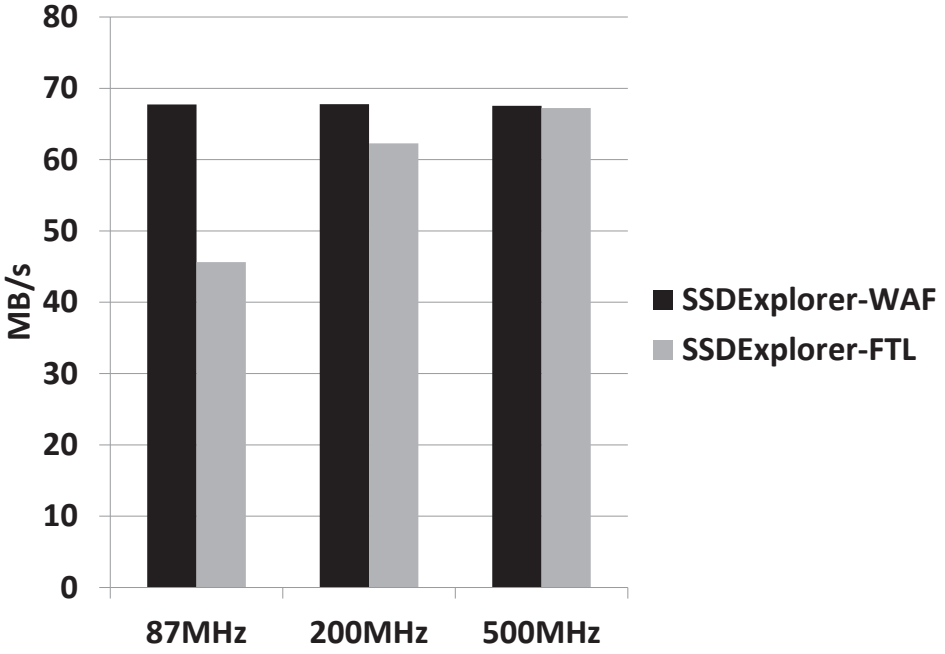


Figure 1.6: Sequential write bandwidth achieved by SSDExplorer using the WAF abstraction model and a real FTL.

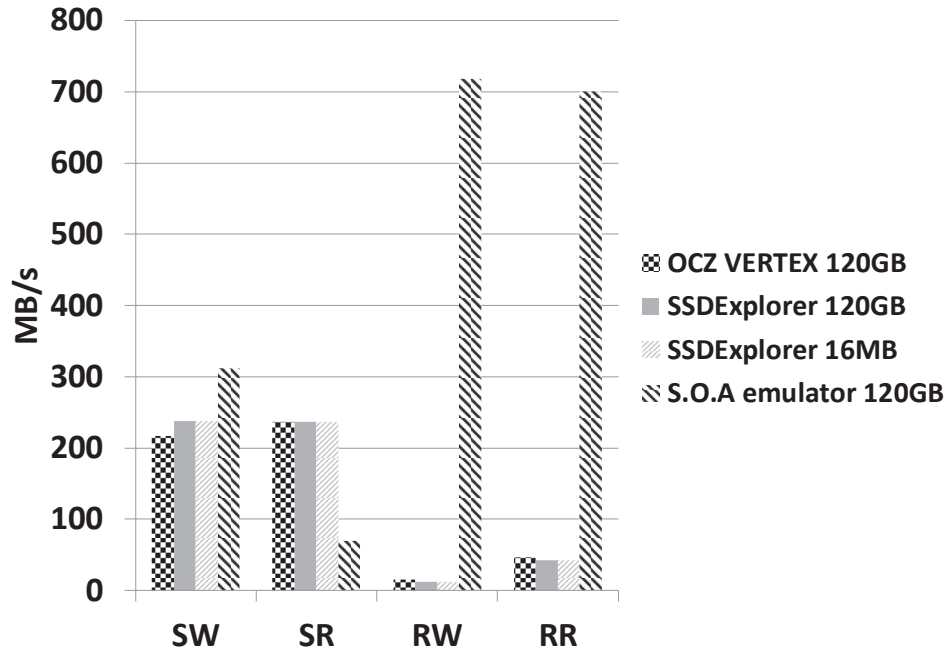


Figure 1.7: Performance comparison between OCZ Vertex 120GB, SSDExplorer (both mock-up and full disk simulation), and a S.O.A emulator tool [3] in terms of throughput for Sequential Write (SW), Sequential Read (SR), Random Write (RW) and Random Read (RR).

mance become marginal for the overall bandwidth estimation process because the maximum achievable performance is mainly dominated by NAND Flash memories timings (i.e., the CPU is able to dispatch all the FTL functionalities without a blocking time).

Starting from the above results, the WAF model can be exploited with a marginal performance misalignment given the constraint of a relatively high SSD controller frequency. However, since state-of-the-art SSD controller frequencies are in the 300 - 600 MHz range [39], a WAF abstraction of the FTL represents a good simulation speed/accuracy trade-off.

1.4 Performance comparison with real SSD platforms

1.4.1 Consumer Device

In order to assess the accuracy achieved by SSDExplorer, a direct comparison between the proposed framework and an OCZ Vertex 120GB [7], a widely

adopted device, has been carried out. This device has been chosen to speed up the validation phase, since it is based on a well-known and documented controller [40], running at 166 MHz, that can be easily simulated. The validation methodology followed in this section makes use of standard synthetic benchmarks to quantify the I/O performance of SSD devices [41]: a sequential and a random 100% write and 100% read workloads with a block size of 4kB are injected inside the simulated disk. The choice of using synthetic benchmarks rather than realistic ones [42] is justified by the fact that the latter approach would introduce a complication in the validation process of the results since the implications behind their use would require a tailored FTL matched with the architecture that the simulator is able to characterize. Moreover, a realistic benchmark could hinder the behavior of a SSD since the chosen workload may put it in a favorable working point, thus neglecting worst case conditions. In all the following analysis, it has been used the WAF abstraction model both simulating a mock-up version of the disk (i.e., a 16 MB SSD) and a full disk with addressable space equal to those of the OCZ device (i.e., 120 GB SSD).

As shown in Fig. 1.7, for a sequential workload, SSDExplorer matches the OCZ device performance with an error margin of about 8% in the write operation and 0.1% for the read operation. When a random workload is used, the performance deviation from the OCZ disk amounts to 6% and 2% for writes and reads, respectively. These deviations are due to the lack of any information about the write caching algorithm in the WAF model [6]. By looking at the OCZ Vertex reference manual [7] it can be found that caching is massively adopted to reduce the amount of write operations redirected to the non-volatile memory subsystem and hence simulated write operations (both sequential and random) show offsets higher than read operations. In light of this consideration, the results reported in Fig. 1.7 confirm the accuracy provided by SSDExplorer. This is even more relevant if it is considered that these low error margins can be achieved avoiding the real FTL implementation.

To ultimately prove the accuracy of the proposed framework it has been performed the same comparison also with a state of the art emulation tool (S.O.A emulator) [3]. Since this tool embodies a fully reconfigurable FTL, to achieve a fair comparison its parameters have been configured to provide the same WAF value used in SSDExplorer. As it can be seen in Fig. 1.7, the performance mismatch between the OCZ Vertex and the S.O.A emulator is 30% and 70% for sequential write and sequential read, respectively, whereas for random workloads the S.O.A. emulator results are completely out of scale. The roots of these discrepancies reside in the inability of the S.O.A emulator to accurately model the host interface command queuing, the multi-channel

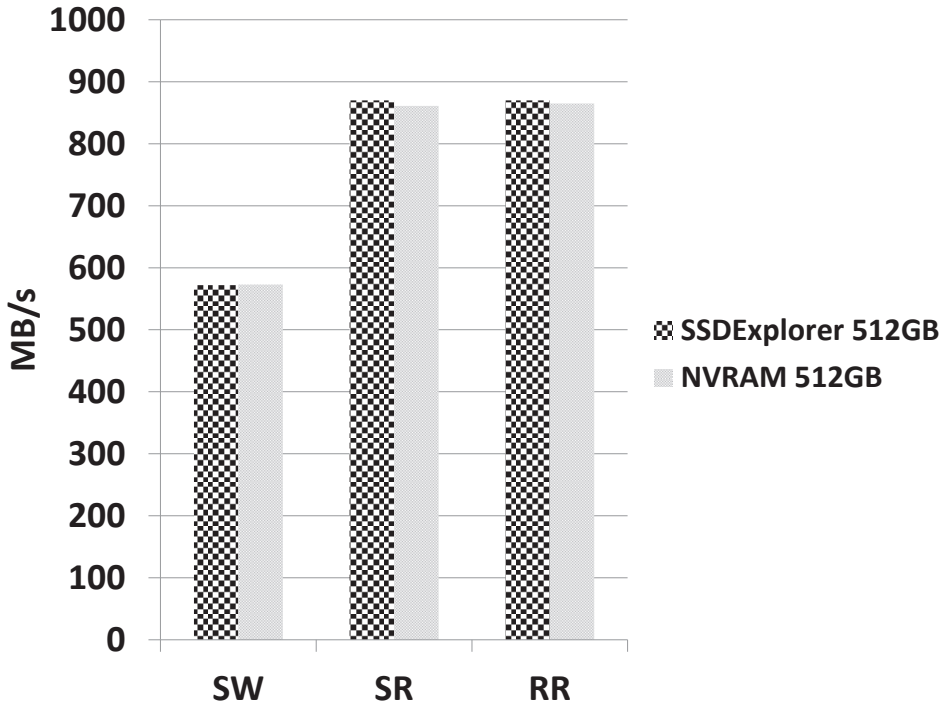


Figure 1.8: Performance comparison between a 512GB NVRAM card and SSDExplorer (full disk simulation), in terms of throughput for Sequential Write (SW), Sequential Read (SR), and Random Read (RR).

interleaving, and the ECC behavior. Therefore, S.O.A. emulators cannot be used for FGDSE, whose paradigm is aimed at accurately describing all the components belonging to a SSD in order to get accurate performance breakdown curves. It is interesting to point out that the mock-up SSD simulation results equals the results of a full disk simulation, thus validating the approach proposed in Section 1.3.2.

1.4.2 Enterprise Device

In this paragraph SSDExplorer has been configured to track the behavior of a 512 GB NVRAM card currently exploited in cloud applications [8]. For this comparison it has been took as reference a disk architecture featuring a 8 channels/4 targets controller, a single 2GB DDR3 (1333MT/s) DRAM buffer and a PCIE Gen 2 x8 lanes host interface. Simulated NAND Flash memories have been configured as 2X enterprise-MLC devices with a page program time $t\_PROG = 1.8$ ms, a page read time $t\_READ = 115$ μs and a block erase time $t\_BERS = 6$ ms. The main aim of this comparison is to demonstrate the flexibility and the accuracy of the proposed framework in

simulating the interaction between the memory subsystem (NAND Flash and DRAM buffer) and the host interface even when complex architectures are explored. Finally, all tests have been pursued with sequential write, sequential read, and random read workloads. The specifications for performing the random write test were not available by the manufacturer. As shown in Fig.1.8 the SSDExplorer capabilities in simulating large state-of-the-art SSDs are demonstrated by the obtained extremely low performance mismatch: about 0.01% for both read and write operations.

1.5 Simulation Speed

SSDExplorer is totally written in SystemC, then its capability to be accurate is traded with simulation time. Since SSDExplorer includes PA-CA and TL-CA models, the number of kilo-Cycles per Second (kCPS) represents the only metric to be adopted to evaluate speed features, whereas the performance of emulation/simulation tools, mainly based on behavioral models, are measured in elapsed CPU time, thus making impossible any direct comparison. Fig. 1.9 shows the kCPS achieved by SSDExplorer for 9 different SSD architectures (see Table. 1.3 for details) on an Intel Xeon CPU E5520 clocked at 2.27GHz with 12GB of RAM, which runs a Redhat x86-64 Linux operating system. The considered workload is a sequential 4kB write that distributes among all the simulated NAND Flash targets, and the FTL abstraction through the WAF model is exploited.

Table 1.3: SSD configurations exploited to evaluate the simulation speed.

Configuration	**SSD architecture**
C1	1-CHN;1-TARGET
C2	2-CHN;1-TARGET
C3	4-CHN;1-TARGET
C4	8-CHN;1-TARGET
C5	16-CHN;1-TARGET
C6	32-CHN;1-TARGET
C7	1-CHN;2-TARGET
C8	1-CHN;4-TARGET
C9	1-CHN;8-TARGET

In Fig. 1.9, the first set of results (configuration C1 - C6) shows the simulation speed dependency on the number of instantiated channels, whereas the

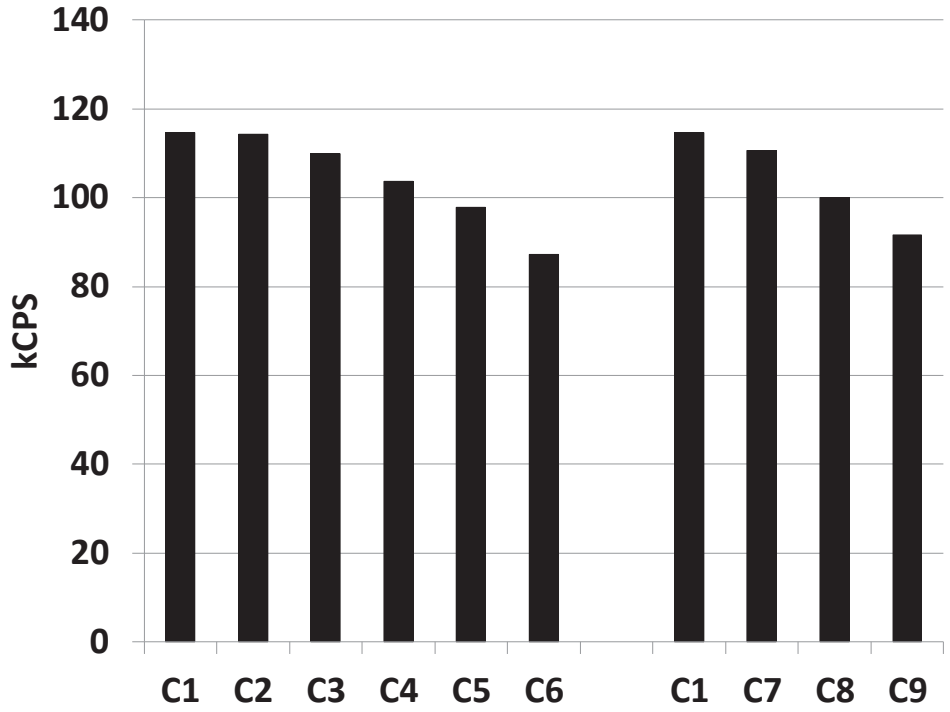


Figure 1.9: SSDExplorer simulation speed with different SSD configurations using the WAF abstraction model.

second set of results (C1, C7 - C9) shows the dependency on the number of NAND Flash targets. If the simulations are performed with the same configurations, but using a real FTL instead of its abstraction, the SSDExplorer's simulation speed drops averagely by a factor three.

It is worth to point out that, even for resource-hungry configurations, the simulation speed is in the order of 100 kCPS which is an optimal reference value for PC-CA EDA tools [43].

Chapter 2

Design trade-offs for NAND flash-based SSDs performance

During the design phase of an SSD, it is mandatory to keep in mind which is the target application on which disk will be used. In fact, depending on which feature has to be guaranteed such as, performance, latency or reliability, several aspects related to the SSD's architecture must be wisely tuned. As a consequence, since optimizing one or all of these metrics could heavily modify the final architecture and cost of the disk, it is mandatory to perform a design space exploration locating possible bottleneck or resource misallocation before the final tapeout. In this chapter it is shown which are the main actors of the disk's architecture able to modify its behavior in terms of bandwidth and latency. Moreover, it is demonstrated that exploring these features only at the beginning of life of the SSD is not sufficient to give a clear insight on how the disk will behave under different working conditions or ware-out states.

2.1 Design for maximum performance

In this Section it is shown an example of how SSDExplorer can be used to find the optimal SSD design point for a given target performance. The goal is to achieve the minimum resources allocation, given the host interface bandwidth constraint. Table 2.1 shows a set of representative design points used to this purpose. All simulations are performed using a synthetic workload composed of a sequential write trace whose payload is fixed to 4kB. Moreover, all data have been collected using two different DRAM buffer management policies typically exploited in consumer and enterprise environments [44]: *write back and write through caching* and *no caching* [29]. For the former, the SSD

Table 2.1: SSD configurations for optimal design point exploration.

Configuration	SSD architecture
C1	4-DDR-buf;4-CHN;4-TARGET;2-DIE
C2	8-DDR-buf;8-CHN;4-TARGET;2-DIE
C3	8-DDR-buf;8-CHN;8-TARGET;2-DIE
C4	8-DDR-buf;8-CHN;8-TARGET;4-DIE
C5	8-DDR-buf;8-CHN;8-TARGET;8-DIE
C6	16-DDR-buf;16-CHN;8-TARGET;4-DIE
C7	16-DDR-buf;16-CHN;4-TARGET;2-DIE
C8	32-DDR-buf;32-CHN;4-TARGET;2-DIE
C9	32-DDR-buf;32-CHN;1-TARGET;1-DIE
C10	32-DDR-buf;32-CHN;8-TARGET;4-DIE

controller notifies the end of each transaction to the host system when data have been moved from the host interface to the DRAM buffers. For the latter policy, the notification is triggered only when all data have been actually written to the NAND flash memory. All experimental results consider a 4X Multi-Level Cell NAND flash technology whose main characteristics are t_PROG which ranges from 900 μs to 3 ms, $t\_READ = 60$ μs and t_BERS which ranges form 1 ms to 10 ms [45].

Fig. 2.1 shows how different architectures exploit the performance of a SATA II host interface. The *SATA ideal* curve refers to the theoretical throughput achievable only by the host interface. Instead, the *SATA+DDR* curve shows a more realistic metric for the host interface performance since it incorporates the time spent by its internal DMA engines to transfer data from the host system to the DRAM buffers (i.e., the time to process the transactions from the host). Starting from this consideration, the best design point is the one that tries to achieve the *SATA+DDR* bandwidth by maximizing the bandwidth of the *DDR+FLASH* curve (i.e., the time spent by the flash memory to flush the DRAM buffer and write the data). The *SSD cache/ SSD no cache* curves represent the bandwidth of the entire disk that considers the bandwidth of the *DDR+FLASH* contribution and the potential saturation effect of the host interface indicated by *SATA ideal* (i.e., considering the overall disk performance dependently on the adopted caching strategy).

When caching is used, the *SSD cache* bars in Fig. 2.1, indicates *C6*, *C8* and *C10* as the best candidates since they reach the target performance and saturate the host interface bandwidth. However, when the resource cost

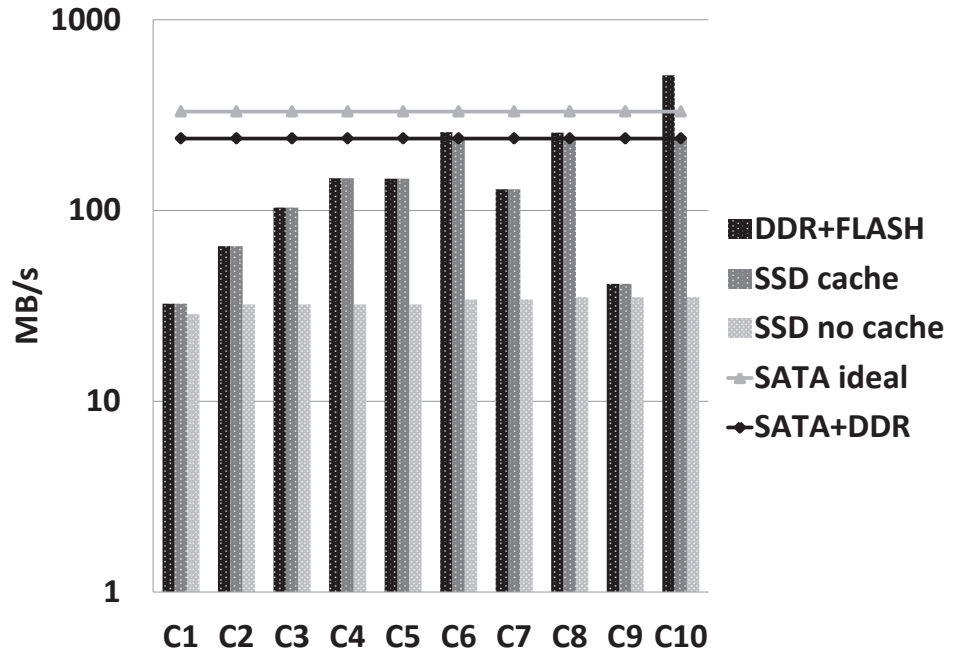


Figure 2.1: Sequential Write: SATA II host interface. Comparison of the configurations proposed in Table. 2.1 for the optimal design point exploration.

constraints are taken into account, it is clear that *C6* represents the right choice since it is the configuration able to reach the host interface limit with the lowest resources consumption. On the contrary, when *no caching* is used, the overall disk performance (the *SSD no cache* contribution) is strongly limited. In this scenario there is no configuration able to reach the target performance and so, the search for the optimal design point falls on *C1*.

The reason behind the performance flattening with *no caching* lies on the SATA interface and, in particular, into its limited command queue depth. In fact, the SATA protocol is able to manage only a maximum of 32 commands at once, thus in a SSD exploiting a *no caching* policy, the host interface cannot acquire new commands until the current ones have been executed by the NAND flash memories. This implies that, the internal parallelism provided by the device cannot be exploited, which becomes clear when checking the SSD performance indicated in the *DDR+FLASH* results.

To overcome this limitation and unveil the performance provided by highly parallel SSD configurations, Fig. 2.2 shows the results achieved when a PCIE-Gen2 featuring 8 lanes and the NVMe protocol is exploited. Due to the high PCIE speed, the host interface will no longer represent a SSD performance bottleneck. In fact, even the most parallel configuration (i.e.,

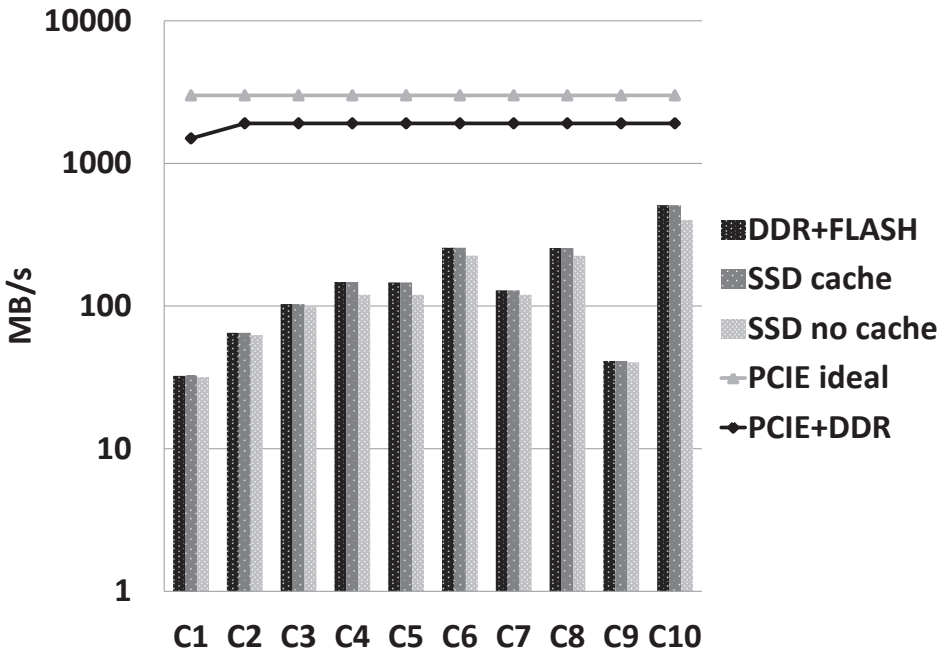


Figure 2.2: Sequential Write: PCIE host interface. Comparison of the configurations proposed in Table.2.1 for the optimal design point exploration.

C10) is not able to saturate the interface bandwidth. However, the major result shown in Fig. 2.2 can be evidenced by looking at the *SSD no caching* contributions. In this case, since the NVMe protocol can handle up to 64k-commands, the SSD internal parallelism can be reached and fully exploited. However, a performance gap between these configurations still exists. Indeed, the time spent to flush the incoming data to the NAND flash memories for SSD cached architectures is hidden. It is worth to point out that, when a NVMe protocol with a PCIE interface is exploited, since there are no intrinsic architectural limitations, the search for the most efficient design point is driven by the hardware costs. If maximum performance is the main driver during the design phase of the SSD, *C10* is the best solution. On the other hand, if the performance-cost trade-off is leveraged, solutions ranging from *C3* to *C8* are eligible.

2.2 Design for minimum latency

As presented in Section 2.1, the main parameters able to increase the SSD bandwidth are the internal controller parallelism (i.e., the number of memory channels and NAND flash targets), and the host interface Queue Depth (QD).

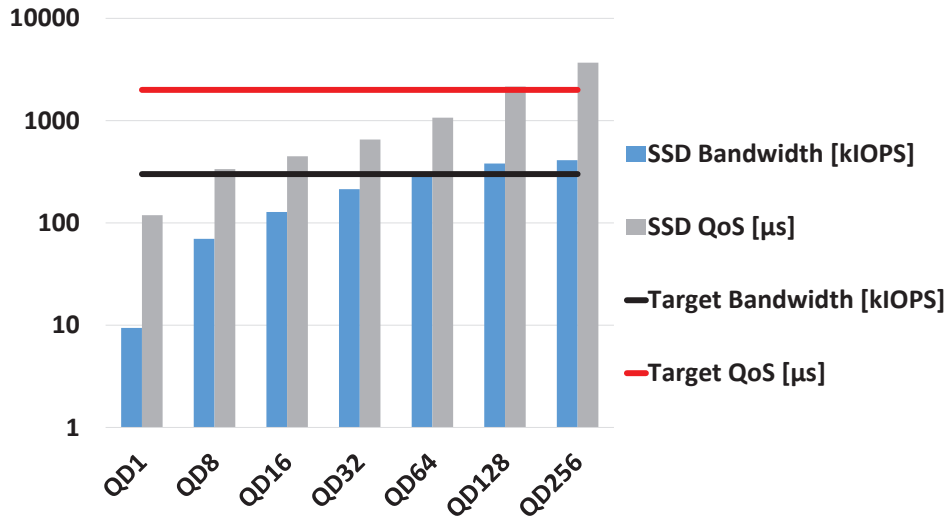


Figure 2.3: SSD bandwidth and QoS as a function of the host queue depth (QD). The target performance and QoS are marked for all the tested cases.

This latter, however, even if it allows exploiting all the architectural resources made available by the disk, severely impact an important figure of merit of an SSD: the Quality of Service (Qos) [46]. In an SSD, the QoS is calculated as the 99.99 percentile of the SSD's latency distribution and it is used to qualify how the SSD behaves in the worst-case corner conditions. Thanks to this metric it is possible to understand if the target disk architecture is suitable for a specific use case such as real time systems or safety-critical environments. Fig. 2.3 shows how the disk bandwidth and the QoS scale when the host QD is varied from one to 256 commands. In this case it has been considered an SSD composed by 8 channels with 8 mid-1X TLC NAND flash targets, with an average read time of 86 μs, each. The used workload is a 100% 4 kB random read traffic. As it can be seen, it is possible locating a performance/latency trade-off by which the higher is the host interface queue depth the higher is the SSD QoS. This behavior, however, is in contrast with the requirements of high performing SSDs by whom it would be appropriate to achieve the target bandwidth with the lowest QoS. In fact, as aforementioned, nowadays user applications such as financial transactions or e-commerce platforms [47], are designed to work with storage devices which have to serve an I/O operation within a specific time-frame which is usually upper-bounded by the QoS requirement.

In order to deal with this performance/latecy trade-off it is usual to design the whole SSD architecture for the target performance and QoS, forcing the host interface to work only with a specific QD. In such a way the design

space of the SSD controller is reduced and it is possible to tune its internal parameters to achieve the application requirements. As a matter of example, if a 300 kIOPS bandwidth and a 2 ms QoS are required (solid lines in Fig. 2.3), the configurations shown in Fig. 2.3 indicate that a QD of 64 commands has to be used by the host system. All other configurations are not suitable for the target requirements since they are able to meet either the bandwidth or the QoS.

2.2.1 The Head-of-Line (HoL) blocking effect

One of the main limitations of the design methodology presented in the previous Section, is the lack in the intrinsic flexibility of the QD usually required by the host system. In fact, during the design phase of an SSD, it is usual to define only the target performance requirements because it is not know *a priori* on which application the disk will be used. As a matter of example, SSDs used in flexible I/O environments where users can instantiate complete virtual platforms in a "plug-and-play" fashion [48], have to guarantee the target performance and QoS for a general purpose traffic. As a consequence, since in these systems the actual host QD is strictly dependent upon the number of parallel threads on which the target application is running, the number of commands that will be issued by the user and the corresponding QD, cannot be forecasted. Therefore, it is clear that to meet the general purpose workload specifications of these environment, the SSD has to be designed to provide the target performance and QoS independently from the host QD.

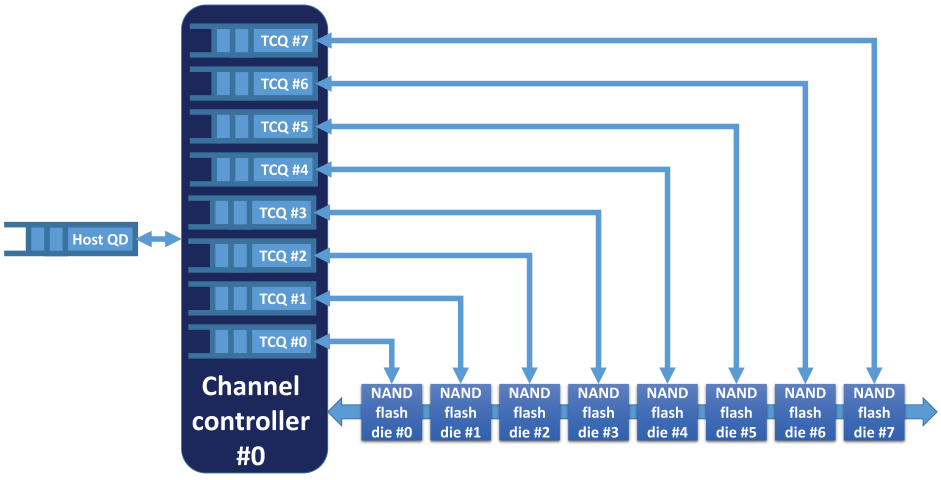


Figure 2.4: Queueing hierarchy implemented inside the SSD controller

To deal with this requirement it is possible to exploit a phenomenon called Head-of-Line (HoL) blocking whose effect is to limit the number of outstanding commands inside the SSD. To clarify this point it is useful to study how host commands are manged and queued by the SSD controller.

Fig. 2.4 shows the queueing hierarchy usually implemented in traditional SSD controllers. As it can be seen, beside the external host QD, it is usual to embed on the SSD channel a small command queue for each NAND flash memory die: the Target Command Queue (TCQ). Basically, the TCQ allows the host to continue to issue commands to NAND flash targets even when it tries to read or program a chip which is already busy on another operation. In fact, when this condition is verified, the operation is simply enqueued on the specific TCQ and the SSD controller can continue to fetch and send other commands from the host QD to the target NAND flash memory. This approach, on the first hand allows maximizing the bandwidth of the SSD since TCQs are able to keep busy all the NAND flash memories at any time, on the other hand, however, it increases the number of outstanding commands inside the SSD since several operations have to wait inside the TCQs before being served. At this point it is clear that, when a lot of commands are issued inside the disk (i.e., when a big host QD is used), the SSD controller and all the related TCQs fall in a deep saturation state which leads to an increased QoS.

When the HoL blocking is used, the aforementioned latency problem can be partially solved. In fact, in this case when the number of commands enqueued in single a TCQ exceeds a predefined threshold, it is possible to trigger a blocking state inside the SSD controller which stall's the submission of a new command inside the host QD. In such a way, depending on the HoL threshold value, it is possible to avoid long command queues inside the TCQs, and hence, the disk's QoS can be kept within a specific window.

Fig. 2.5 shows the effectiveness of the HoL blocking effect in reducing the QoS of the target SSD architecture when different host QD are considered. The same SSD configuration and workload used in the previous Section have been exploited. As it can be seen, unlike in the case presented in Fig. 2.3, as soon as the target performance of 300 kIOPS is reached (QD64 configuration), the HoL blocking effect starts to keep the QoS below the target requirements even when high QDs, such as QD128 and QD256, are considered.

The fine-grained QoS calibration made available by the HoL blocking effect, however, does not come for free. In fact, as it can be seen in Fig. 2.6, if besides the bandwidth and the disk's QoS also the average SSD latency is considered, it is clear that the HoL Blocking effect has to be wisely used. As a matter of fact, it is shown that when the HoL blocking system starts to work, it trades the QoS reduction with an increase of SSD's average latency.

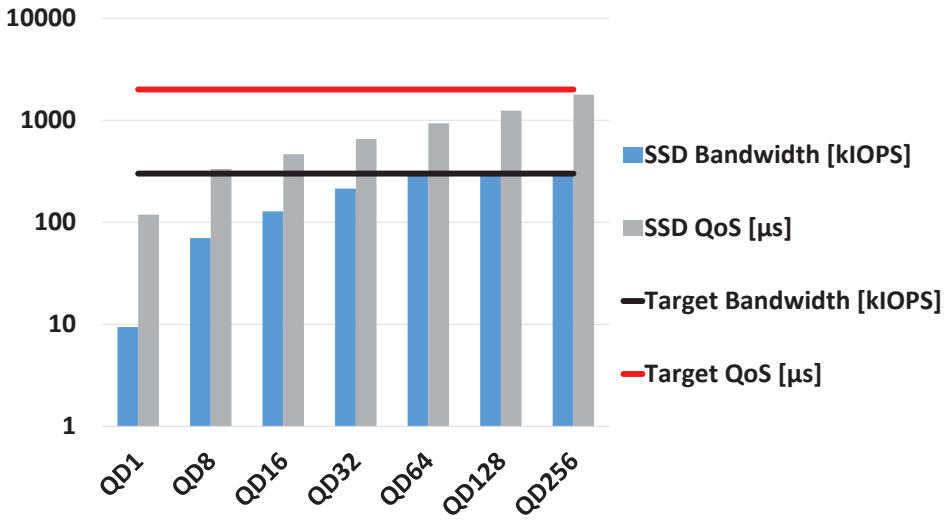


Figure 2.5: SSD bandwidth and QoS as a function of the host queue depth (QD) when the HoL blocking effect is used. The target performance and QoS are marked for all the tested cases.

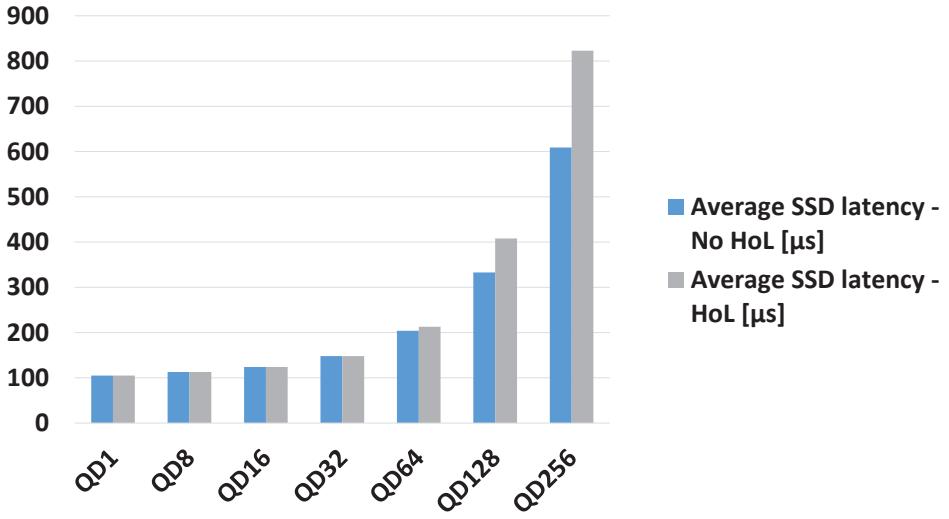


Figure 2.6: Average SSD latency evaluated with and without the HoL blocking effect.

Moreover, this behavior becomes more pronounced when high QD are used (QD64, QD128, and QD256) that is when a higher QoS reduction is required.

2.3 Performance/Reliability trade-off

Achieving high performance (i.e., high bandwidth and low latency) with the lowest resource allocation is one of the main goal of nowadays SSD architectures. However, even if high read/write bandwidth can be easily achieved by such storage devices, one main SSDs problems which limits their widespread adoption is their reliability, which is dependent upon the non-volatile NAND flash memories used as the storage medium. These components are subject to a progressive wear-out whose physical roots reside in the tunnel oxide degradation related to the Fowler-Nordheim tunneling exploited for their program/erase. Such mechanism leads to a variation of the Floating Gate charge which translates to a threshold voltage shift [1]. A direct indication of this phenomenon is an increase in the Raw Bit Error Rate (RBER) in a NAND flash memory. The RBER is the percentage of bits in error after a single read operation [49]. Such an increase translates into the inability to correct data after a number of Program/Erase operations (i.e., P/E cycles) or after long retention times.

Error Correction Codes (ECC) such as Bose, Chaudhuri, and Hocquenghem (BCH) ECC or even Low Density Parity Check (LDPC), are required to improve the reliability of saved data, while sophisticated write algorithms were designed for an optimal control of the threshold voltage distributions within the NAND flash [50, 51]. However, the relentless demand for storage capacity also impact non-volatile memories whose have quickly evolved from Single-Level Cells (SLC) to more complex Multi-Level Cells (MLC) memories in which more than one bit is stored in a single cell. This aggressive technology evolution, while broadening SSD exploitation also for big-data cloud servers, also results in an exponential RBER increase [52]. To deal with increasing RBERs, NAND and SSD controller vendors have introduced new read techniques such as the Read Retry (RR) and the LDPC Soft Decoding (SD), bridging the gap between the internal memory algorithms and ECC [53]. Basically, these approaches, leverage on a iterative read process of the page in error with the aim of either to reduce the RBER of the memory or to improve the error correction capabilities of the ECC (ECC_{th}) [54].

This procedure, however, inevitably exposes an overall performance degradation since a single page is available only after several read operations [55]. Moreover, although these algorithms are able to enhance the reliability of NAND flash memories, the memory controller has to be modified to carefully

handle these procedures. At this point it becomes straightforward that the performance/reliability trade-off introduced by emerging reliability enhancing algorithms for flash memories must be projected to the performance/reliability trade-off of the entire SSD.

In the next Sections a thorough evaluation of the performance/reliability trade-off in SSDs will be performed showing that:

- RR and SD algorithms have been devised to significantly improve the SSD reliability, without considering how they will be managed by the system and, therefore, how they impact on the performances of the disk;

- RR and SD introduce a read bandwidth degradation which is essential to determine whether it can be masked. In fact, while several disk operations such as wear leveling, garbage collection and bad block management can be masked at the firmware level (for instance by background execution) [50, 51], both the SD the RR algorithm, that is a fundamental part of the read operation, cannot be masked at the firmware level since they act directly on the storage medium. Therefore, only a direct action at the architectural level, especially in the host system interface and the ECC architecture, appears to be able to mask the performance drop.

2.3.1 The Read Retry: RR

RR are specific algorithms embedded in NAND flash chips which are able to heavily reduce the RBER of the memory either during endurance (the number of program/erase cycles) or retention time. These approaches leverage on a fine-grained tuning of the internal NAND flash read parameters (such as the read threshold voltage references) which are moved back and forth depending on the actual were-out state of the device [56, 57]. Thanks to this read voltage threshold shift it is possible to limit the number of read errors, and hence, to reduce the ECC correction efforts in terms of power consumption and time.

Figs. 2.7 and 2.8 show the RBER for a MLC NAND flash memory manufactured in the 2X node as a function of the P/E cycles and of retention time for a given P/E cycle, respectively. Data have been collected up to twice the rated endurance of the memory. The figures show the different average values for the upper and lower pages and that of the entire memory.

Increasing RBERs, result in a higher probability of erroneously decoding the bits read in a page. As a consequence, when the number of erroneous bits in a page exceeds ECC_{th}, the page content is no longer reliable. Figs.

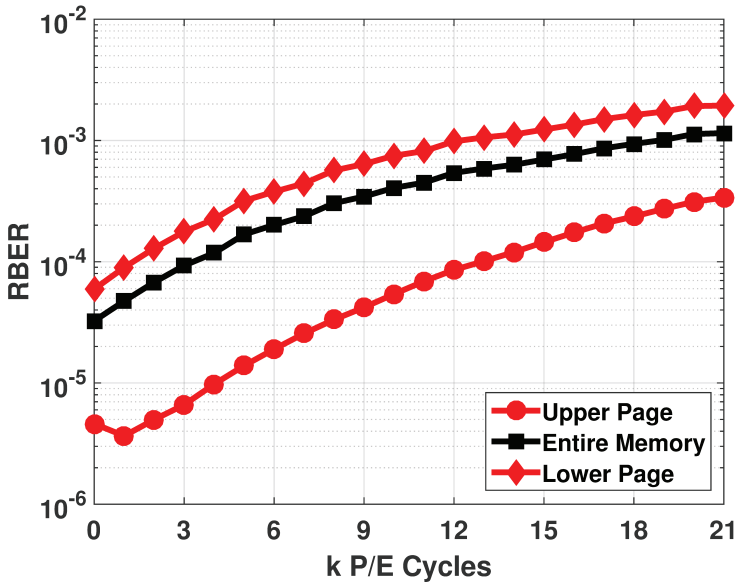


Figure 2.7: Example of RBERs in a commercial 2X node MLC NAND flash as a function of program/erase (P/E) cycles. The average values for the upper and

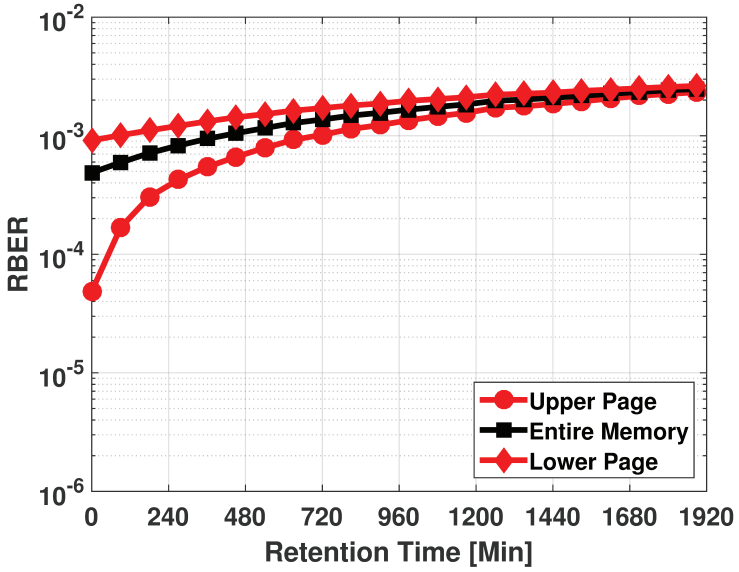


Figure 2.8: Example of RBERs in a commercial 2X node MLC NAND flash as a function of retention time after a specific P/E cycle. The average values for the upper and lower pages and that of the entire memory are shown.

2.9 and 2.10 show the percentage of read failures for the endurance and retention cases, respectively, when no RBER reduction techniques are applied. In this case the ECC is able to correct up to 100 bits in error in a 4320 Bytes codeword (ECC_{th}). The different behaviors exhibited by endurance and retention characterizations are due to the different drifts of the threshold voltage distributions: in particular, in the endurance case, only lower pages may produce uncorrectable read operations. In any case, however, the number of uncorrectable pages increases with the number of P/E cycles and retention time at a given P/E cycle.

Figs. 2.11 and 2.12 show the effectiveness of RR in reducing the RBER. As it can be seen, both endurance and retention are positively impacted by this algorithm, moreover, when it is applied no uncorrectable page events are detected by the ECC up to twice the rated endurance. However, even if the RR algorithm is able to reduce the RBER of a NAND flash memory, when it is used inside an SSD it has to be wisely exploited. In fact, whenever the ECC detects that a page cannot be correctly decoded, it can request for a RR intervention. In this case, in order to reduce the RBER, the page is read again from the memory with a set of modified voltage reference parameters and a fine-grained read voltage sensing process which prolong the memory read time. This process can be repeated several time until ECC is able to correct the target page. As a consequence, it becomes clear that since the occurrence of an uncorrectable page event (that is the reading of a page that results as uncorrectable if the RR technique is not applied) cannot be predicted and therefore the RR intervention cannot be forecasted, the increase in read latency may reduce the SSD bandwidth below acceptable values unless designed at the architecture level.

Table 2.2: SSD architectures considered to evaluate the RR algorithm

Parameter	**Architecture**			
Host Interface	SATA III			
DDR-Buffer	1			
ECC	1 x Channel			
Disk capacity	1 TB			
Configurations	**C1**	**C2**	**C3**	**C4**
Channels	1	2	4	8
NAND flash Targets per Channel	16	8	4	2

To test how the RR technique impacts the SSD's performance it has

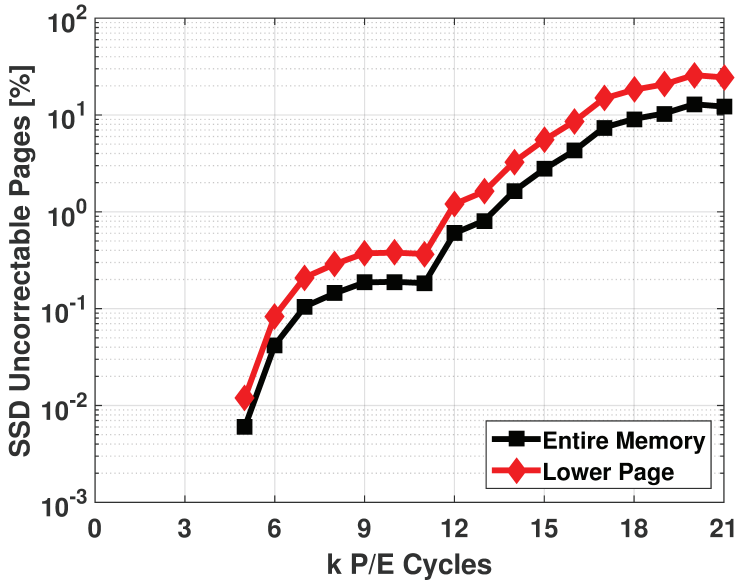


Figure 2.9: Percentage of uncorrectable pages (thus requiring the activation of the RR techniques) as a function of program/erase (P/E) cycles. The average value for the lower pages is shown. No uncorrectable upper pages have been

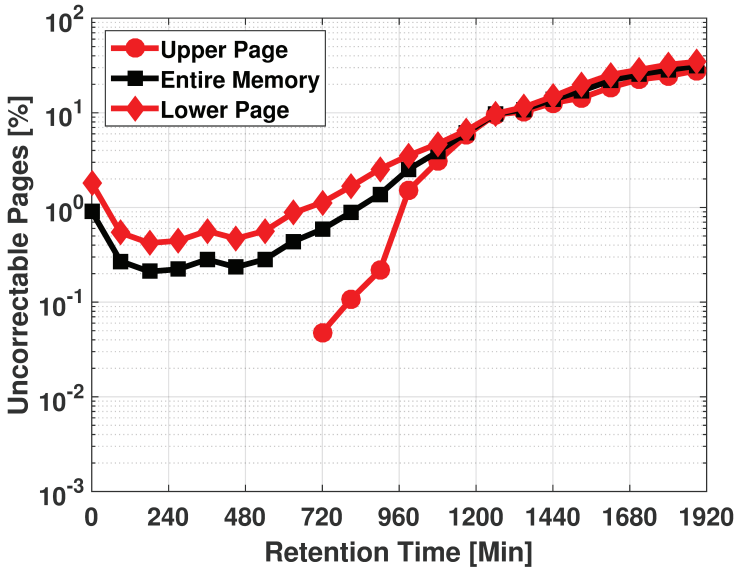


Figure 2.10: Percentage of uncorrectable pages (thus requiring the activation of the RR techniques) as a function of retention time after a specific P/E cycle. The average values for the upper and lower pages and that of the entire memory are shown.

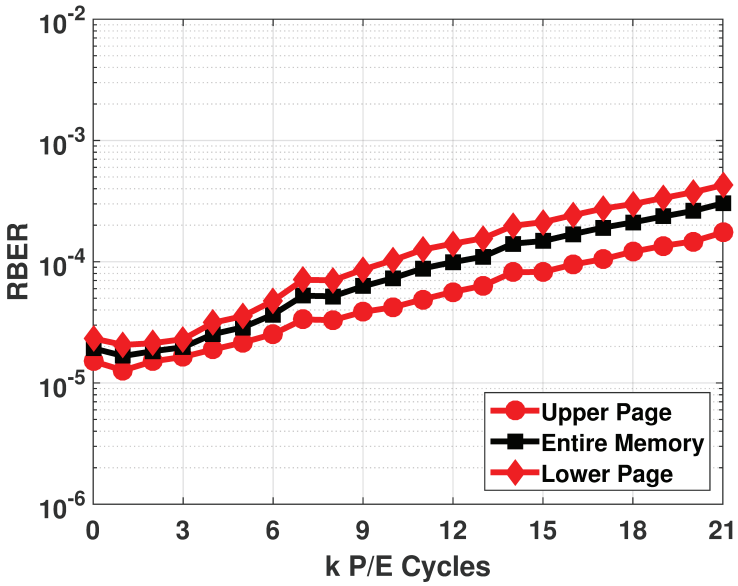


Figure 2.11: RBER after RR intervention as a function of program/erase (P/E) cycles. The average values for the upper and lower pages and that of the entire

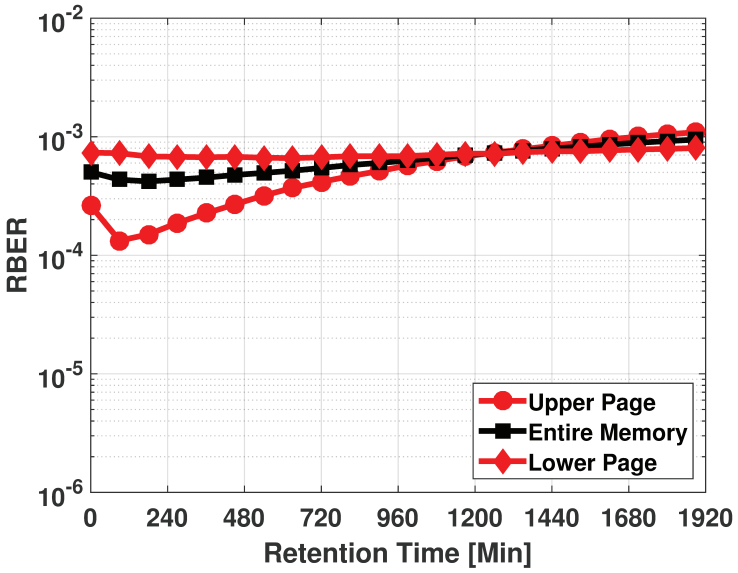


Figure 2.12: ERBER after RR intervention as a function of retention time after a specific P/E cycle. The average values for the upper and lower pages and that of the entire memory are shown.

been decided to simulate 4 different disk architectures summarized in Table. 2.2. It is worth pointing out that even if the SSD architecture is changed for different simulations, it has been decided to keep constant the disk capacity to avoid any possible problems in the data interpretation. As stated in Section 2.3.1, the two main components that contribute to the RR-induced drawbacks for the SSD performance are the NAND flash memories and the ECC. To this extent, in order to accurately reproduce their behavior, the ECC engine has been modeled choosing a BCH code able to correct up to 100 errors in a 4320 bytes codeword. For the latency modeling of the BCH decoder, which is the ECC block active during a read operation, the internal syndrome and Berlekamp-Massey modules follow the state of the art implementation provided in [32], whereas the Chien search machine is modeled as follows:

- a faster high parallel machine to serve a relative small number of errors;
- a slower yet ultimate accurate machine to guarantee the maximum correction capability.

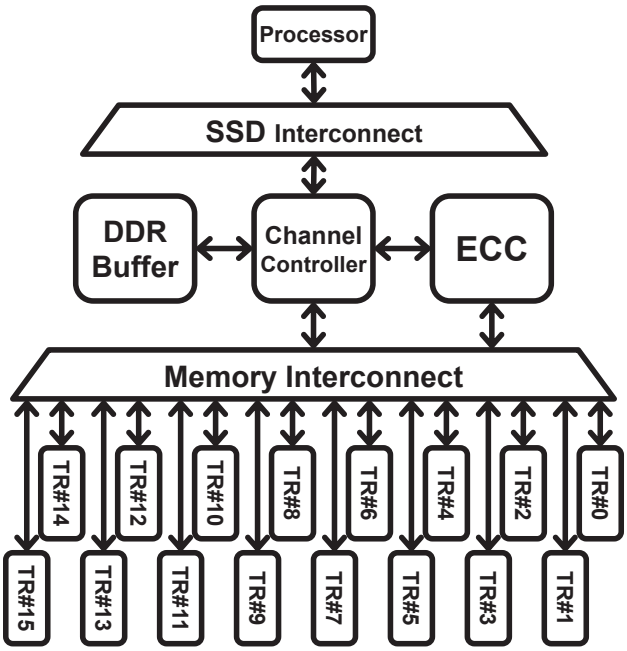


Figure 2.13: Baseline architecture considered in this Section

Concerning the NAND flash it has been modeled a 2X MLC technology with a page size of 16 kbytes plus Spare and a block size of 8Mbytes. From a previous characterization of the memory, the RBER values, the read latencies

of the memory, and the uncorrectable page event statistics have been calculated and back annotated into SSDExplorer. Starting from a RBER value exhibited by the NAND flash, the corresponding number of errors is extracted from a probability density function [58] and then fed into the modeled ECC engine in order to extract the exact delay incurred by the decoder.

2.3.1 A) Retry impact on SSD performance: fig. 2.13 shows the baseline architecture simulated which encompassed a single ECC BCH engine connected to a single channel serving 16 memory targets each composed by two dies.

For sake of clarity and to avoid any artifact in the performance analysis due to the inclusion of a host interface model (e.g. SATAIII [24] or PCI-E NVMe [27, 59]), it has been decided to neglect the aforementioned component. This approach allows evidencing the actual bandwidth related to the memory and ECC subsystems only.

The main result concerns the SSD read bandwidth as a function of the RBER which tracks the memory wear-out (see Fig. 2.14.) Since the real SSD behavior depends on a combination of P/E cycles and retention times, the use of RBER as a reference metric allows to ignore the actual degradation mechanism while generalizing the performance evaluations. Two points are highlighted in the figure, representing a 2% drop (point A) and a 10% drop (point B) with respect to the SSD read bandwidth at Beginning of Life (BoL), respectively. The former point will be used to show in details the behaviors of the ECC engine and of the NAND memories in a normal read condition whereas the latter point, representing the maximum acceptable degradation level, will be used to illustrate architectural limitations when the number of uncorrectable pages become significant.

Fig. 2.15 shows, for point A, the distribution of the correction times spent by the ECC engine which depend on the number of errors to be corrected, whereas Fig. 2.16 shows the read latency distribution in a NAND flash. This latter figure evidences the following main contributions: the read times for the upper and lower pages and those ascribed to internal read scheduling mechanisms. As it can be observed, the ECC correction times are one order of magnitude lower than the memory read latencies and, therefore, it has a marginal impact on the disk performance.

Figs. 2.17 and 2.18 show the same distributions for point B. In particular, a broadening in the ECC correction time distribution can be observed. The maximum correction time, evidenced in Fig. 2.17, is related to the case of an uncorrectable page. Fig. 2.18, on the other hand, shows that when a significant number of uncorrectable pages is detected with the consequent

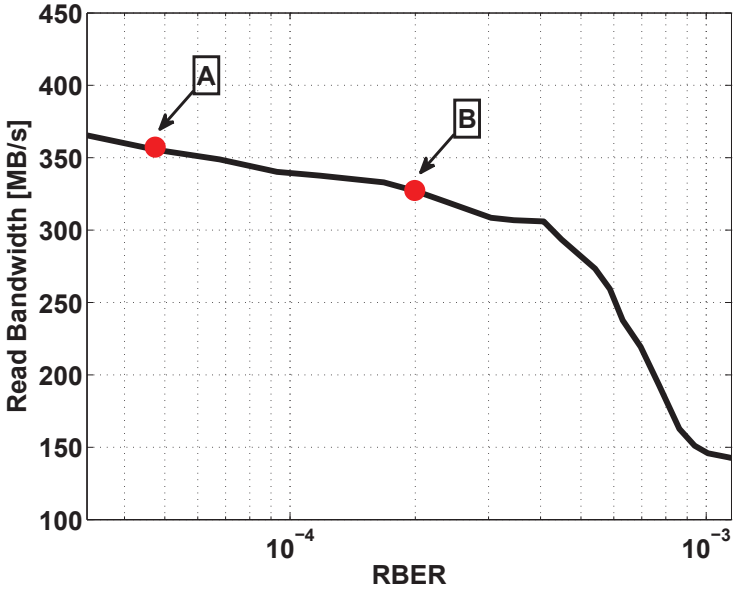


Figure 2.14: SSD read bandwidth vs. RBER for the baseline architecture. Point A and B mark a 2% and a 10% performance degradation compared to the BoL, respectively.

RR activation, two other contributions to the NAND flash read latency are introduced: the former related to the commands activating RR, the latter associated to the actual execution time of the RR algorithm.

Finally, it must be stressed that the percent of uncorrectable page readings producing a 10% drop in the SSD read bandwidth is in the order of 0.6% to 0.8%.

2.3.1 B) Retry impact on the SSD architecture: as shown in Section 2.3.1 A), the performance degradation caused by uncorrectable pages requiring the RR activation is related to the sum of the memory latency and of the ECC correction time, this latter resulting as negligible in normal reading conditions. This implies that the ECC engine is the module to address to try reducing the read bandwidth degradation. To this purpose it has been repeated the analysis by modifying the number of channels and, therefore, the number of ECC engines. Fig. 2.19 shows the read bandwidth behavior as a function of RBER, up to the point corresponding to a 10% degradation with respect to BoL, for the four channels/NAND flash targets configurations analyzed (see Table 2.2).

As expected, by increasing the number of channels (in particular architectures C3 and C4) it is possible to achieve higher bandwidths. The RBER

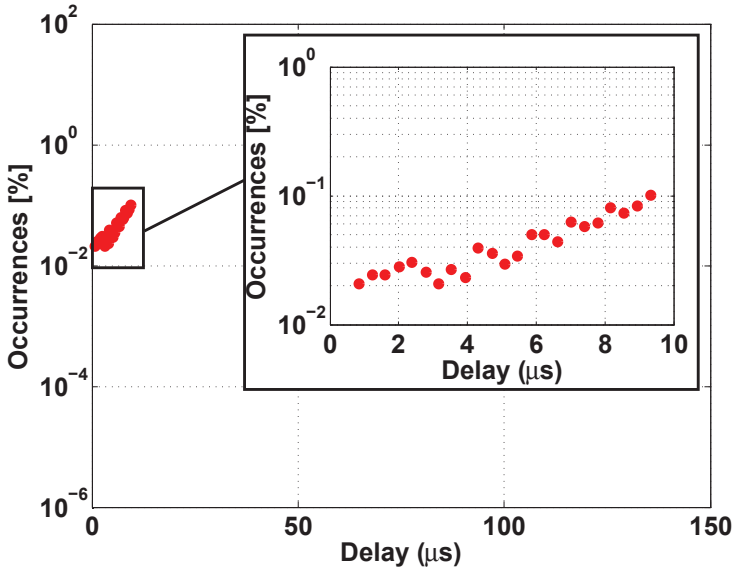


Figure 2.15: Distribution of the ECC correction times at point A of Fig. 2.14.

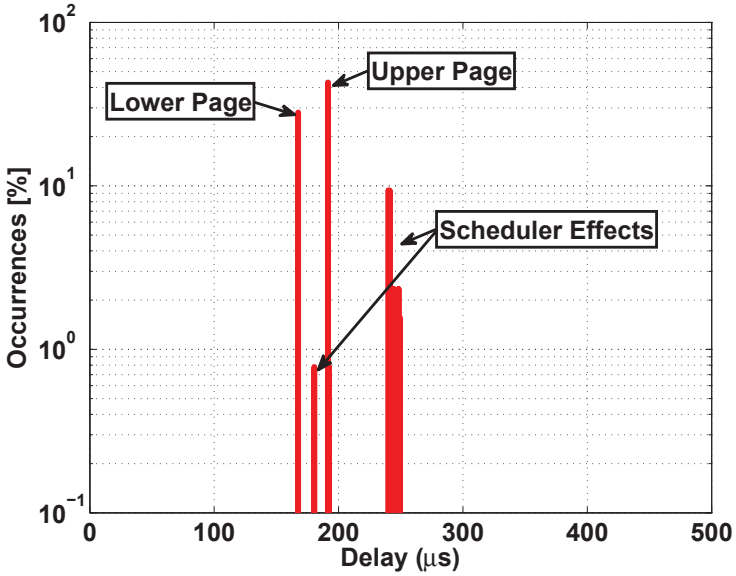


Figure 2.16: Distribution of the NAND memory latencies at point A Fig. 2.14.

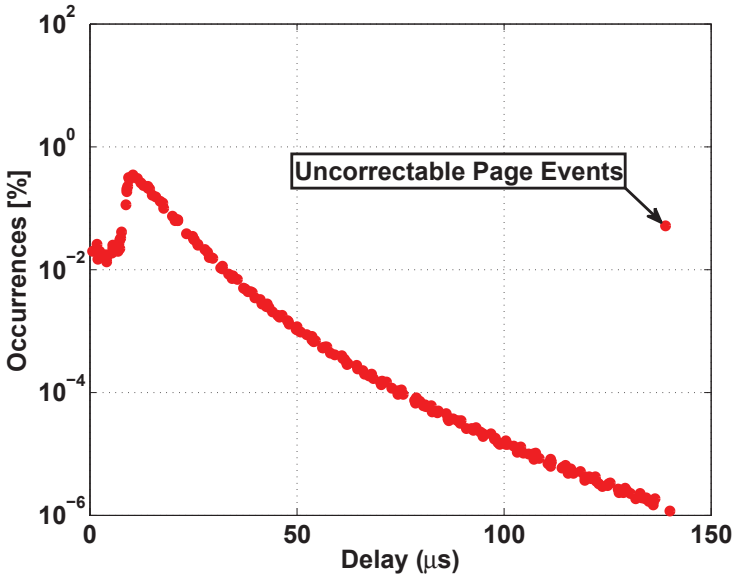


Figure 2.17: Distribution of the ECC correction times at point B of Fig. 2.14.

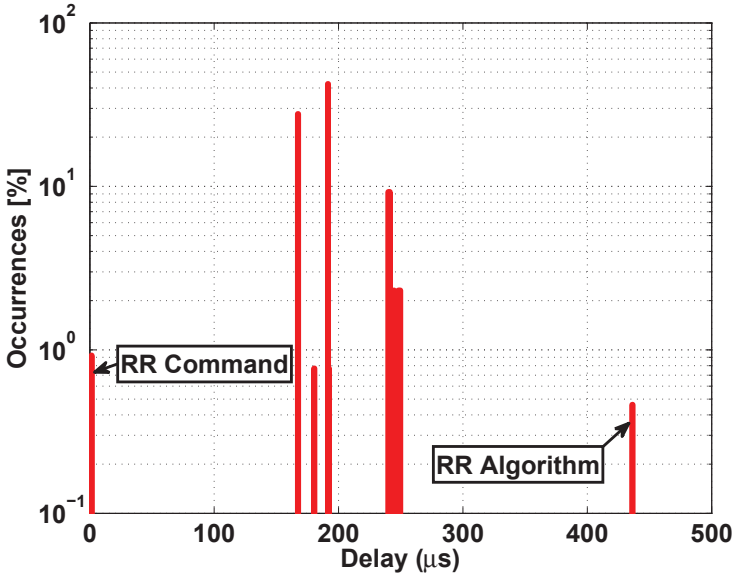


Figure 2.18: Distribution of the NAND memory latencies at point B of Fig. 2.14.

corresponding to a 10% degradation, however, is in the same order of magnitude for the four cases. So merely increasing the architectural resources, without considering the interaction of the host interface requirements, does not seem to solve the problem of the RR impact on the SSD performance.

When considering the SSD system as a whole, it is clear that the overall performance is driven by the slower between the ECC/memory system and the host interface. Therefore, if the host interface bandwidth is over designed with respect to that of the ECC/memory system, the performance drop caused by memory wear-out and RR techniques is noticeable by the user. If, on the contrary, the host interface bandwidth represents the system bottleneck, the ECC/memory system must be designed so that, without any useless silicon waste measured as channel number and flash target increase, the performance drop caused by memory wear-out and by RR techniques does not bring the ECC/memory bandwidth below that of the host interface. As an example, considering a SATA III host interface, in Fig. 2.20 it is shown that the configuration C3 barely mask the RR-induced performance drop, whereas SSD configuration C4 allows sustaining a higher wear-out level for the NAND flash targets before exposing to the SSD user an effective read bandwidth drop.

2.3.2 LDPC Soft Decision: SD

As discussed in Section 2.3, one main SSDs' limitation is their reliability, which is dependent on the non-volatile NAND flash memories used as storage medium. The RBER of these devices is steadily growing because of the aggressive technology scaling needed to increase the memory capacity. Such an increase translates into the inability to read correct data after a number of P/E operations or after long retention times. Fig. 2.21 shows the measured average RBER as a function of endurance for three MLC and one TLC NAND flash memories manufactured in 2X, 1X, and mid-1X technology nodes as described in Table. 2.3. As it can be seen, as the number of P/E cycles increases, the error rate quickly grows up. In addition, either by scaling from a 2X to a mid-1X node or switching from a MLC to a TLC storage paradigm, the RBER increases significantly.

To broaden NAND Flash reliability figures and, consequently, data trustworthiness over the whole SSDs' lifetime, the sole use of RR algorithms with BCH decoders is no longer sufficient (see Table 2.4). In fact, as it can be seen in Fig. 2.22, using this approach on these memories does not give the expected results since, after their intervention, the percentage of uncorrectable pages still remains high. Therefore, it is clear that to guarantee the data reliability over the whole memory endurance, it is necessary to exploit so-

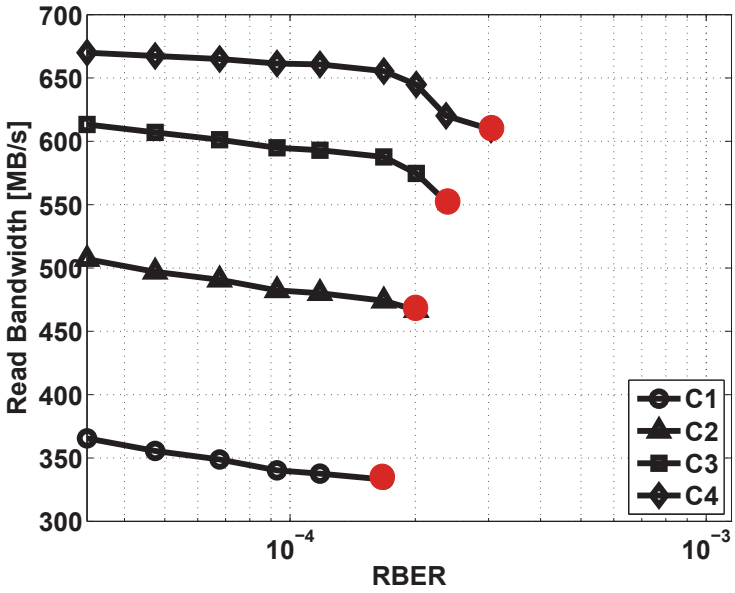


Figure 2.19: SSD read bandwidth vs. RBER for the four architectures described in Table 2.2. Curves are shown up to a 10% performance degradation with respect to BoL.

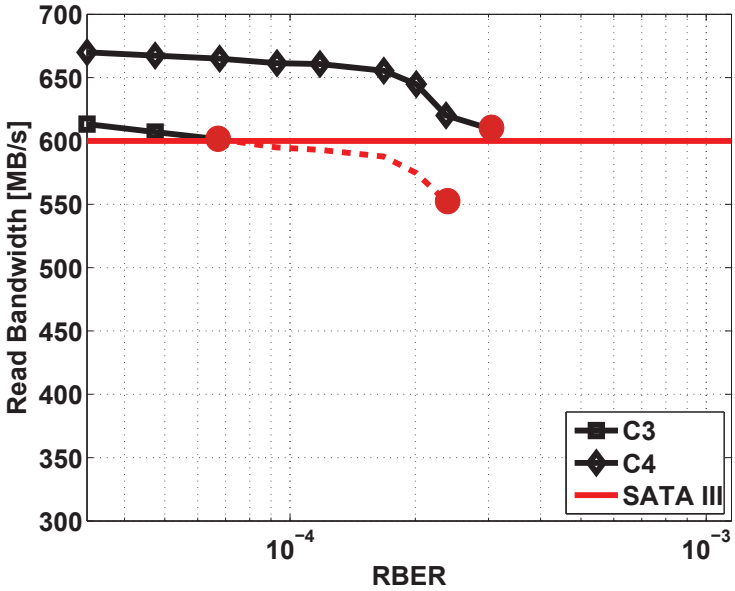


Figure 2.20: SSD read bandwidth vs. RBER for the C3 and C4 architectures compared with the SATAIII theoretical bandwidth. Configuration C4 allows sustaining a larger wear-out prior exposing the performance degradation induced by RR to the user.

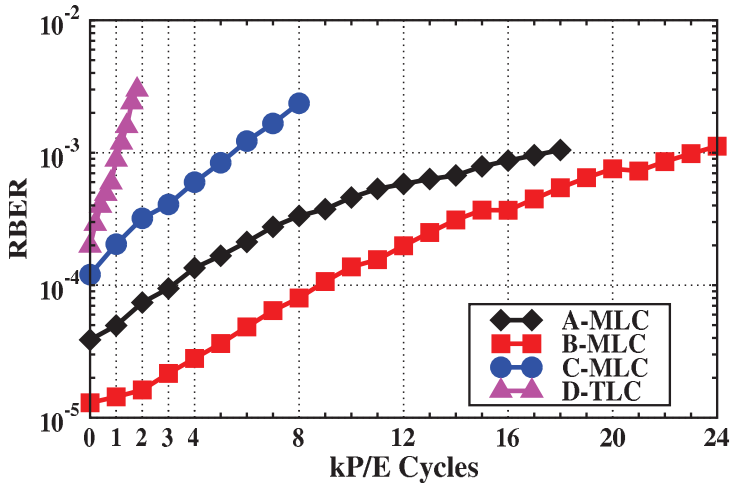


Figure 2.21: Measured average RBER up to twice the rated endurance in 2X, 1X and mid-1X technology node MLC and TLC NAND flash memories as a function of P/E cycles.

Table 2.3: Main characteristics of tested NAND flash memories.

Sample	**Memory type**	**Rated endurance**	**Measured avg. read time**	**Measured avg. program time**	**Technology node**
A-MLC	Consumer	9 k P/E	68μs	1400 μs	2X
B-MLC	Enterprise	12 k P/E	40μs	2000 μs	1X
C-MLC	Enterprise*	4 k P/E	70μs	2500 μs	Mid-1X
D-TLC	Enterprise*	0.9 k P/E	86μs	2300μs	Mid-1X

*Early samples

phisticated ECC such as the LDPC.

Table 2.4: Measured endurance sustained with a BCH ECC engine and the RR algorithm.

Sample	Measured endurance
A-MLC	6 kP/E
B-MLC	19 kP/E
C-MLC	5 kP/E
D-TLC	1 kP/E

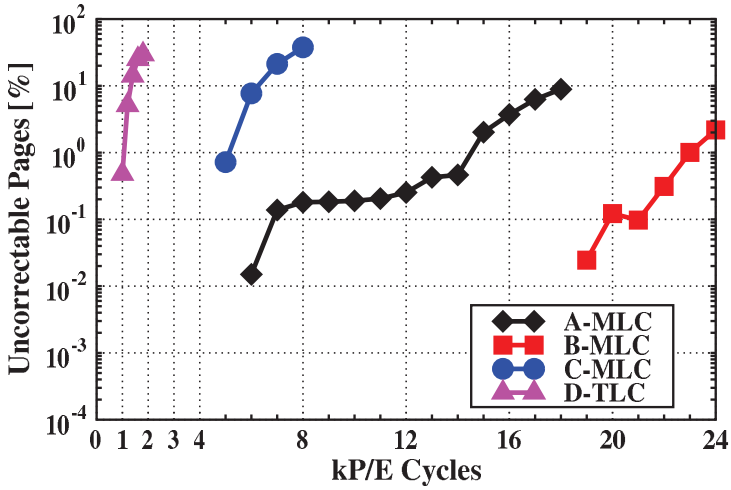


Figure 2.22: Measured percentage of uncorrectable pages up to twice the rated endurance as a function of P/E cycles. A BCH with a capability to correct up to 100 bits in error in a 4320 Bytes codeword and the RR algorithm are used.

Due to their superior error correction capabilities, Low Density Parity Check (LDPC) codes now represent a forced choice for SSDs [60, 61]. Conventional LDPC decoders, if properly designed, can sustain a NAND Flash RBER up to 10^{-2} [61, 62, 63, 15]. The LDPC correction engine usually leverages on two sequential correction approaches: *i)* the hard decision (HD) which corrects errors by means of a single read operation of the selected memory page; *ii)* a sequence of soft-level decisions (SD) which perform, with considerably higher latency, a fine-grained multiple-read sensing operation exploiting the RR command made available by NAND flash memories.

As summarized in Fig. 2.23, each soft-level requires two page read operations with two different read references and two data transfers to the ECC

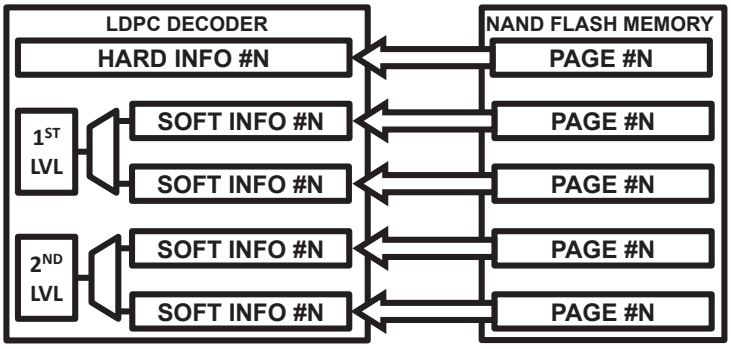


Figure 2.23: Standard LDPC decoding (HD + two-level-SD). Each soft-level requires two extra read operations and two data transfer operations.

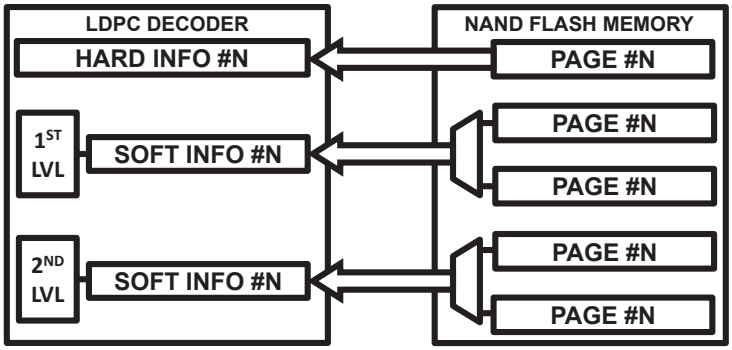


Figure 2.24: LDPC decoding (HD + two-level-NASD). Each soft-level requires two extra read operations and only one data transfer since read data are conventionally combined before data transfer.

engine. The algorithm continues this process until the page is correctly read or the maximum number n of soft-levels is reached and the page is marked as uncorrectable. The overall n-level SD algorithm requires $2n$ page reads and $2n$ data transfers operations. This serial approach is used mainly because high code-rates [64] are adopted to exploit the full SSD capacity and hence HD has the same limitations of BCH codes in terms of RBER [15]. Therefore as soon as this strategy fails to correct data, it is requested the intervention of the SD, with a higher correction range. However, in [15] it has been shown that, as soon as the HD approach starts to fail, there is an overhead both in terms of increased SSD power consumption and overall SSD latency since additional read operations all requiring the RR algorithm are requested with respect to the HD approach.

An alternative LDPC correction approach that limits the drawbacks of the SD has been presented in [65]. The assumption of this methodology, named NAND-Assisted Soft Decision (NASD), is that data for ECC engine are produced by the NAND flash memory itself, which internally reads the target page twice for each soft-level. Then, read data are combined and only one transfer to the ECC is performed for each soft-level, as shown in Fig. 2.24.

Soft Decision VS NAND-Assisted Soft Decision

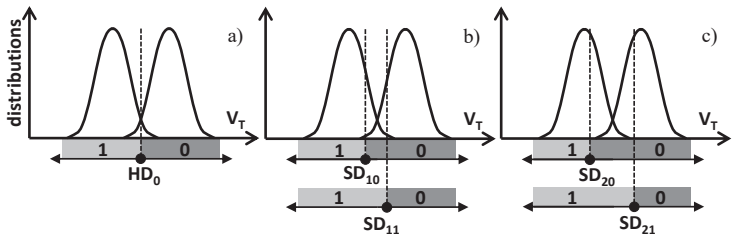


Figure 2.25: Two levels LDPC sensing scheme. A memory page is read by setting the read voltage at HD_0 and determining, for each bit, whether $V_T < HD_0$ or $V_T > HD_0$ (a). If the ECC engine is not able to correct possible read errors, the soft decision algorithm starts and the page is read twice by moving the read references around HD_0, at SD_{10} and SD_{11} (b). If the page is still marked as uncorrectable, the page is read again with the SD_{20} and SD_{21} references (c).

Flash memories are read page-wise by using a defined read reference, hereafter denoted as HD_0. Cells are read as 1 or 0 depending on their threshold voltage V_T with respect to HD_0 (see Fig. 2.25a). If during the ECC decoding phase the page is evaluated as uncorrectable, the LDPC decoding algorithm

can be retried with the SD. To accomplish this second step, more information about the actual position of the NAND flash threshold voltage distributions must be collected. Basically, the algorithm moves sequentially the internal read references to SD_{10} and SD_{11} (Fig. 2.25b) thus reading the page twice and storing the two data content in two page registers inside a page buffer. Data from the page buffer are transferred byte-wise from the flash memory to the LDPC decoder to be analyzed with those previously read with HD_0. If the decoding process still fails, a second iteration is performed by moving the read references to SD_{20} and SD_{21} as shown in Fig. 2.25c. The algorithm continues this process until the page is correctly read or the maximum number of soft-levels is reached and the page is marked as uncorrectable.

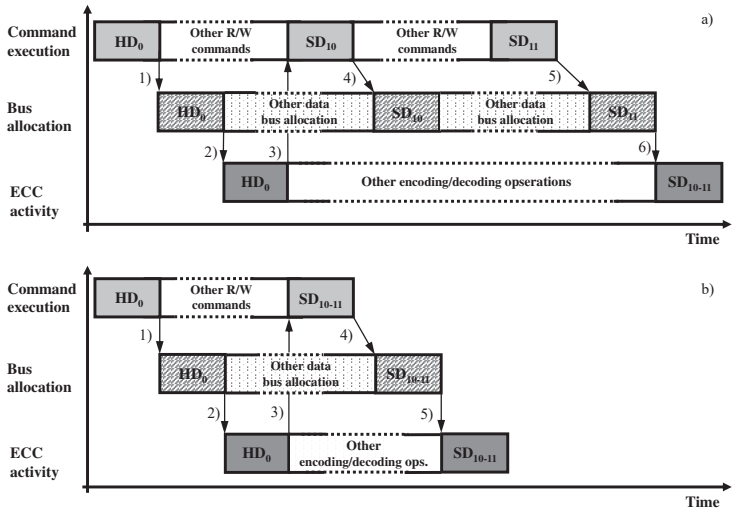


Figure 2.26: Time sketch, for a cluster of NAND flash dies sharing the same data bus, of the command queue, of the data bus allocation, and of the ECC engine activity. Numbers enlighten the events sequence during a single soft-level decision operation. Case a) and b) refer to the SD and NASD approach, respectively.

Table. 2.5 summarizes the number of operations performed by both algorithms. As it can be seen, NASD is able to halve the number of page transferred from the NAND flash memory to the ECC. As a consequence, the overall soft decision process is shortened and hence, the SSD performance are improved. To understand the effective NASD efficiency, it must be taken into account that read operations are temporally separated from the successive data transfer operations. Fig. 2.26 sketches the commands queue for NAND flash dies sharing the same I/O bus, the corresponding data bus allocation, and the ECC engine activity. After a HD_0 read, the

SSD controller can send other read or write commands to the same NAND flash die or to other dies. When the ECC engine communicates the read failure to the controller, additional SD_{10} and SD_{11} reads are scheduled. In the SD approach the two read data are transferred separately when the I/O bus is available, with the risk that between the SD_{10} and SD_{11} transfer the bus is contended by other data transfers to/from other NAND flash dies (see Fig. 2.26a). In the NASD approach, on the contrary, since SD_{10} and SD_{11} read data are combined in a single data transfer, the consequent soft decision operation can start in advance with respect to the SD case (see Fig. 2.26b). The advantages, that become more pronounced when additional soft-levels are considered. Moreover, since the number of data transfers between the memory and the ECC are reduced, NAND flash memory I/O buses access are reduced as well and hence a considerable power reduction is provided.

Table 2.5: Read and data transfer operations in SD and NASD approaches.

LDPC	**One soft-level**	**Two Soft-levels**	**#n soft-levels**
SD	2 page read 2 data transfer	4 page read 4 data transfer	#$2n$ page read #$2n$ page transfer
NASD	2 page read 1 data transfer	4 page read 2 data transfer	#$2n$ page read #n page transfer

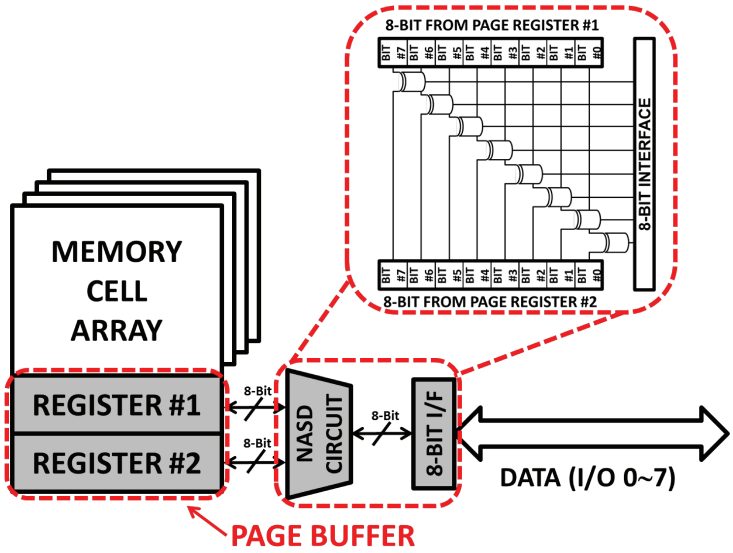


Figure 2.27: NASD combinational circuitry. Just 8 logic XOR gates (or XNOR) have to be added before the 8-bit I/O interface.

The main component exploited by NASD is the NAND flash page buffer

which is used to store data for each soft-level operation. In present NAND flash chips, this buffer is composed by two registers used especially for read cache and read retry operations [66, 67, 68, 69]. At this point it becomes clear that the NASD implementation does not require any other register inside the memory and it can be performed by a simple combinational logic placed between the internal NAND flash page buffer and the I/O interface. In fact, the two read operations performed by NASD can be easily stored into the existing registers of the page buffer and a simple block composed by 8 XORs (or 8 XNORs) acting as a combinational circuitry is sufficient. Since the I/O interface limits the parallelism to 8-bits, the logic combination between the pages stored inside the two registers can be performed on-the-fly in a byte-wise fashion during the data transfer phase (see Fig. 2.27). Regardless of the internal NAND architecture, just a single combinational logic can be integrated in a single NAND chip. As a consequence, the NASD implementation inside a NAND flash memory becomes a easy task which does not impact neither chip area nor power consumption.

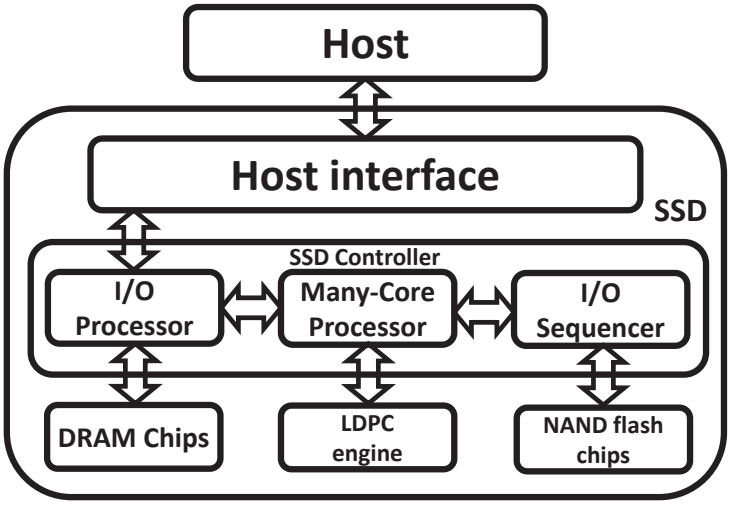


Figure 2.28: Simulated SSD architecture.

In order to test the effectiveness of NASD with respect to the starndard SD, a 512GB SSD composed by 8 channels with 8 NAND flash dies per channel has been simulated. The simulations have been performed considering the 4 different memories presented in Table 2.3 as SSD's storage medium. Fig. 2.28 shows the main building blocks of the SSD. Besides the standard I/O Processor exploited for the host-interface address fetch phase and the many-core Processor on which the Flash Translation Layer is executed, there is also an I/O Sequencer acting as a read/write dies scheduler. In order to fully

exploit the internal parallelism offered by the SSD, host random addresses which could cause die collisions (i.e., requests for a die already scheduled) are parsed and sequentially issued to NAND flash chips. In such a way, even if random commands are sent by the host, only sequential patterns are processed by NAND flash memories hence maximizing the throughput.

Table 2.6: Tested Host system configurations

	Consumer	**Enterprise** [70]
Host Processor	Intel-Core i5-4570	Intel-Xeon e5-2630
Processor clock	3.2 GHz	2.3 GHz
#N Cores	4	24
DRAM size	12 GByte	16 GByte
Workload generator [71]	fio 2.1.10	fio 2.1.10
Avg. I/O submission time	3.5 μs	0.5 μs
Host queue depth [27]	64	256
Host kIOPS (requested)	$\approx$ 200	$\approx$ 600
SSD kIOPS (sustained)	$\approx$ 450	$\approx$ 450

All data have been collected simulating two different host platforms (Table. 2.6). The first one is a consumer system which does not exploit the full SSD architecture (able to sustain 450 kIOPS) since I/O requests settle around 200 kIOPS. As a consequence, all internal error recovery techniques which exploit additional read operations produced by the ECC for the soft decoding step are partially hidden by the SSD's architecture which masks all non-user reads. The second one is an enterprise workstation designed to serve hundreds of parallel processes which requests up to 600 kIOPS. In this case the disk performance cannot match this specification so that any further read produced by any error recovery technique will burden on the final SSD's performance. Thanks to these two different test-cases it has been possible to test the NASD effectiveness over standard SD when disk resources such as NAND-flash I/O buses are partially or completely allocated for user operations.

Results presented in Section 2.3.2 A) refer to an enterprise host and a 100% 4 kB random read workload which represents the most challenging situation for the SSD performance characterization. In fact, when mixed read/write workloads are considered, since the DRAM chip in the SSD caches all the write operations, the measured average latency and bandwidth figures of the disk do not reflect the actual SSD behavior. Section 2.3.2 B) will extend the discussion to realistic workloads for both hosts.

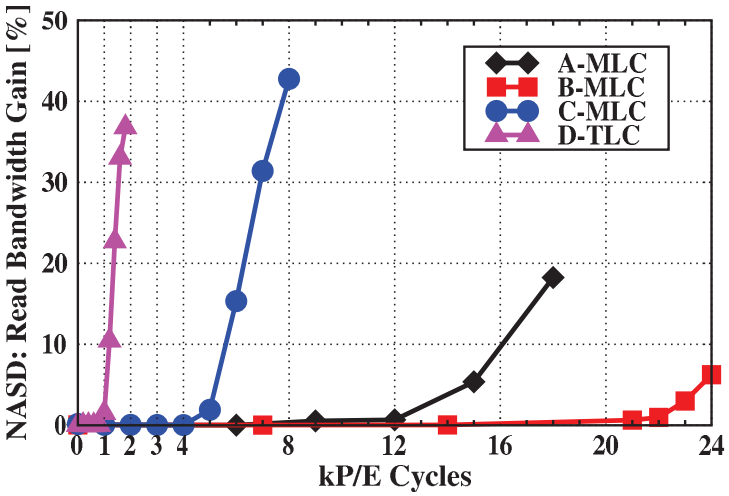


Figure 2.29: SSD read bandwidth gain achieved by NASD with respect to SD as a function of the memory endurance for the 4 considered memory types and the Enterprise host.

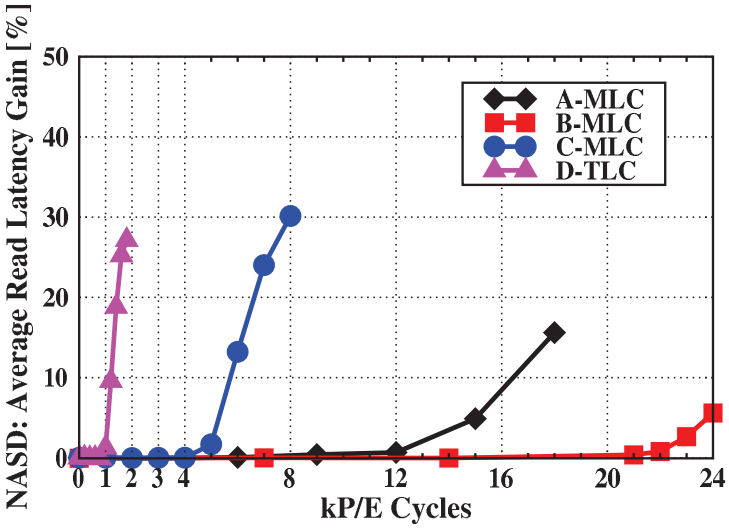


Figure 2.30: SSD average read latency gain achieved by NASD with respect to SD as a function of the memory endurance for the 4 considered memory types and the Enterprise host.

2.3.2 A) 100% random read workload - Enterprise host: fig. 2.29 shows the SSD's read bandwidth gains achieved by the NASD approach with respect to the SD, as a function of the memory endurance. Bandwidth (IOPS) has been calculated as the average number of read commands completed in a second. As it can be seen, for all the considered memories the NASD technique provides a significant gain. NASD advantages are more pronounced when large number of uncorrectable pages, triggering a massive ECC intervention, are detected.

Fig. 2.30 shows the average read latency gains achieved by NASD with respect to SD as a function of memory endurance. Latency has been calculated as the average time elapsed between a read command submission and its completion. All results concerning average latency reflect those obtained for bandwidth (Fig. 2.29).

Fig. 2.31 shows the SSD's cumulative latency distributions calculated at twice the rated endurance for the D-TLC sample and both SD and NASD approaches. From these data it is possible to extract the SSD's QoS defined as the 99.99 percentile of the cumulative latency distribution [46]. QoS represents the predictability of low latency and consistency of high bandwidth while servicing a defined workload and it can be considered as the key metric to assess the SSD's performance in a worst-case scenario. Fig. 2.32 shows the calculated QoS at twice the rated endurance for all the considered memories and for both the SD and NASD approaches.

2.3.2 B) Realistic workloads - Enterprise and Consumer hosts Since the NASD advantages are tightly coupled to the command pattern, besides the 100% random read workload considered so far, simulations have been also performed considering three realistic workloads [72], as detailed in Table 2.7 where *write ratio* represents the percentage of write commands in the command sequence, whereas *write amplification factor* denotes the number of additional writes produced by the SSD firmware for each single host write [6].

Table 2.7: Workloads characteristics

Workload	**Write ratio [%]**	**Write amplification factor**
MSN	96	1
Financial	81	1.32
Exchange	46	1.94

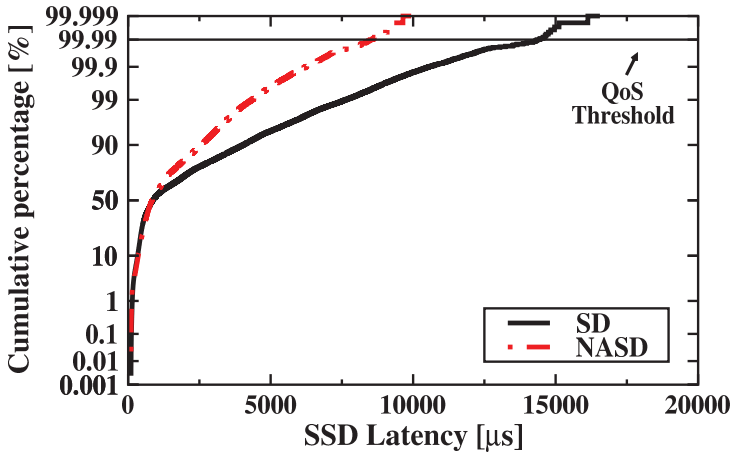


Figure 2.31: Cumulative percentage on a normal probability paper of the SSD latency calculated at twice the rated endurance when a D-TLC sample is used and both SD and NASD are considered. The QoS threshold is calculated as the 99.99 percentile of the cumulative distribution [46].

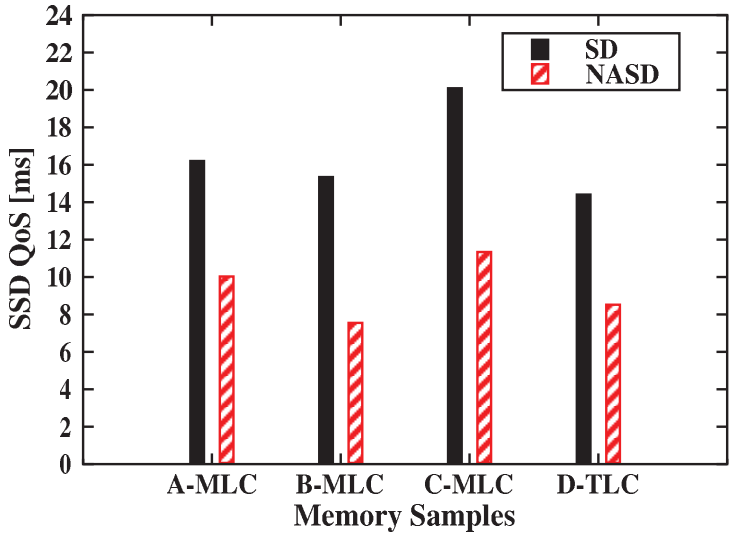


Figure 2.32: Calculated QoS at twice the rated endurance for the 4 considered memory types and the Enterprise host.

Table 2.8: Bandwidth (in kIOPS for SD and in % of gain for NASD vs SD) @ twice the rated endurance for both the consumer and the enterprise host

Workload	Consumer Host								Enterprise Host							
	A-MLC		B-MLC		C-MLC		D-TLC		A-MLC		B-MLC		C-MLC		D-TLC	
	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD
MSN	143	4.78	147	4.17	142	5.26	141	5.23	142	4.14	148	4.79	142	5.94	142	5.72
Financial	135	1.45	142	0.54	101	3.68	104	4.36	143	1.88	148	0.45	119	3.67	118	4.19
Exchange	143	2.25	151	0.60	94	4.95	97	5.40	171	2.47	187	0.44	126	5.28	127	6.0
100% read	204	0.03	204	0.03	140	24.41	127	24.85	299	18.24	402	6.24	152	42.77	156	36.80

Table 2.9: Average latency (in μs for SD and in % of gain for NASD vs SD) @ twice the rated endurance for both the consumer and the enterprise host

Workload	Consumer Host								Enterprise Host							
	A-MLC		B-MLC		C-MLC		D-TLC		A-MLC		B-MLC		C-MLC		D-TLC	
	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD
MSN	373	2.24	343	0.11	393	5.46	396	3.57	1485	0.57	1468	0.16	1553	1.78	1554	1.92
Financial	467	1.50	442	0.53	624	3.60	610	4.29	1724	1.61	1651	0.19	2088	3.99	2093	4.41
Exchange	442	2.21	417	0.60	673	4.84	646	5.21	1465	2.28	1343	0.47	1979	5.28	1973	5.89
100% read	312	0.01	311	0.01	454	19.71	502	19.90	834	15.6	623	5.58	1654	30.14	1620	27.15

Table 2.10: Quality of Service (in *ms* for SD and in % of gain for NASD vs SD) @ twice the rated endurance for both the consumer and the enterprise host

Workload	Consumer Host								Enterprise Host							
	A-MLC		B-MLC		C-MLC		D-TLC		A-MLC		B-MLC		C-MLC		D-TLC	
	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD	SD	NASD
MSN	47.07	33.95	32.06	36.32	25.17	22.24	34.58	20.46	53.88	28.32	35.48	34.11	31.62	22.73	45.24	22.34
Financial	14.26	17.85	11.43	22.56	12.36	14.59	13.59	13.00	92.60	37.58	80.65	32.61	71.83	5.25	80.75	26.16
Exchange	7.50	21.35	5.93	14.04	9.37	15.11	8.76	14.00	44.20	22.45	37.83	29.53	39.89	16.46	44.21	23.69
100% read	1.50	23.08	0.77	20.85	2.67	21.47	1.59	29.42	16.20	38.10	15.34	50.80	20.08	43.56	14.40	40.84

Tables 2.8 ÷ 2.10 show the bandwidth, the average latency, and the QoS at twice the rated endurance for the 4 tested NAND Flash memories and for the two host architectures. Simulation results show that NASD outperforms SD when between the two data transfers required by the SD technique other commands are scheduled, thus temporally separating the data transfer operations and introducing a performance degradation. When realistic workloads are considered, NASD advantages are clearly evident for the MSN workload, which is characterized by a high number of program operations whose duration is significantly higher than read operations. NASD advantages are highlighted when QoS is concerned since it takes into account the worst-case latency conditions rather than an average behavior such as bandwidth and average latency. As it can be observed, the QoS improvements for the MSN workload are in a 20% ÷ 40% range.

Chapter 3

Design trade-offs for RRAM-based SSDs performance

As presented in the previous chapters, when high performing storage devices are needed, SSDs are the most effective solution for both consumer applications and large enterprise environments [1]. However, the increased volume of produced and processed data, especially in the latter case, calls for a higher storage density of the SSDs which depends on the capacity features of the exploited storage medium: the NAND Flash memories. However, as already shown in Chapter 2, such a density increase sacrifices the memory performance and inherent reliability, leading to an unacceptable degradation of the main figures of merit for a SSD: latency and bandwidth [33].

In the last years, a growing interest on possible candidates for NAND Flash replacement in SSDs is represented by the Resistive RAM (RRAM) technology [73]. These memories offer the non-volatility feature of NAND flash devices with a lower read/write latency, higher scalability, lower power consumption and higher long-term reliability. However, up today, the reduced storage capacity of RRAM memories [74] has limited their usage to specific purposes such as boosting the SSD's performance or saving critical data during power fault events like in the hybrid system described in [75]. In this case, RRAM are used as a Storage Class Memory, which, coupled with NAND flash memories, are able to minimize the latency and improve the overall system bandwidth-reliability figures of the disk.

The advent of high density 1T-nR RRAM crosspoint arrays fully compatible with the state-of-the-art NAND Flash interface [76, 77] is paving the way to innovative "All-RRAM" SSD's architecture. In these systems, NAND flash memories are completely replaced by RRAM devices which offer a highly

reliable and extremely faster storage medium.

In this section a thorough design space exploration of a 1T-nR 512 GBytes "All-RRAM" SSD architecture is performed locating possible architectural bottlenecks and inefficiencies. Without any lack of generality, it has been decided to devise the compatibility of RRAM chips with the standard NAND flash interfaces [22], and hence a traditional SSD controller is embodied in the simulation environment without any dedicated RRAM algorithms. Collected results show that "All-RRAM" SSDs fit the need of extremely low latency only when a proper management of the operations and their queuing mechanism is provided.

3.1 "All-RRAM" SSD configuration

The RRAM chip considered in the proposed "All-RRAM" SSD architecture is a configurable 16 planes 32 Gbits memory module with a 8 bits ONFI 2.0/Toggle Mode compliant capable of 200 MT/s [22] (see Fig. 3.1). Each plane is composed by a 2 Gbits crosspoint RRAM array with a page size of 256 Bytes [76]. The RRAM chip features an internal memory controller that can work either with a native addressing mode (i.e., each addressed page is 256 Bytes-wide) or in a multi-plane emulated addressing mode which allows accessing from 512 Bytes up to 4 kBytes within a single operation. A read operation takes 1μs per page. The main array characteristics of this technology are summarized in Table 3.1.

The simulated SSD configuration (depicted in Fig. 3.2) is a 512 GBytes disk composed by 16 populated channels with 8 RRAM targets each. The SSD controller configuration, the additional Error Correction Code modules and dedicated DRAM buffers are included to keep the compatibility with state-of-the-art NAND-Flash based SSDs [78]. The exploited disk interface is a Gen2 PCI-Express port with 8 lanes on top of which the NVMExpress protocol is used [27]. Finally, different 100% random read workloads have been used to fully exploit the native and the emulated addressing mode of the considered RRAM chips by aligning the logical block address of the disk with the effective RRAM page size. Write workloads are not considered in this chapter since the targeted disk architecture includes DRAM buffers where write operations are cached on top of it, thus not representing a significant burden for the latency and bandwidth figures.

Table 3.1: Main characteristics of the simulated RRAM devices

Chip parameter	Configuration
IO-Bus interface	ONFI 2.0-Toggle Mode
IO-Bus speed	200MT/s
Native Page Size	256 Bytes
Emulated Page Size	512-1024-4096 Bytes
T_{READ} per Page	1 μs

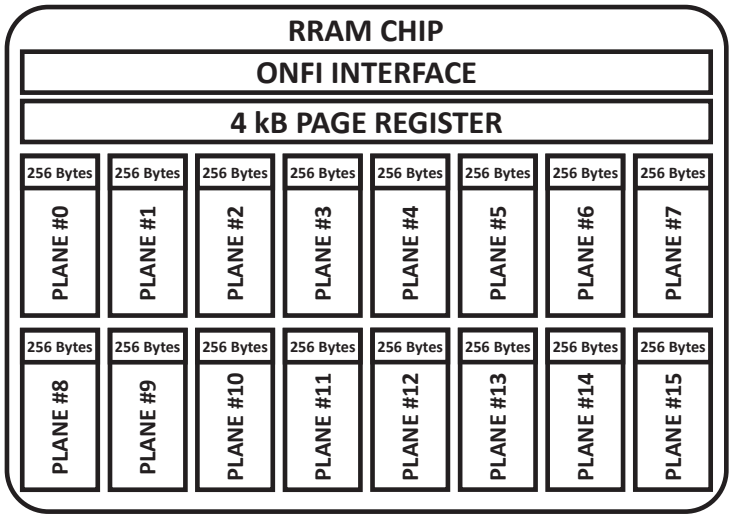


Figure 3.1: 32 Gbits RRAM memory module architecture.

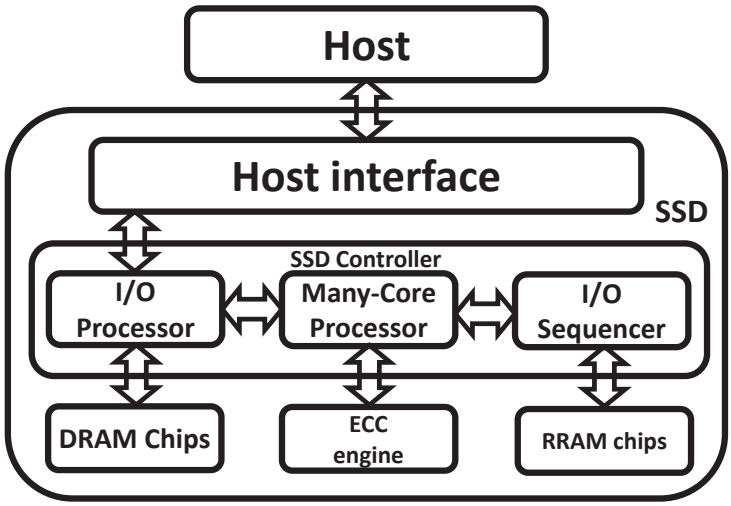


Figure 3.2: Block diagram of the simulated "All-RRAM" 512 GBytes SSD architecture.

3.2 Page size VS queue depth

A thorough statistical assessment of the "All-RRAM" SSD read latency and bandwidth figures is provided by simulating 500k random read operations with different page sizes used in the RRAM devices and a variety of command queue depths submitted to the SSD. As shown in Figs.3.3 and 3.4, by setting a fixed queue depth of 16 read commands it is demonstrated that the read bandwidth of the SSD proportionally increases with the memory page size, while the latency remains almost constant. A different behavior is observed when the RRAM page size is fixed and the read commands queue depth is varied from 1 to 32. The bandwidth increases proportionally with the queue depth until a saturation trend, dependent on the RRAM page size, is reached (see Fig.3.5). Concerning the latency, it is possible to observe a similar trend except that the saturation trend is observed at lower queue depth values (see Fig.3.6).

In SSD architectures, especially those foreseen in enterprise environments, it is important to evaluate possible losses of the Quality of Service (QoS), that are perceived as long response times of the disk when read operations are submitted by a host [46]. By evaluating in depth the cumulative distribution function (CDF) and the probability density function (PDF) of the latency figures for the "All-RRAM" SSD it is shown that when user transactions matches the RRAM chip page size (i.e., 256 Bytes) and only one operation is served at a time it is achieved an extremely low latency, measured in terms of tens of microseconds as shown in Fig. 3.7. The highest read response time (i.e., 99.99 percentile of the CDF) is around 16 μs, that is well below the few hundreds of microseconds offered by NAND Flash-based SSDs. However such a queue depth and RRAM page size do not reflect the workload conditions of nowadays host platforms and file-systems which are designed to issue multiple-outstanding read operations with a fixed payload of 4 kBytes. With this respect, as shown in Fig. 3.8, when the host interface queue depth is fixed to 32 commands and the user operations match the native 256 Bytes RRAM addressing mode, the median latency rapidly increases up to 66 μs. Eventually the read response times of the "All-RRAM" SSD match those of a simulated 1X-MLC NAND Flash-based SSD when a 4 kBytes RRAM page size is devised. Indeed, as it can be seen in Fig. 3.9, as soon as the the "All-RRAM" SSD is stimulated to operate in state-of-the-art working conditions, its read responsiveness becomes comparable to that of a NAND Flash-based SSD and hence the RRAM advantages are vanished.

To clarify the roots of the former results it has been decided to asses the actual SSD resources exploitation in terms of the percentage of RRAM I/O bus interface use and the percentage of active RRAM dies both calculated

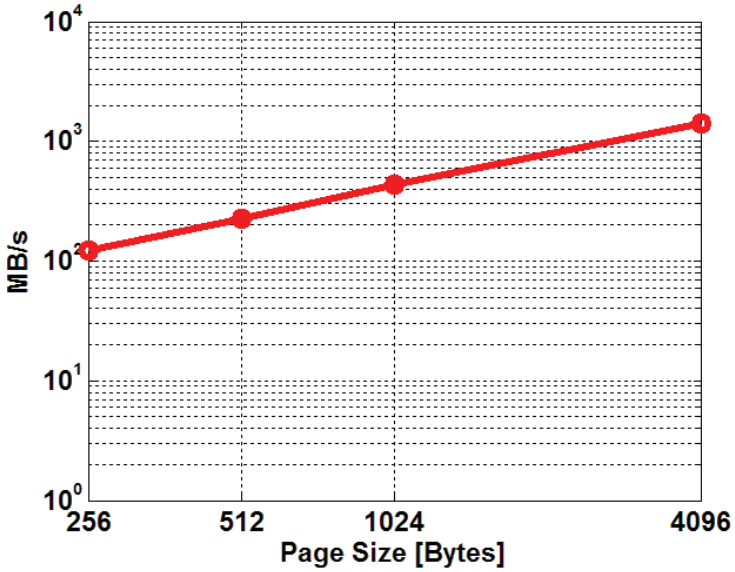


Figure 3.3: Simulated SSD average read bandwidth as a function of the RRAM page size when a queue depth of 16 commands is fixed.

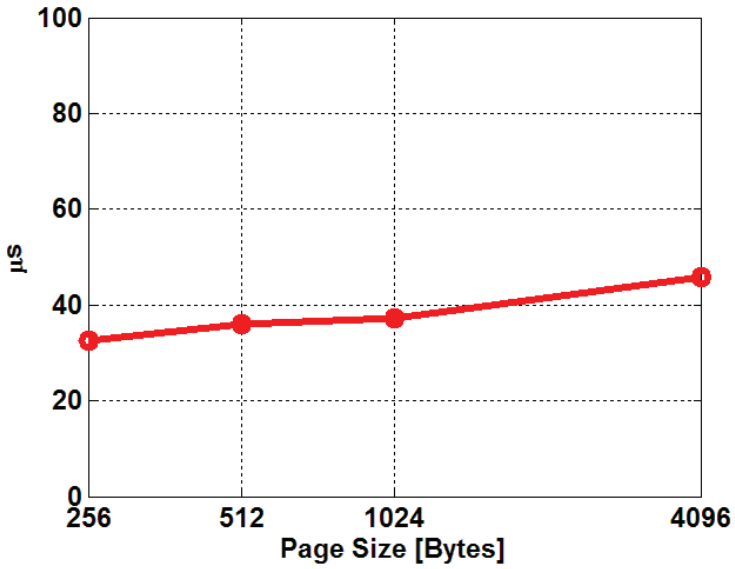


Figure 3.4: Simulated SSD average read latency as a function of the RRAM page size when a queue depth of 16 commands is fixed.

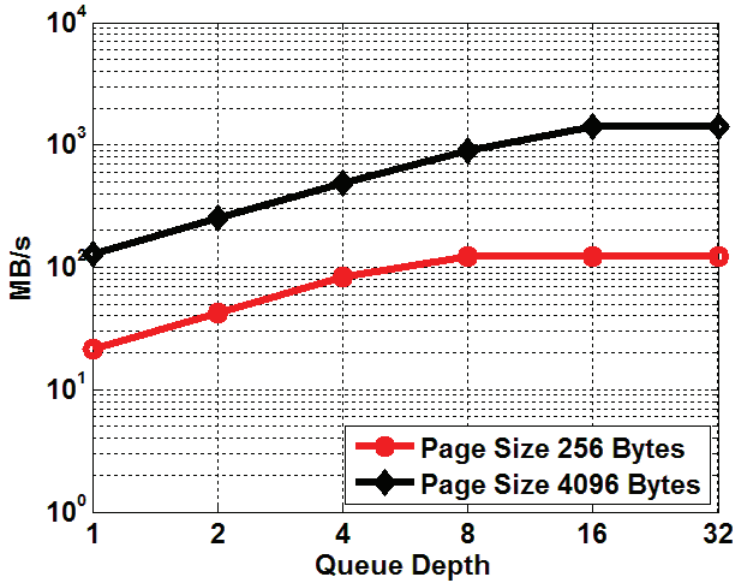


Figure 3.5: Simulated SSD average read bandwidth as a function of the host interface queue depth. Native 256 Bytes and 4 kBytes multi-plane RRAM addressing modes are considered.

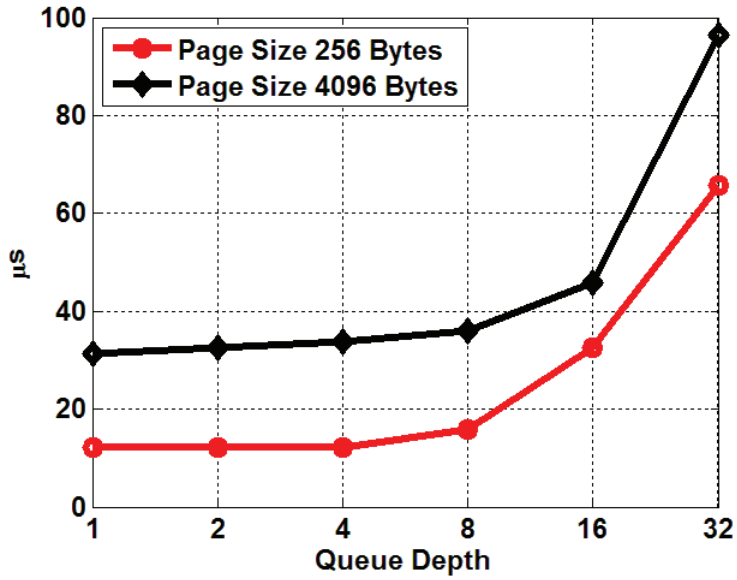
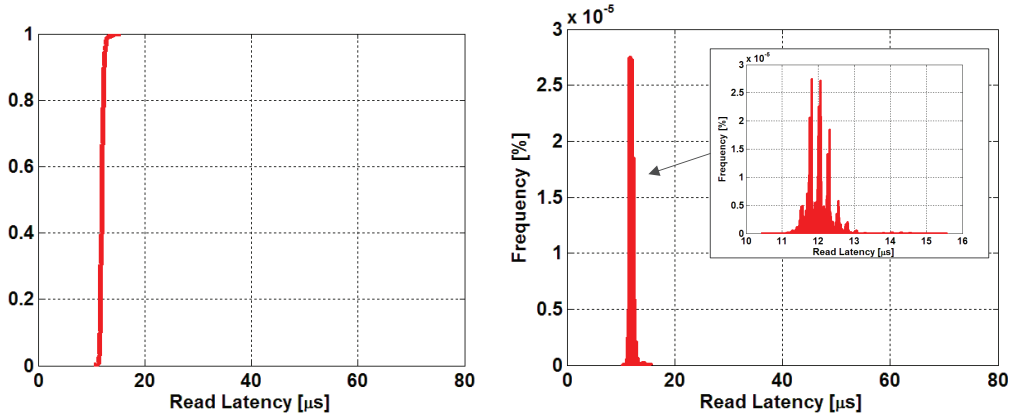
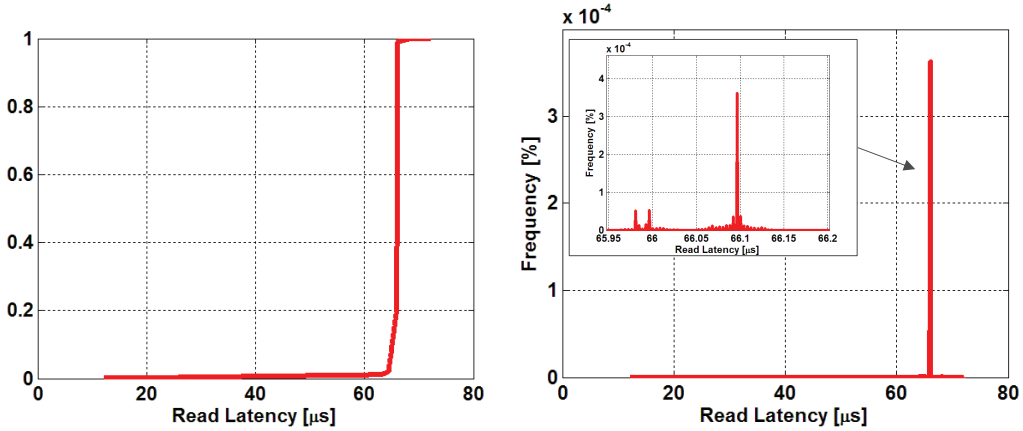


Figure 3.6: Simulated SSD average read latency as a function of the host interface queue depth. Native 256 Bytes and 4 kBytes multi-plane RRAM addressing modes are considered.



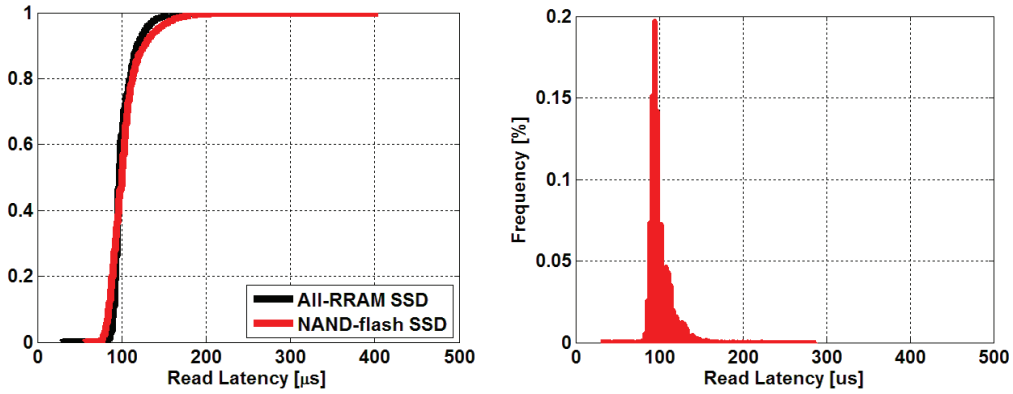
(a) (b)

Figure 3.7: Cumulative distribution function (a) and probability density function (b) of the simulated SSD read latency when a queue depth of one command is used with the native 256 Bytes RRAM page size.



(a) (b)

Figure 3.8: Cumulative distribution function (a) and probability density function (b) of the simulated SSD read latency when a queue depth of 32 commands is used with the native 256 Bytes RRAM page size.



(a) (b)

Figure 3.9: Cumulative distribution function (a) and probability density function (b) of the simulated SSD read latency when a queue depth of 32 commands is used with the emulated 4 kBytes RRAM page size. A comparison with a state-of-the-art NAND Flash SSD is provided.

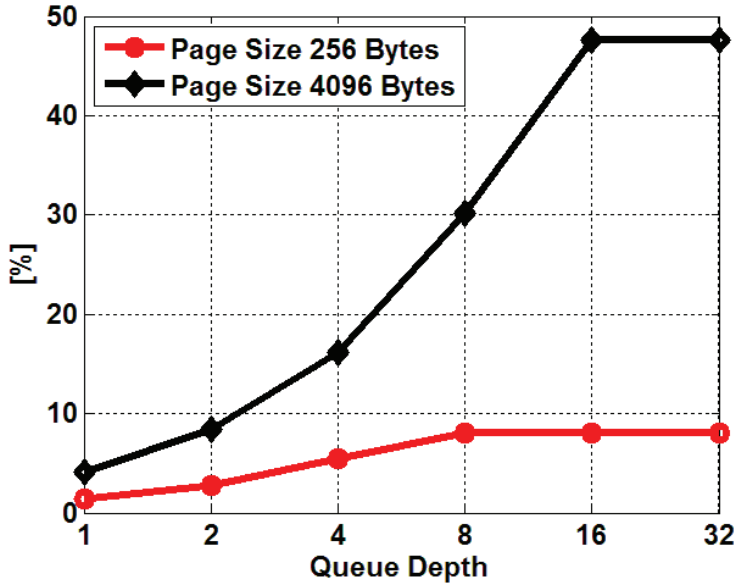


Figure 3.10: Average RRAM I/O bus interface use as a function of the host interface queue depth. Native 256 Bytes and 4 kBytes multi-plane RRAM addressing modes are considered.

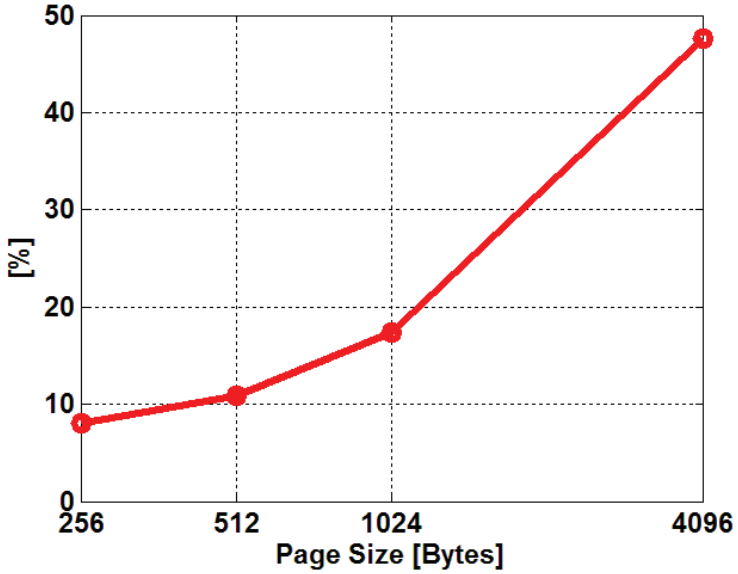


Figure 3.11: Average RRAM I/O bus interface use as a function of the envisioned RRAM page size when a queue depth of 16 commands is fixed.

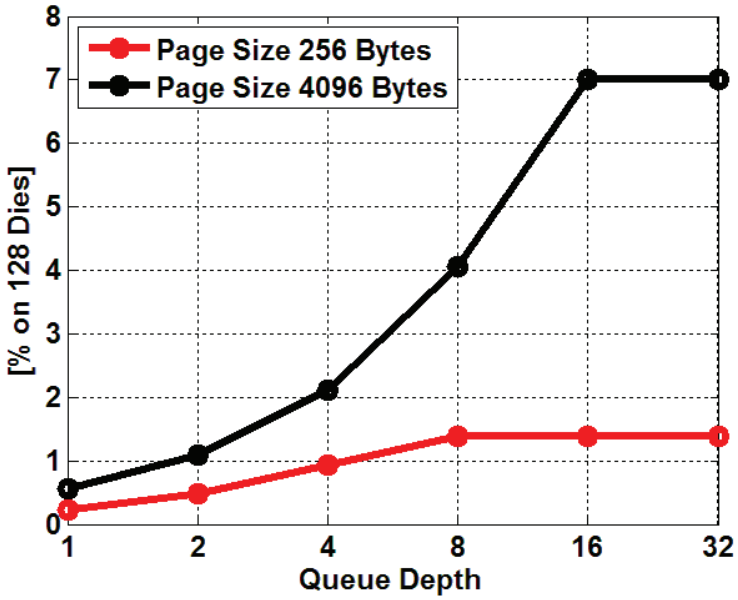


Figure 3.12: Percentage of active RRAM dies as a function of the host interface queue depth. Native 256 Bytes and 4 kBytes multi-plane RRAM addressing modes are considered.

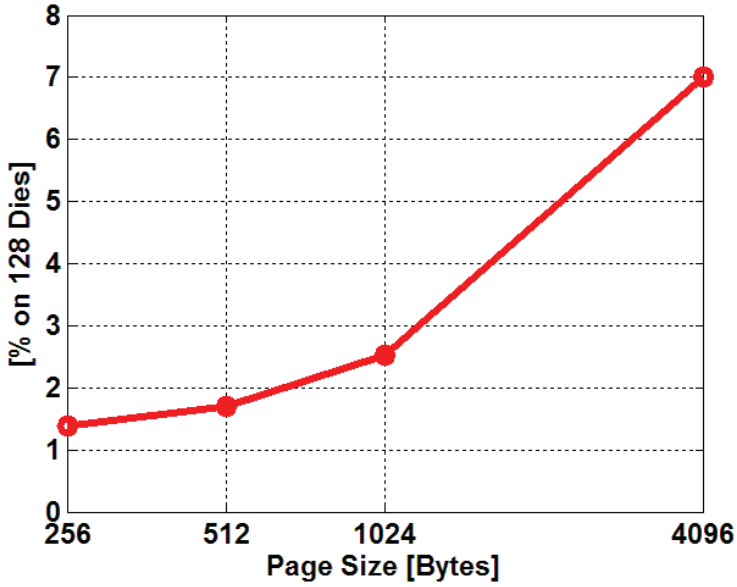


Figure 3.13: Percentage of active RRAM dies as a function of the envisioned RRAM page size when a queue depth of 16 commands is fixed.

as a function of the envisioned RRAM page size and disk commands queue depth. As the read transactions payload increases and the queue depth is large enough to serve multiple commands more data have to be transferred from the memories to the SSD controller. This yields to a massive overhead in terms of data transfers impacting the percentage of the I/O memory bus use, which rapidly grows up reaching 48% when 4 kBytes transactions are served with a queue depth equal to 16, as it can be seen in Figs. 3.10 and 3.11. As a consequence the overall SSD latency is impacted and hence the performance advantages introduced by RRAM memories become less perceivable. Another important consideration can be drawn by observing the average percentage of active RRAM dies in the SSD considering a 100% random read workload. As it can be seen in Figs. 3.12 and 3.13, even when 4 kBytes transactions are envisioned by the host, its value remains far below 10%. These results clearly denote a high underuse of the SSD resources. In fact, as previously described in Section 3.1, the proposed "All-RRAM" SSD exploits a traditional controller designed for NAND flash memories and leverages only on the compatibility of RRAM memories with the NAND flash bus interface. This approach, on the first hand allows reducing "All-RRAM" SSDs costs since standard ip-cores can be reused, however, on the other hand does not permit to properly use the underlying storage system which is completely different from NAND flash memories.

3.3 Optimum design point exploration of "All-RRAM" SSDs

One of the main problems highlighted in the previous Section which heavily impact the actual "All-RRAM" SSD performance, is the memory I/O bus inefficiency. In fact, as shown in Figs. 3.14 and 3.15, when it is performed a latency breakdown of the storage layer considering a 200 MT/s I/O bus frequency, it is clear that, compared to NAND flash memories, when RRAMs are used the I/O bus transfer time is the dominant contributor.

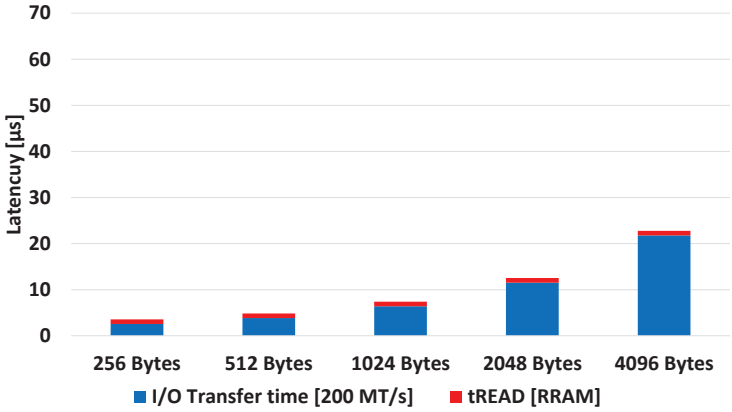


Figure 3.14: Storage latency breakdown when a RRAM and a 200 MT/s I/O bus are used.

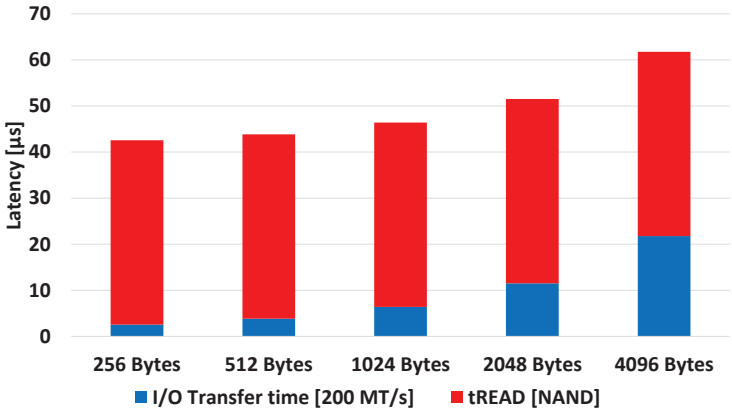


Figure 3.15: Storage latency breakdown when a 1X-MLC NAND flash memory and a 200 MT/s I/O bus are used.

The use of a 200 MT/s I/O bus rate, however, represents a forced choice in nowadays SSD architecture because it is able to guarantee an excellent signal

integrity over relatively long distances. In fact, due to placement constraints of SSD boards, the SSD controller and NAND flash packages are usually physically separated. Therefore, in order to avoid any additional noise on top of the NAND flash channel it is mandatory to exploit reduced I/O bus transfer rates.

Thanks to the advent of extremely fast storage media such as RRAMs, memory vendors are now investing a great effort to push the I/O interfaces to operate at extremely high frequencies. 800 MT/s I/O bus transfer rates have been successfully achieved with new ONFI protocols [79] which, however, to be exploited require a massive redesign of the analog circuitry of the SSD controller. To this extent, these protocols still not represent a standard for NAND flash based SSD, however, in the next future they will provide a path for a full exploitation of fast non-volatile storage media.

In order to understand how these new SSD controller generations could positively impact the performance of an "All-RRAM" SSD, a complete design space exploration has been performed considering a 800 MT/s I/O bus transfer rate and 5 different RRAM page sizes: 256 B, 512 B, 1 kB, 2 kB, and 4 kB. The bandwidth of the disk, the average latency and the Quality of Service (QoS) [46] have been retrieved for all the page size configurations. The bandwidth is the average number of read commands completed in a second; the average latency is the average time elapsed between a read command submission and its completion; the QoS has been calculated as the 99.99 percentile of the SSD's latency distribution. To provide a complete performance exploration of the SSD's architecture, data have been collected for different host queue depths (QD) ranging from 1 up to 32 commands [27].

NAND-like mode: 4 kB RRAM page size

This case study represents the mere replacement of a NAND Flash memory with a RRAM chip in a user-transparent mode and will be the baseline for the following results. Therefore, as already presented in Section 3.2, in order to provide a full compatibility with NAND Flash dies, the RRAM dies must operate in a 16-planes mode.

As it can be seen in Figs. 3.16 and 3.17 (dashed lines), the average latency, the QoS, and the bandwidth increase with QD. In particular, the bandwidth of the SSD reaches a saturation trend when a QD equal to 8 commands is used, denoting that the SSD controller has reached its maximum performance.

As demonstrated in Fig. 3.14, the average read latency of the storage layer of an SSD, depends on two contributors: the memory T_{READ} and the byte-wise I/O transfer time taken to move the read data from the memory to

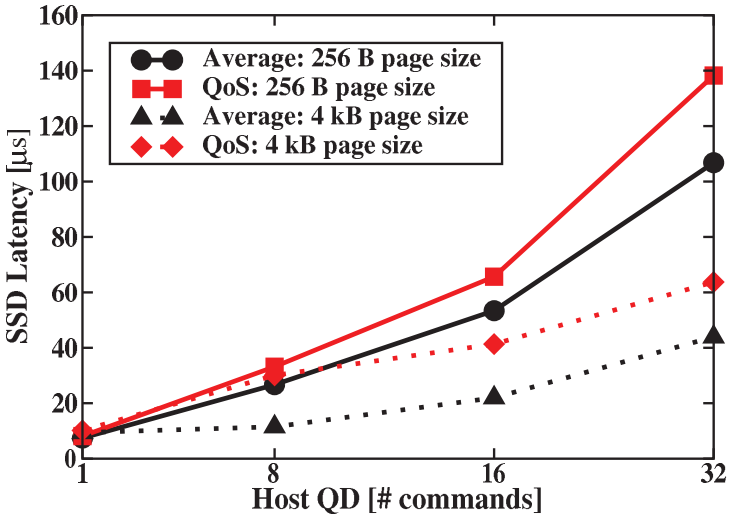


Figure 3.16: Average latency and QoS of the simulated "All-RRAM" SSD when a page size of 4 kB and 256 B is used, respectively.

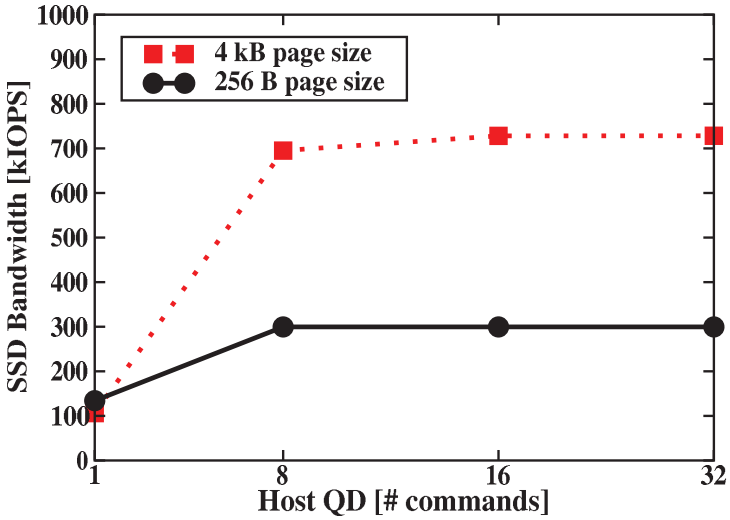


Figure 3.17: Average latency and QoS of the simulated "All-RRAM" SSD when a page size of 4 kB and 256 B is used, respectively.

the SSD controller. As shown in Fig. 3.16, by considering QD = 1 (i.e., one command issued at a time), it is possible to observe an average SSD latency of 9.4 μs which is almost 4 times smaller than the one observed in Fig. 3.6. This improvement is mainly due to the lower I/O bus transfer time taken by the 800 MT/s memory interface. In fact, in this case the transfer of a 4 kB page takes only 6 μs instead of the 21 μs taken by the traditional 200 MT/s interface. However, it must be highlighted that compared to the memory T_{READ} time, the I/O bus transfer contributor still dominates the overall SSD latency.

Although the obtained latency is far below the typical values of NAND-based SSDs [76], RRAMs can be extensively optimized to reach even higher performance. In particular, the optimization must concern the partitioning of the 4 kB transactions imposed by the host interface into smaller chunks. The aim of this methodology is to reduce the data transfer time individuating the number of parallel multi-planes to be simultaneously accessed (i.e., optimal memory page size) and the number of internal read commands to be handled by the SSD firmware.

Single-plane 256 B RRAM page size

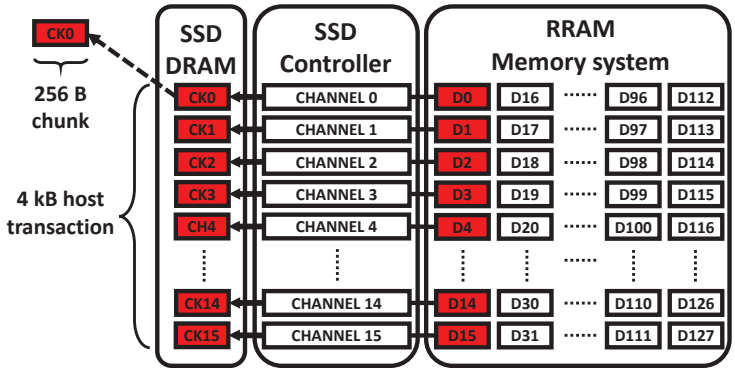


Figure 3.18: Translation algorithm performed by the SSD. Each 4 kB host transaction is split across the SSD's channels.

The minimum read granularity allowed by RRAMs is a single plane 256 B page read operation which, if used, should reduce the transfer time and hence also the SSD latency. However, as shown in Fig. 3.18, since the host works with 4 kB transactions, it is necessary to split the host operations in 16 chunks of 256 B each. This operation is performed by the SSD's firmware, which, by using the 16 channels architecture and the DRAM buffer, reads

each 256 B chunk in parallel and rebuild the 4 kB transaction before sending the data back to the host.

Figs. 3.16 and 3.17 (solid lines) show the bandwidth, the average latency and the QoS achieved by the aforementioned approach. By considering the simulations performed with QD = 1 the straightforward conclusion could be that the 256 B page size will reduce SSD latency, increase the bandwidth, and improve the QoS.

However, for QD > 1 these considerations will not hold true since the number of operations internally handled by the SSD controller increases by a factor 16, leading to a saturation of its processing capabilities. This turns into a dramatic performance loss aggravated by the fact that all data chunks must be temporarily stored inside the DRAM buffer, whose access is contended by all the SSD channels causing resources starvation.

Multi-planes 512-1024-2048 B RRAM page sizes

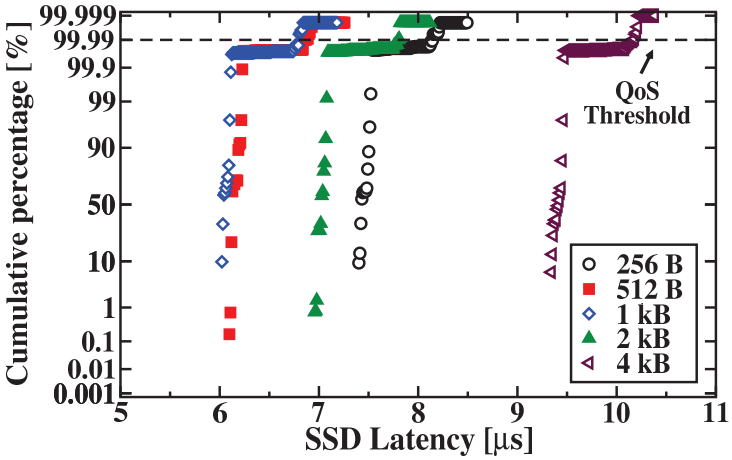


Figure 3.19: Cumulative percentage on a normal probability paper of the latency of the "All-RRAM" SSD when a QD = 1 and a page size of 256 B, 512 B, 1 kB, 2 kB, and 4 kB is considered. The QoS threshold is marked for all the tested cases.

To reduce both the amount of commands processed by the SSD controller and the number of accesses to the internal DRAM memory, different RRAM page sizes have been considered: 512 B, 1 kB, and 2 kB. To exploit these configurations and keep unaltered the 4 kB host transactions payload, both the SSD's firmware and the RRAM memories have been co-designed to work in multi-plane mode. With this respect, when a n*256 B RRAM page size

is used, being n = [2, 4, 8], the SSD's firmware is configured to read 16/n chunks of n*256 B each from 16/n parallel channels.

Fig. 3.19 shows the cumulative latency distributions and the QoS of the simulated "All-RRAM" SSD as a function of the page size, when a host QD = 1 is selected. As it can be seen, the optimum disk latency is achieved neither with the standard 256 B page size nor with the 4 kB NAND-like mode, but rather with a 1 kB multi-plane page configuration. This phenomenon is manifested also at different host QD as shown in Figs. 3.20 a-d, where the disk latencies evidence two page size optima: for QD $<$ 8 the value is 1 kB, whereas it becomes 2 kB for QD $>$ 8.

Fig. 3.21 shows the bandwidth achieved by the considered "All-RRAM" SSD as a function of both the RRAM page size and the host QD. As it can be seen, when QD = 1, the maximum bandwidth is achieved when a page size of 1 kB is used, whereas for QD $>$ 16 the maximum bandwidth is achieved for 4 kB. However, if it is considered that the optimization of the SSD must simultaneously concern all the figures of merit, it appears that a good trade-off between bandwidth, latency, and QoS still exists for 1 kB and 2 kB page sizes.

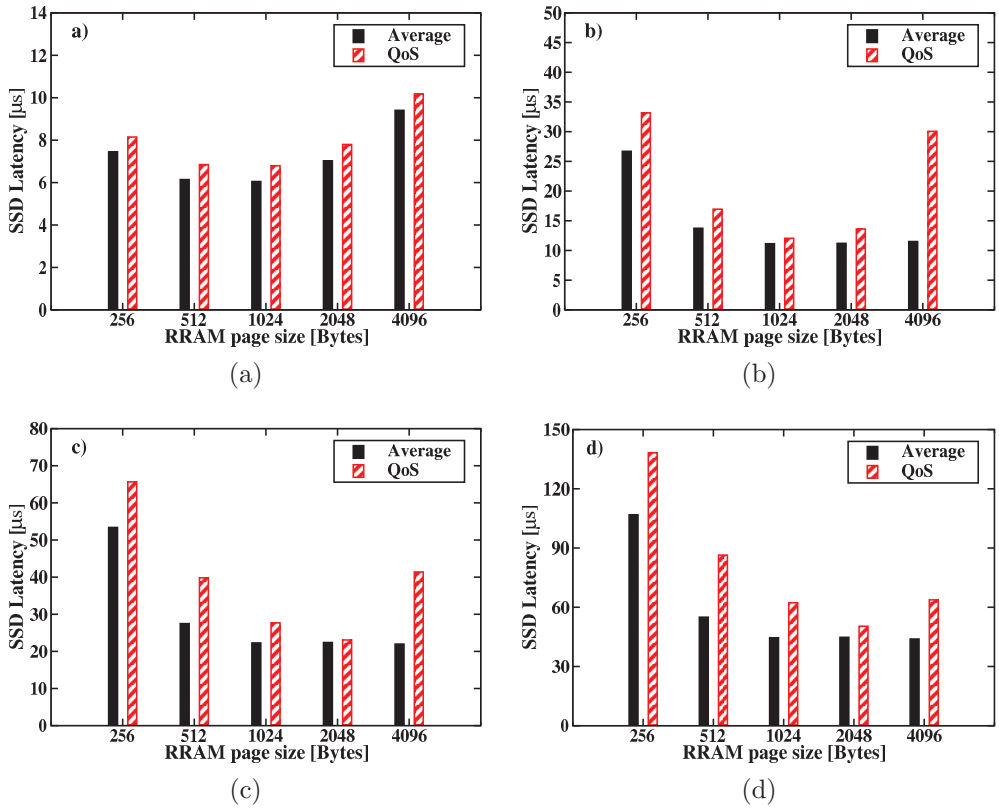


Figure 3.20: SSD average latency and QoS when all the different RRAM page sizes are used and a host QD = 1 (a), 8 (b), 16 (c), and 32 (d) is considered.

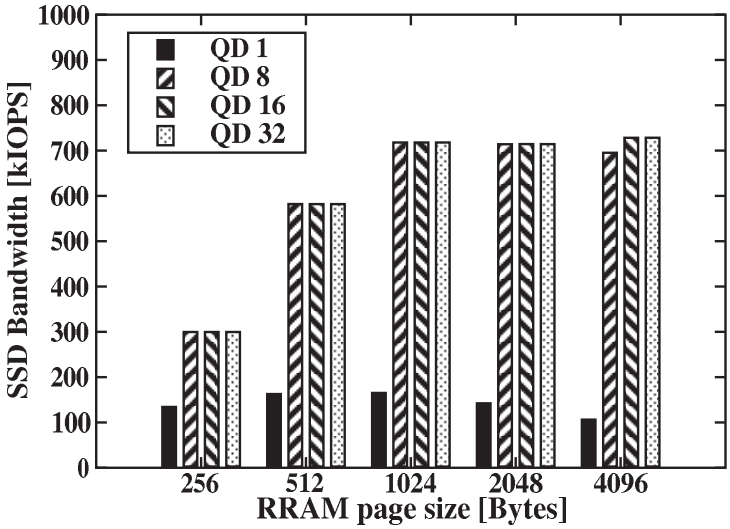


Figure 3.21: SSD bandwidth as a function of the RRAM page size and the host QD.

Chapter 4

Next steps: power efficient SSD architectures and beyond

The increasing use of SSDs in hyperscale environments such as cloud computing and big-data servers is now pushing the storage revolution made available by solid-state non-volatile memories to a new edge. With this respect, the new constraint that now drives the design phase of an SSDs is the power consumption, which limits the actual storage capacity of these devices. Achieving an ExaBytes-dense (10^{18} Bytes) disk completely constructed from NAND flash memories, is the aim of "All-Flash" arrays which are now of great interest both in the industry and in the academia [80, 81, 82, 83, 84, 85]. Basically, they aim to completely replace traditional HDDs in order to enable a fast path also for rarely used data called "cold-storage". However, to pursue this goal, beside the traditional power optimization performed inside SSD controller it is mandatory to start to consider the storage infrastructure, namely, the system composed by the user's application and the actual storage device, as a whole. In fact, putting all the effort to achieve a high dense and power efficient SSD able to work for general purpose workloads, is now becoming an extremely complex task because first and foremost the NAND flash memories cannot be used in this way. As a matter of example, as shown in Section 2.3.2, even if TLC NAND flash memories are able to provide a big storage capacity, they cannot be used for write intensive workloads because of the reduced endurance rate and the high power consumption. To clarify this latter consideration it is sufficient to consider that TLC memories show a higher program time compared to MLC an SLC storage paradigm, and hence they will introduce a higher power consumption. As a consequence it is clear that user's applications have to be carefully co-designed with the underlying storage system considering not only either the target performance or the power consumption, but rather the combination of both. This approach is leading

to a new design methodology of SSD architectures called "software-defined SSDs". This methodology tries to connect the application development step with the SSD design phase, and to build a custom power-efficient and high-performing storage environment thought for a specific workload.

Along with the emerging power-efficient SSD design methodologies made available by the "software-defined" approach, it is now becoming of great interest a lightweight storage architecture on which the FTL and the all the routines responsible for the NAND flash memories management are moved from the SSD controller to a high-performing host processor such as a Multi-Purpose Processing Arrays (MPPA). The motivation for moving certain responsibilities from the SSD controller to a more powerful host processor [86] is to make I/O data commands predictable from the host-side. In this way, the device enables the host to adapt the FTL algorithms and optimizations to match the appropriate user workloads that it executes. This approach, called "Open-Channel SSDs" [87], enhance the previously mentioned "software-defined" paradigm introducing a dynamic evolution or modification of the SSD's characteristics and enabling the design of extremely power-efficient and high-performing SSD architectures.

In this chapter are presented two simulation methodologies that can be used to:

- assess the power consumption of a specific SSD architecture;
- efficiently co-design the resources of a "Open-Channel" storage architecture;

The following discussions have to be intended as a possible extension of the results presented in this thesis. With this respect, the main goal of this chapter is to show in how many parts the SSD design space can be divided, and to highlight the increasing necessity to contextualize the design phase of an SSD inside the application scenario where it will be used.

4.1 Assessing SSDs' power consumption with SSDPower

As presented in Chapter 2, NAND-Flash based SSDs feature an embedded controller and a parallel array of NAND Flash chips, which, if properly designed allows achieving high throughput, low read/write latency and high reliability. However, to deal with the increasing performance requirements of hyperscale systems such as cloud computing and big data servers, SSD designers started to leverage on memory I/O parallelism embodying on the

same system hundreds of parallel NAND flash dies. This approach, as shown in Section 2.1, leads to extremely fast yet complex solutions, which on the first hand allow achieving the target performance/capacity of the disk, but on the other hand heavily impact the overall SSD power consumption. This latter point is of a particular concern in cloud environments due to restricted power budget and the severe green policies [88]. To overcome the SSD's power consumption issue and to better understand which are the main actors responsible for power consumption, several approaches have been proposed. Among them two techniques arise: the first one is a top-down approach which leverages on a real hardware setup able to measure the SSD's power consumption during its functional behavior [89, 90]; the second one is a simulation-driven bottom-up approach which assesses the disk power consumption from that of NAND flash memories [91]. However, a closer look to these methodologies reveals that both have several drawbacks. From the SSD's designer perspective, measuring the disk power when its design is already completed does not provide any useful information. In the same way, simulating the NAND flash power figures turns out to be inaccurate when internal program and read algorithms are not disclosed and hence, any power estimation could heavily differs from that measured on real silicon implementations. At this point, it is clear that assessing the disk power consumption during its design phase would be a desirable feature which could help designers to efficiently define the disk architecture thus avoiding any resource misallocation. In this Section, it is proposed SSDPower a co-simulation framework for SSD power consumption estimation. The tool exploits a mixed-bottom up approach that provides SSD's power consumption figures starting from measured NAND flash current traces. SSDPower allows broadening the design space exploration of an SSD relating the disk architecture and the host workload with the NAND flash current profiles, thus helping the estimation and the optimization of the whole NAND flash sub-system power consumption.

4.1.1 SSDPower Rationale

The proposed SSDPower framework bases on the measurements of the NAND flash chips power consumption through the analysis of their current drawn during read and write operations, and the SSDExplorer simulator. The main idea is to exploit the Kirchhoff's current law to assess the power consumption of the NAND flash sub-system of a SSD. With this respect, since NAND flash chips are connected to the same power supply, the total current drawn by the system will be the sum of each NAND flash chip contribution. Fig. 4.1 shows a measurement taken with an oscilloscope proving this assumption on a real SSD product. In this case, the SSD controller issued two read

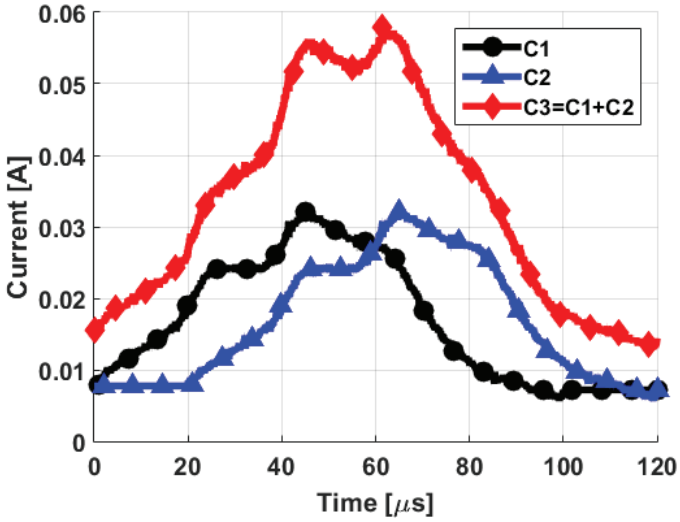


Figure 4.1: Superposition principle measured during standard NAND flash read operations. C1 and C2 are the current measured on single chips while C3 is the current measured on the main power supply.

commands with a 20 μs skew on two different NAND flash chips connected to the same power supply; C1 and C2 are the currents measured on the two chips while C3 is the current measured on the main power supply. As it can be seen, C3 is the exact sum of the currents C1 and C2, and this proves definitively the possibility to exploit the superposition effect also for NAND flash power consumption estimations. At this point, in order to estimate the power consumption of the whole NAND flash sub-system of an SSD it is sufficient to:

- measure the current drawn by a single chip during standard read and program operations;
- monitor the memories' ready/busy switching activity;
- superimpose the current waveforms of the memories to their switching activity and sum all the contributions.

Fig. 4.2 shows the baseline setup exploited for the NAND Flash current characterization step. As it can be seen, a V/I converter [92] has been connected in parallel to a 300 $m\omega$ resistor (R_SENSE) sensing the voltage drop between its terminals produced by the current I_LOAD. The resistor is connected in series to the VCC power supply of a standard SO-DIMM test board where four different NAND flash chips are mounted (see Fig.

Table 4.1: Main characteristics of the measured NAND flash memories.

VCC	3.3 V
Technology node	Mid-1X
Storage paradigm	TLC-128Gb
Interface BUS	Toggle
BUS speed	400 MT/s
Average TPROG (lower page)	800 μs
Average TPROG (middle page)	1900 μs
Average TPROG (upper page)	4300 μs
Average TREAD (lower page)	80 μs
Average TREAD (middle page)	100 μs
Average TREAD (upper page)	80 μs

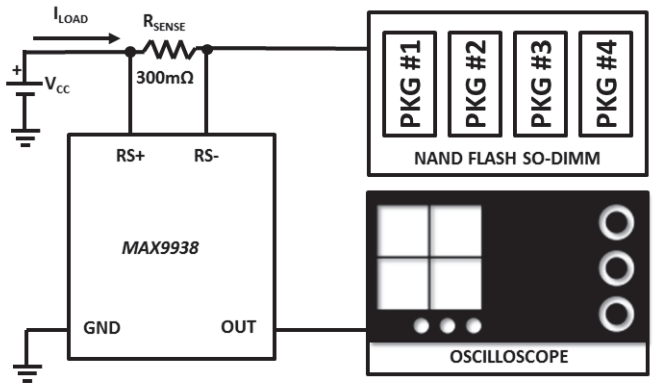


Figure 4.2: Schematic of the measurement setup used to characterize NAND Flash power consumption.

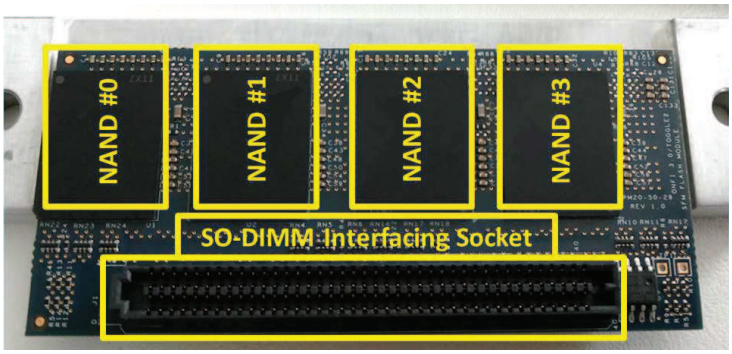


Figure 4.3: TLC NAND Flash SO-DIMM test board.

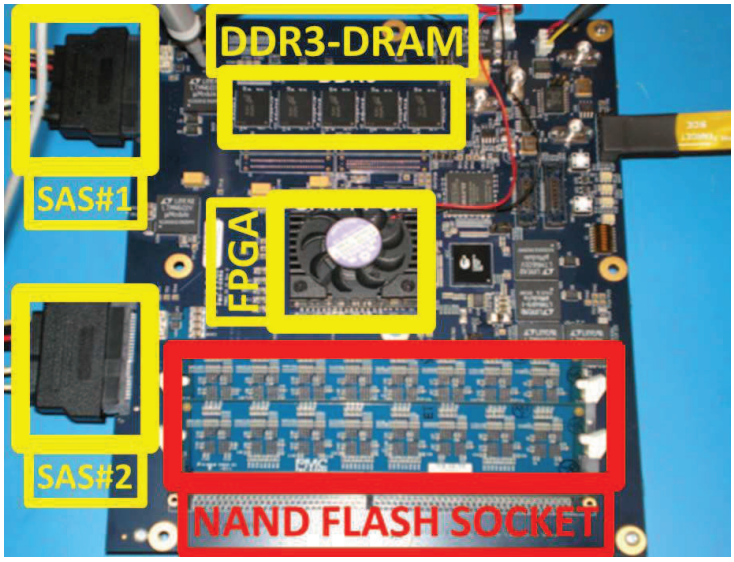


Figure 4.4: ATE used for NAND flash memories characterization.

4.3). Both the sense resistor value and the current-sense amplifier gain have been wisely selected to produce enough response bandwidth to capture all the current fluctuations of NAND Flash memories' operations. Finally, the output current of the V/I converter has been measured by an oscilloscope. Table. 4.1 summarizes the characteristics of the tested memories. The test board communicates through the interfacing socket with a dedicated Automated Test Equipment (ATE) shown in Fig. 4.4. The ATE is composed by a programmable FPGA, a DRAM-buffer for data manipulation, two SO-DIMM card sockets and two SAS ports for PC connection. The FPGA is programmed to behave as a simple NAND flash controller that takes user commands as inputs and issues standard NAND flash read and program operations to all the chips. Figs. 4.5 show the result of the NAND flash current characterization. As it can be seen, thanks to the exploited setup it has been possible monitoring the actual latency and all the current peaks introduced by read and program operations. Especially for this latter, the oscilloscope measurement allowed capturing the current drawn during the Incremental Step Pulse Program algorithm, which represent an extremely useful information when SSD power optimization algorithms have to be studied.

The NAND flash power measurement setup has been designed only for memories characterization and power consumption assessment. With this respect, in fact, the test equipment can only send raw IO/s to memories under test and it is neither capable to execute a complete SSD firmware nor to handle the host interface protocol. Moreover, since it does not have

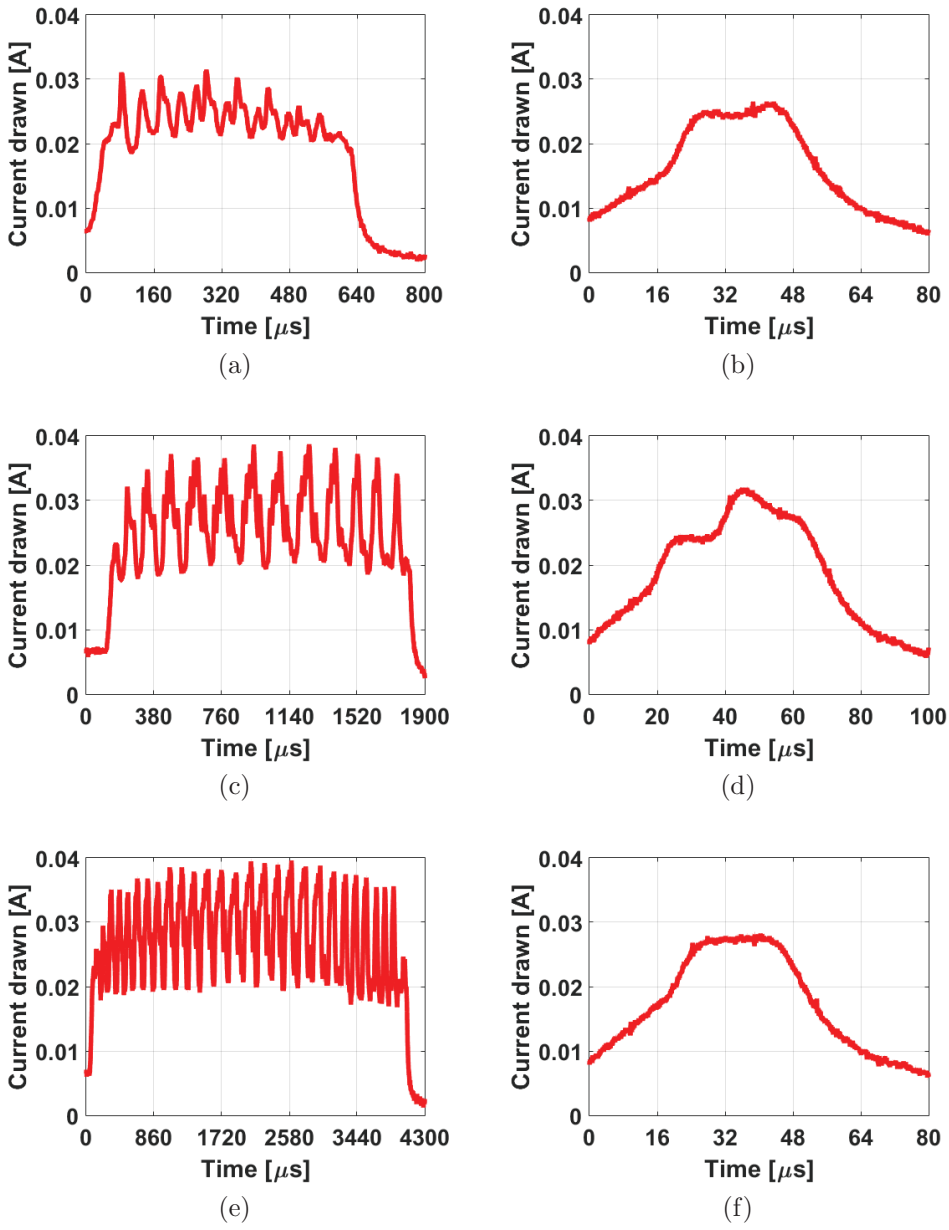


Figure 4.5: Current measured during program and read operations in a single NAND flash chip. a) lower page program; b) lower page read; c) middle page program; d) middle page read; e) upper page program; f) upper page read.

any notion about the SSD architecture, it lacks of an intrinsic flexibility and hence, it is not possible to explore any specific configuration without a new hardware implementation. In this Section, this problem has been addressed exploiting a SSDExplorer. The tool, in fact, is also able to monitor all the operations sent by the controller to the NAND flash memory array and to dynamically compute its ready/busy switching activity depending on the selected architecture and workload. This latter feature makes the tool the suitable solution for the final SSDPower implementation.

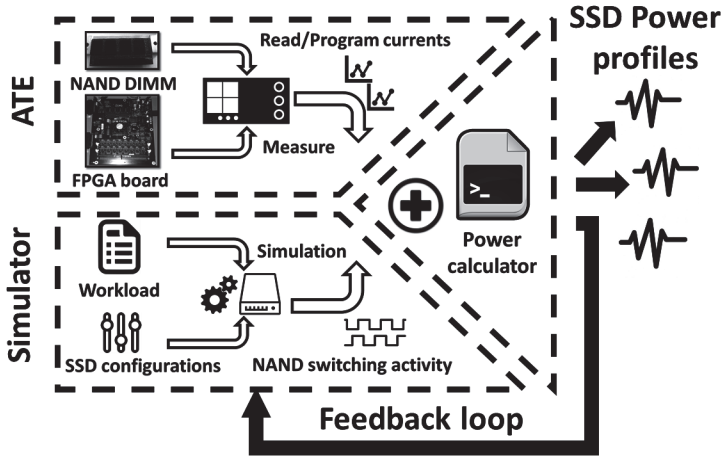


Figure 4.6: SSDPower test flow.

Fig. 4.6 summarizes the main steps performed by SSDPower. Basically, the power traces gathered by the measurement setup are stored in a numeric format and directly sent to the power calculator engine. Along with the current profiles of NAND flash operations, the SSD simulator injects the memories' ready/busy switching activity calculated for a given disk architecture and a host workload. These contributions are then combined by the power calculator engine which computes the final power profile exploiting the superimposition phenomenon. Thank to these features, SSDPower allows connecting in a single automated test flow the design-space-exploration of latency and performance of a target SSD architecture along with the power consumption of the whole NAND flash memory subsystem. With this respect it must highlighted that the proposed approach is here applied to calculate the power consumption of the NAND flash subsystem, however, it can be applied to all the blocks of the SSD (Error Correction Code, processor, DRAM, etc.) given the corresponding power traces.

One of the main features of SSDPower is that it can be used in a feedback loop with the SSD simulator assessing the power consumption of a tar-

get architecture and dynamically modifying the internal command sequence processed by the SSD. This iterative process can be used in an attempt to optimize the SSD power profile. With this respect, uncontrolled multiple program operations issued on parallel chips may introduce an out-of-spec current consumption representing one of the main problems that limits the internal parallelism, and therefore the performance of an SSD architecture [93].

Table 4.2: Test performed on the tested SSD architecture.

Workload	Test	Power optimization
25% Program 75% Read 512 Mbytes 4 kBytes 100% random	T1	-
	T2	Single program suspend
	T3	Double program suspend

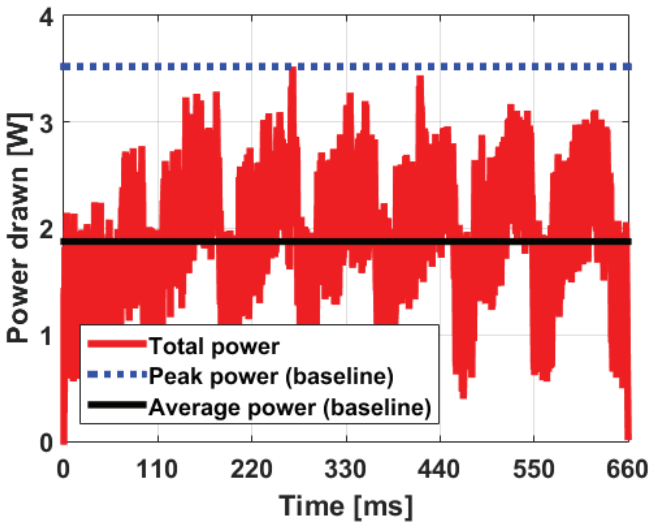


Figure 4.7: Total NAND flash subsystem power profile in the T1 case (baseline).

4.1.2 SSD power consumption optimization

In order to prove the capabilities of SSDPower, it has been simulated the power consumption of an SSD configured with 16 populated channels with 8 NAND flash targets each [8] in the test configurations shown in Table. 4.2. Fig. 4.7 shows the power consumed by the NAND flash subsystem when

the test T1 is considered. In this case, the average and the peak power consumption are 1.87 W and 3.52 W, respectively, whereas the time required to process the workload is 660 ms. As it can be seen, even if random operations are issued by the host system, the power profile is composed by a repetitive sequence of power fluctuations. This is an expected behavior because random program operations are serialized by the SSD controller to deal with the standard in-order programming sequence of NAND flash memories. After this test, which represents the baseline configuration considered in this work, the tool has been used to study a dedicated peak power management algorithm able to optimize the power profile of multiple program operations issued on parallel chips. The power consumption problem has been addressed exploiting a power throttling algorithm based on program suspend operations [94]. Basically, depending on the power profile drawn by the disk, the simulated SSD controller can introduce a variable amount of suspend windows inside any program operations so that to disoverlap current peaks on parallel chips. Moreover, in order to reduce also the average power consumption of the disk, during program suspensions no other operations have been issued to idle chips. Finally, considering that the program of an upper page is the most power hungry operation (according to Fig. 4.5e), the peak power management algorithm has been applied only to upper page programming. In the test T2 a single suspend window of 200 μs has been issued by the ATE setup during upper page programming; then the suspend window has been shifted on all possible suspension points made available by the internal NAND flash logic circuitry.

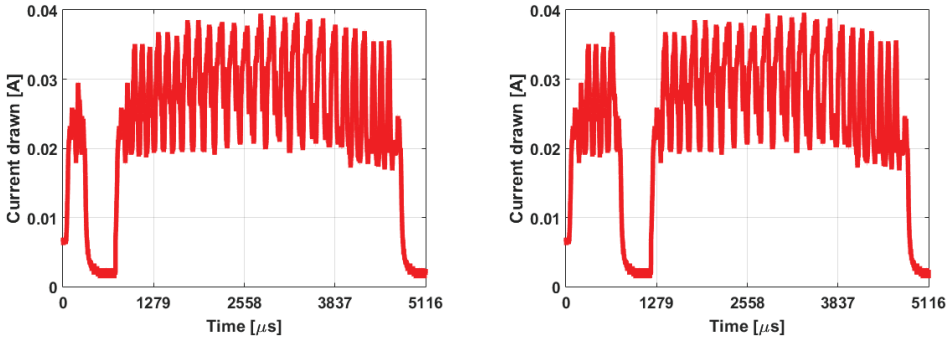
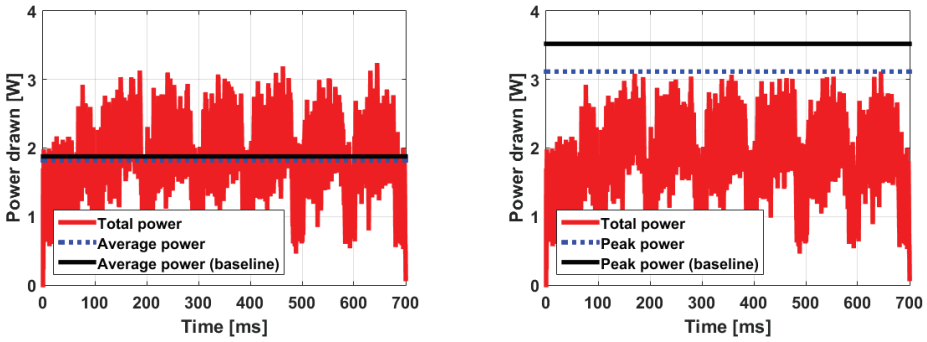


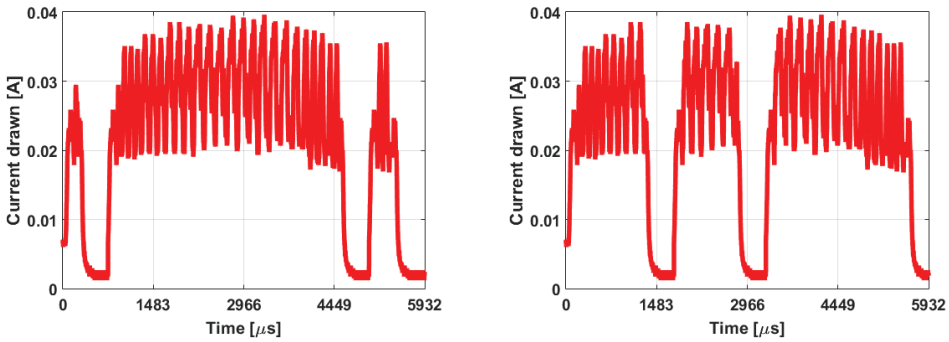
Figure 4.8: Current measured during single program suspend operations with optimized suspend window for average a) and peak b) power reduction.

The current profiles optimizing either the average or the peak power consumption of the NAND flash subsystem are shown in Fig. 4.8a)-b), respec-



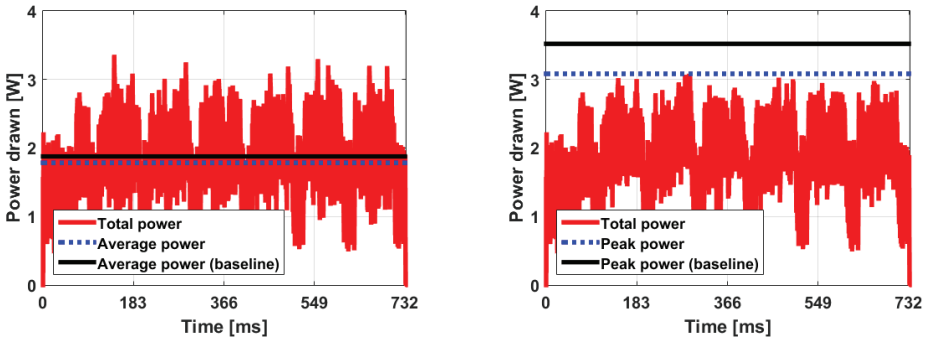
(a) (b)

Figure 4.9: Total NAND flash subsystem power profile in the T2 case when average power a) or peak power b) are optimized.



(a) (b)

Figure 4.10: Current measured during double program suspend operations with optimized suspend windows for average a) and peak b) power reduction.



(a) (b)

Figure 4.11: Total NAND flash subsystem power profile in the T3 case when average power a) or peak power b) are optimized.

tively. Fig. 4.9 shows the total power consumed when either the average a) or the peak power b) are optimized. With respect to the baseline Test T1, the average and the peak power of the NAND flash subsystem are reduced of about 3.5% and 11.5%, respectively. As expected, however, the proposed approach negatively impacted the total processing time of the disk which increases of about 5%. This phenomenon is a direct consequence of the suspension window applied on upper pages, which increased their latency of about 16%. In the test T3 two suspend windows of 200 μs have been issued by the ATE setup during upper page programming and shifted on all possible suspension points. The current profiles optimizing either the average or the peak power consumption of the NAND flash subsystem are shown in Fig. 4.10a)-b), respectively. Fig. 4.11 shows the total power consumed when either the average a) or the peak power b) are optimized. With respect to the baseline Test T1, the average and the peak power of the NAND flash subsystem are reduced of about 5% and 13%, respectively. As expected, however, the proposed approach negatively impacted the total processing time of the disk which increases of about 10%. This phenomenon is a direct consequence of the suspension windows applied on upper pages, which increased their latency of about 27%. The comparison between tests T1, T2 and T3 evidenced that the peak power management algorithm trades a reduction of either the average or the peak power of the NAND flash memory subsystem, with the total SSD command processing time.

4.2 Accelerating data-intensive applications with MPPAs and SSDs

To completely understand how an SSD behaves in a complete host system, it is mandatory to consider that applications running on top of it have not been developed to work with NAND flash memories. The development of filesystems or data intensive applications, in fact, was driven for many years by HDDs characteristics which are completely different from a NAND flash based SSD. As a consequence, in order to make HDD-like applications compatible with SSDs, it is mandatory to leverage on algorithms (the FTL) whose responsibility is the management and the translation of the host commands into NAND flash-compatible sequences. However, even if FTLs have been extensively studied and optimized to be as efficient as possible, their execution is still performed on top of the SSD controller which has to reserve a considerable amount of computational resources, and hence power, to execute it.

As presented in Section 4.1, the NAND flash memory subsystem adopted in high-performing SSDs leverages on a massive parallelization of the memory dies, and hence, it is one of the most power-hungry parts of the disk. As a consequence, only a relatively small amount of power (in the order of tens of Watts) can be used for the SSD controller implementation [88]. At this point it becomes clear that designing a low-power processor able to execute extremely complex algorithms is becoming more and more challenging, and, in future SSDs architecture where thousands of NAND flash chips will be used, there will be less room for complex processing units inside the disk.

The "Open-Channel" initiative [87] aims to solve the aforementioned problems introducing a lightweight system where both user's applications and the storage layer are NAND-flash aware [95]. Basically, in this new storage architecture both the FTL routines and the ECC algorithms are moved from the SSD controller to a more powerful processor located outside the SSD card. This processing unit can be either the host processor or a dedicated accelerator in the form of a Multi-Purpose Processing Array (MPPA).

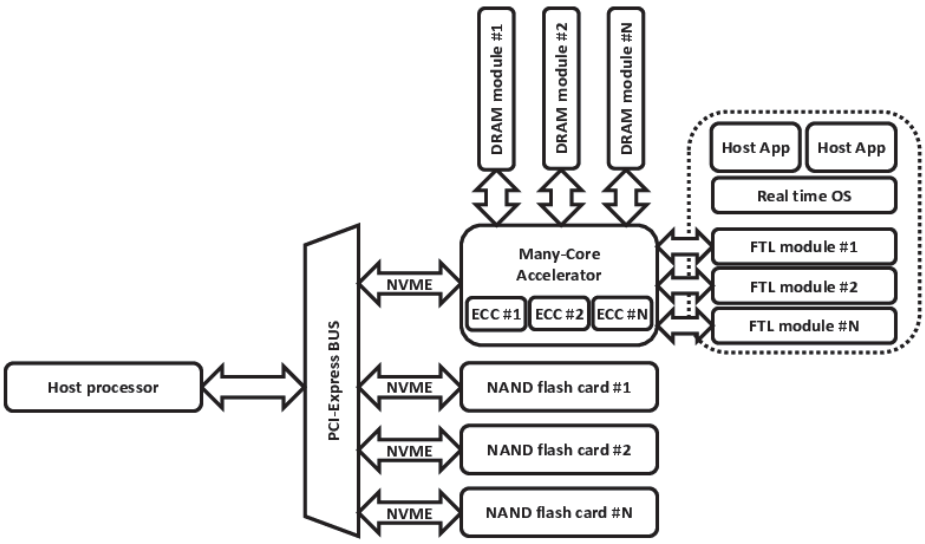


Figure 4.12: Baseline architecture modeled by the "Open-Channel" storage layer.

Fig. 4.12 shows a hypothetical architecture that can be modeled by the "Open-Channel" storage system. Basically, thanks to the PCI-Express interconnection and the NVM-Express protocol [26], a bunch of NAND flash cards and an MPPA can establish a pear-to-pear communication without requesting any specific management from the host processor [96]. In this architecture, however, the used NAND flash cards have not to be intend as

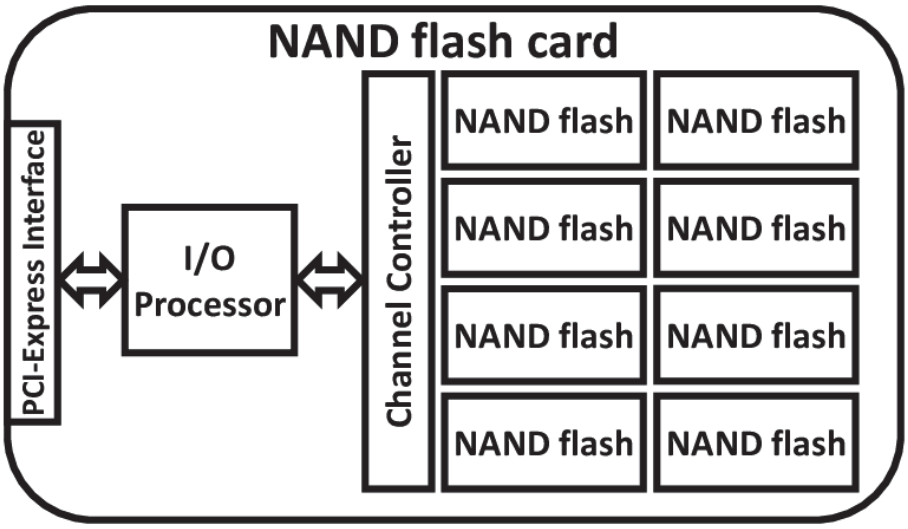


Figure 4.13: NAND flash card used in the "Open-Channel" storage system.

traditional SSD because, beside a simple I/O processor and a channel controller deputed for NAND flash addressing, they do not embody any complex processor, DRAM, FTL or even ECC (see Fig. 4.13). As a consequence, data read/written from/to these cards have to be considered as the raw output/input of the NAND flash memories without any further manipulation.

One of the main advantages made available by this new storage system, is that the host processor can view the MPPA either as a block device or as a programmable accelerator. In fact, using the I/O-pmem library [97] through the NVM-Express protocol, it is possible to address the the DRAM modules hosted on top of the accelerator like a standard I/O block device. The MPPA, on the other hand, can manipulate and process the incoming host's data exploiting specific applications hosted on top of its processors. With this respect, in fact, the accelerator can execute these programs by means of a custom Real Time Operating System (RT-OS) designed to manage I/O operations in a more efficient way than the host OS. Moreover, along with the custom RT-OS, different FTLs managing the NAND flash cards or even ECC algorithm can be executed on top of the MPPA relieving the non-volatile storage layer to execute complex and power-hungry algorithms. This solution leads to a flexible and power-efficient framework for data intensive applications where the data manipulation is performed directly on top of the storage backbone and the host system is free to execute other critical task operations.

4.2.1 Simulation model

A possible way to simulate the proposed version of the "Open-Channel" architecture is to use a programmable virtual manager able to virtualize all the peripherals needed by the host system. With this respect, Qemu represents the preferable choice since it is open-source and it is ready to emulate complete OS with several virtualized peripherals such as the DRAM controller, the Advanced Host Controller Interface (AHCI), the Network Interface controller (NIC), the PCI-Expresse controller, etc. [4].

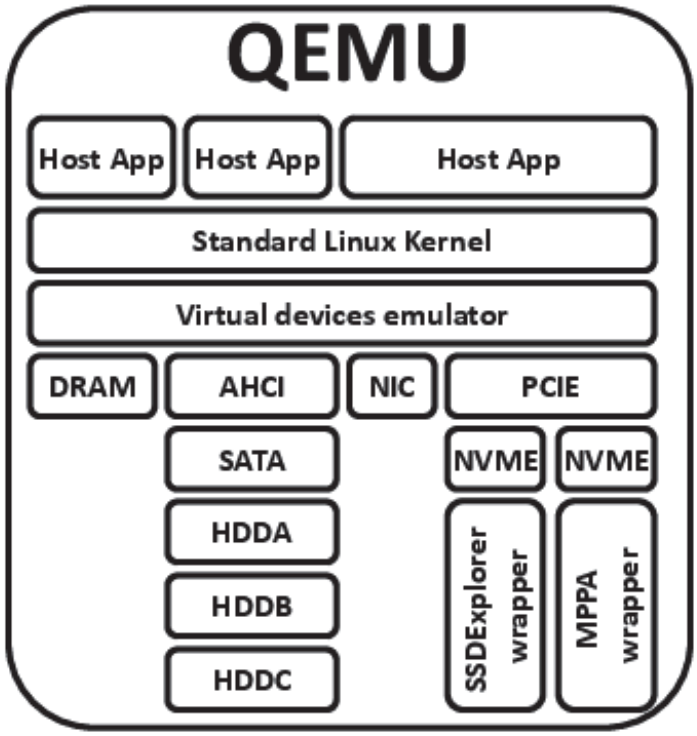


Figure 4.14: Template architecture modeled by Qemu. A custom wrapper for the SSDExplorer and the MPPA simulators are attached to the PCI-Express bus like traditional block devices.

As it can be seen in Fig. 4.14, the Qemu platform allows developing virtual devices which can be attached to the simulator's drivers and recognized as standard block devices. In this case, two wrappers have been attached to the NVM-Express driver:

- the SSDExplorer wrapper: which acts as an I/O trace collector for the "Online-Offline" simulation mode of the tool presented in Section 1.3.1;

- the MPPA wrapper: which is a programmable-transactional-based functional simulator of the routines executed by the accelerator. Basically, this tool introduces the proper delay for all the operations performed by the user and the FTLs producing the corresponding I/O trace for the SSDExplorer wrapper;

Thanks to this approach it will be possible assessing in a single integrated framework how the new storage layer proposed by the "Open-Channel" architecture will impact the host system in terms of performance and latency. Moreover it will be possible to relate the user applications to the actual power consumption of the NAND flash cards, allowing to optimize not only the performance of the system by also its power efficiency.

Conclusions

In this thesis it has been presented a thorough design space exploration of present and future Solid State Drive architectures. This work has been made possible thanks to the development of SSDExplorer, a dedicated CAD tool able to accurately simulate the SSD's behavior performing a fast yet fine-grained design space exploration of the disk's architecture. Thanks to this tool, it has been possible relating the design phase of an SSD with the features of the used memories, highlighting how the reliability, the architecture, and the power consumption of these devices are able to impact the final disk's performance. Whit this respect, it must be highlighted that in previous research works the SSD's performance were coupled only with the FTL efficiency because neither an accurate hardware modeling nor the memories' characteristics were taken into account.
In this work it has been demonstrated that:

- when NAND flash memories are used as the main storage medium it is mandatory to design the SSD architecture considering also the reliably figures of these devices. In fact, to deal with the increasing RBERs of NAND flash memories, and hence to guarantee the data trustworthiness over time, sophisticated error correction algorithms such as the Read Retry or the LDPC Soft-Decision have to be embedded inside the SSD controller. However, collected results show that these approaches trade a data reliability improvement with a deterioration of the latency and the bandwidth figures of the disk, and hence, a performance/reliability trade-off is manifested. This phenomenon, heavily modifies the design phase of an SSD since, it is mandatory to carefully study the disk's architecture in order to maintain the target performance, latency, and QoS requirements during the whole SSD lifetime;

- to efficiently design a high-preforming SSDs constructed from RRAMs ("All-RRAM" SSDs), it is mandatory to consider the architecture of the underlying storage medium avoiding to use these emerging memories as traditional NAND flash chips. With this respect, in fact, achieved re-

sults show that even if RRAMs are provided with the standard NAND flash interface (ONFI) which enables the use of these devices with nowadays SSD controllers, to fully exploit these memories it is necessary to implement a custom disk architecture with a specific data management algorithm. If this optimization is not performed, "All-RRAM" SSDs are not able to keep the promise of a low latency and high performing storage system and, in worst-case conditions, they show the same performance of a traditional NAND flash-based SSD;

- the power consumption of the underlying storage system is one of the main parameters influencing the architecture of an SSDs. With this respect, the growing necessity of high performance and high storage capacity of SSD architectures is pushing designers to embed hundreds or even thousands of memory dies on the same SSD card. This approach leads to extremely complex management routines that cannot be executed on top of a traditional SSD controller which, moreover, has to be designed to be as low power as possible. As a consequence, "software-defined" and "Open-Channel" approaches are now becoming more appealing since they propose to optimize the power consumption of an SSD either customizing the use of the disk limiting its application scenario, or moving the processing effort outside the disk controller exploiting advanced processing systems such as MPPAs;

Overall, what ultimately stands out from the above considerations is that in any case the performance, the reliability, the architecture, and the power consumption characteristics of the exploited storage medium represent the main drivers of the design phase of an SSD. Collected results clearly proved that the "memory-centric" bottom-up design flow devised in this work allows optimizing each single hardware/software resource of the disk connecting the physical characteristics of the exploited memories with the final user application.

Bibliography

[1] R. Micheloni, A. Marelli, and K. Eshghi. *Inside Solid State Drives (SSDs)*. Springer London, 2012.

[2] The OpenSSD Project. `http://www.openssd-project.org/wiki/The_OpenSSD_Project`.

[3] Jinsoo Yoo, Youjip Won, Joongwoo Hwang, Sooyong Kang, Jongmoo Choi, Sungroh Yoon, and Jaehyuk Cha. Vssim: Virtual machine based ssd simulator. In *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–14, 2013.

[4] QEMU: open source processor emulator. `http://wiki.qemu.org/Main_Page`.

[5] The DiskSim simulation environment version 4.0, 2008. `http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf`.

[6] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR*, pages 10:1–10:9, 2009.

[7] Ocz vertex series 120GB SSD. `http://ocz.com/consumer`.

[8] FLASHTEC NVRAM Drives. `http://pmcs.com/products/storage/flashtec_nvram_drives/`.

[9] Youngjae Kim, B. Tauras, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *International Conference on Advances in System Simulation (SIMUL)*, pages 125–131, 2009.

[10] Jongmin Lee, Eujoon Byun, Hanmook Park, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Cps-sim: configurable and accurate clock precision solid state drive simulator. In *Proceedings of the ACM symposium on Applied Computing*, pages 318–325, 2009.

[11] Hoeseung Jung, Sanghyuk Jung, and Yong Ho Song. Architecture exploration of flash memory storage controller through a cycle accurate profiling. *IEEE Transactions on Consumer Electronics*, 57(4):1756–1764, 2011.

[12] E.-Y. Chung. A Solid-State Disk Simulator for Quantitative Performance Analysis and Optimization. In *NVRAMOS*, 2009.

[13] Cagdas Dirik and Bruce Jacob. The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. In *International Symposium on Computer Architecture (ISCA)*, pages 279–289, 2009.

[14] S. Zertal and W. Dron. Quantitative study of solid state disks for mass storage. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 149–155, 2010.

[15] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. Ldpc-in-ssd: Making advanced error correction codes work effectively in solid state drives. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 243–256, 2013.

[16] Sungjin Lee, Kermin Fleming, Jihoon Park, Keonsoo Ha, Adrian M. Caulfield, Steven Swanson, Arvind, and Jihong Kim. Bluessd: An open platform for cross-layer experiments for nand flash-based ssds. In *The 5th Workshop on Architectural Research Prototyping*, 2010.

[17] Systemc 2.0.1 language reference manual, 2002. `http://www.systemc.org`.

[18] Sungpack Hong, Sungjoo Yoo, Sheayun Lee, Sangwoo Lee, Hye Jeong Nam, Bum-Seok Yoo, Jaehyung Hwang, Donghyun Song, Janghwan Kim, Jeongeun Kim, HoonSang Jin, Kyu-Myung Choi, Jeong-Taek Kong, and SooKwan Eo. Creation and utilization of a virtual platform for embedded software optimization:: an industrial case study. In *Proceedings of the International Conference Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 235–240, Oct 2006.

[19] S. Pasrich and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 2008.

[20] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011.

[21] Open Virtual Platforms - the source of Fast Processor Models and PlatformsOpen Virtual Platforms. `http://www.ovpworld.org`.

[22] Open Nand Flash Interface (ONFI). `http://www.onfi.org`.

[23] Evatronix NAND Flash controller ip-core. `http://www.evatronix-ip.com/ip-cores/memory-controllers/nand-flash.html`.

[24] Serial ATA International Organization. *SATA revision 3.0 specifications*. `www.sata-io.org`.

[25] SATA-IP host reference design on SP605 manual, Apr 2013. Accessed.

[26] NVM Express, 2013. `http://www.nvmexpress.org/`.

[27] Nvm express 1.1 specification, 2013. `http://nvmexpress.org/wp-content/uploads/2013/05/NVM_Express_1_1.pdf`.

[28] Open-Silicon. SATA device controller - product brief, 2013. `http://www.open-silicon.com/ip-technology/open-silicon-ip/io-controllers/sata-device-controller/`.

[29] Intel X18-M X25-M SATA Solid State Drive. Enterprise Server/Storage Applications. `http://cache-www.intel.com/cd/00/00/42/52/425265_425265.pdf`.

[30] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011.

[31] Myoungsoo Jung, E.H. Wilson, D. Donofrio, J. Shalf, and M.T. Kandemir. Nandflashsim: Intrinsic latency variation aware nand flash memory system modeling and simulation at microarchitecture level. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12, 2012.

[32] Youngjoo Lee, Hoyoung Yoo, Injae Yoo, and In-Cheol Park. 6.4gb/s multi-threaded bch encoder and decoder for multi-channel ssd controllers. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 426–428, Feb 2012.

[33] L. Zuolo, C. Zambelli, R. Micheloni, D. Bertozzi, and P. Olivo. Analysis of reliability/performance trade-off in solid state drives. In *IEEE International Reliability Physics Symposium*, pages 4B.3.1–4B.3.5, June 2014.

[34] Duo Liu, Yi Wang, Zhiwei Qin, Zili Shao, and Yong Guan. A space reuse strategy for flash translation layers in slc nand flash memory storage systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(6):1094–1107, June 2012.

[35] Tianzheng Wang, Duo Liu, Yi Wang, and Zili Shao. Ftl2: A hybrid flash translation layer with logging for write reduction in flash memory. *SIGPLAN Not.*, 48(5):91–100, June 2013.

[36] Duo Liu, Yi Wang, Zhiwei Qin, Zili Shao, and Yong Guan. A space reuse strategy for flash translation layers in slc nand flash memory storage systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(6):1094–1107, June 2012.

[37] Yuan-Hao Chang, Po-Chun Huang, Pei-Han Hsu, L.-J. Lee, Tei-Wei Kuo, and D.H.-C. Du. Reliability enhancement of flash-memory storage systems: An efficient version-based design. *Computers, IEEE Transactions on*, 62(12):2503–2515, Dec 2013.

[38] Tianzheng Wang, Duo Liu, Yi Wang, and Zili Shao. Ftl2: A hybrid flash translation layer with logging for write reduction in flash memory. *SIGPLAN Not.*, 48(5):91–100, June 2013.

[39] Intel Shows PAX Attendees SSD Overclocking. `http://www.legitreviews.com/intel-shows-pax-attendees-ssd-overclocking_122557`.

[40] Indilix barefoot controller. `http://www.indilinx.com/solutions/barefoot.html`.

[41] IOzone Filesystem Benchmark. `http://www.iozone.org/`.

[42] UMassTraceRepository. `http://traces.cs.umass.edu/index.php/Storage/Storage`.

[43] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M.Olivieri. Mparm: Exploring the multi-processor soc design space with systemc. *Journal of VLSI SIgnal Processing*, 41:169–182, 2005.

[44] An Overview of SSD Write Caching. http://community.spiceworks.com/attachments/post/0013/5918/ssd_write_caching_tech_brief_lo.pdf.

[45] Samsung NAND Flash memory K9XXG08UXM series. http://www.arm9board.net/download/fl6410/datasheet/k9g8g08.pdf.

[46] Intel solid-state drive dc s3700 series – quality of service., 2013. http://www.intel.com/content/www/us/en/solid-state-drives/ssd-dc-s3700-quality-service-tech-brief.html.

[47] The cost of latency, 2015. http://www.telx.com/blog/the-cost-of-latency/.

[48] Platform as a service (paas), 2015. http://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS.

[49] N. Mielke, T. Marquart, Ning Wu, J. Kessenich, H. Belgal, Eric Schares, F. Trivedi, E. Goodness, and L.R. Nevill. Bit error rate in NAND Flash memories. In *IEEE International Reliability Physics Symposium (IRPS)*, pages 9–19, 2008.

[50] Sungjin Lee, Taejin Kim, Kyungho Kim, and Jihong Kim. Life-time Management of Flash-based SSDs Using Recovery-aware Dynamic Throttling. In *USENIX Conference on File and Storage Technologies, (FAST'12)*, 2012.

[51] Ren-Shuo Liu, Chia-Lin Yang, and Wei Wu. Optimizing NAND Flash-Based SSDs via Retention Relaxation. In *USENIX Conference on File and Storage Technologies, (FAST'12)*, 2012.

[52] Laura M. Grupp, John D. Davis, and Steven Swanson. The Bleak Future of NAND Flash Memory. In *USENIX Conference on File and Storage Technologies, (FAST'12)*, 2012.

[53] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis, and modeling. In *Design, Automation Test in Europe Conference (DATE), 2013*, pages 1285–1290.

[54] Xueqiang Wang, Guiqiang Dong, Liyang Pan, and Runde Zhou. *Error Correction Codes and Signal Processing in Flash Memory, Flash Memories*. Igor Stievano (Ed.), 2011.

[55] Intel-Corporation Robert Frickey. Data Integrity on 20nm SSDs. In *Flash Memory Summit*, 2012.

[56] A.G. COMETTI, L.B. Huang, and A. Melik-Martirosian. Apparatus and method for determining a read level of a flash memory after an inactive period of time, February 4 2014. US Patent 8,644,099.

[57] Silicon Motion Jeff Yang. High-Efficiency SSD for Reliable Data Storage Systems. In *Flash Memory Summit*, 2012.

[58] Intel-Corporation Ravi Motwani. Exploitation of RBER Diversity across Dies to Improve ECC Performance in NAND Flash Drive. In *Flash Memory Summit*, 2012.

[59] Pci express base 3.0 specification, 2013. `http://www.pcisig.com/specifications/pciexpress/base3/`.

[60] E. Yeo. An LDPC-enabled flash controller in 40nm CMOS. In *Proc. of Flash Memory Summit*, Aug. 2012.

[61] X. Hu. LDPC codes for flash channel. In *Proc. of Flash Memory Summit*, Aug. 2012.

[62] Erich F. Haratsch. LDPC Code Concepts and Performance on High-Density Flash Memory. In *Proc. of Flash Memory Summit*, Aug. 2014.

[63] Tong Zhang. Using LDPC Codes in SSD — Challenges and Solutions. In *Proc. of Flash Memory Summit*, Aug. 2012.

[64] R. Micheloni, A. Marelli and R. Ravasio. Basic coding theory. In R. Micheloni, A. Marelli, and R. Ravasio, editor, *Error Correction Codes for Non-Volatile Memories*, pages 1–33. Springer-Verlag, 2008.

[65] L. Zuolo, C. Zambelli, P. Olivo, R. Micheloni, and A. Marelli. LDPC Soft Decoding with Reduced Power and Latency in 1X-2X NAND Flash-Based Solid State Drives. In *IEEE International Memory Workshop (IMW)*, pages 1–4, May 2015.

[66] D.H. Nguyen and F.F. Roohparvar. Increased nand flash memory read throughput, March 8 2011. US Patent 7,903,463.

[67] S.H. Lee, S. Bae, J.N. Baek, H.S. Kim, and S.B. Kim. Method of reading data from a non-volatile memory and devices and systems to implement same, March 28 2013. US Patent App. 13/429,326.

[68] N. Shibata, K. Kanda, T. Hisada, K. Isobe, M. Sato, Y. Shimizu, T. Shimizu, T. Sugimoto, T. Kobayashi, K. Inuzuka, N. Kanagawa, Y. Kajitani, T. Ogawa, J. Nakai, K. Iwasa, M. Kojima, T. Suzuki, Y. Suzuki, S. Sakai, T. Fujimura, Y. Utsunomiya, T. Hashimoto, M. Miakashi, N. Kobayashi, M. Inagaki, Y. Matsumoto, S. Inoue, Y. Suzuki, D. He, Y. Honda, J. Musha, M. Nakagawa, M. Honma, N. Abiko, M. Koyanagi, M. Yoshihara, K. Ino, M. Noguchi, T. Kamei, Y. Kato, S. Zaitsu, H. Nasu, T. Ariki, H. Chibvongodze, M. Watanabe, H. Ding, N. Ookuma, R. Yamashita, G. Liang, G. Hemink, F. Moogat, C. Trinh, M. Higashitani, T. Pham, and K. Kanazawa. A 19nm 112.8mm^2 64Gb multi-level flash memory with 400Mb/s/pin 1.8V Toggle Mode interface. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 422–424, Feb. 2012.

[69] Daeyeal Lee, Ik Joon Chang, Sang-Yong Yoon, Joonsuc Jang, Dong-Su Jang, Wook-Ghee Hahn, Jong-Yeol Park, Doo-Gon Kim, Chiweon Yoon, Bong-Soon Lim, Byung-Jun Min, Sung-Won Yun, Ji-Sang Lee, Il-Han Park, Kyung-Ryun Kim, Jeong-Yun Yun, Youse Kim, Yong-Sung Cho, Kyung-Min Kang, Sang-Hyun Joo, Jin-Young Chun, Jung-No Im, Seunghyuk Kwon, Seokjun Ham, Ansoo Park, Jae-Duk Yu, Nam-Hee Lee, Tae-Sung Lee, Moosung Kim, Hoosung Kim, Ki-Whan Song, Byung-Gil Jeon, Kihwan Choi, Jin-Man Han, Kye Hyun Kyung, Young-Ho Lim, and Young-Hyun Jun. A 64Gb 533Mb/s DDR interface MLC NAND Flash in sub-20nm technology. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 430–432, Feb. 2012.

[70] Hp z640 workstation, 2015. `http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04434085`.

[71] Flexible I/O tester, 2015. `http://freecode.com/projects/fio`.

[72] J. Kim, E. Lee, J. Choi, D. Lee, and S. Noh. Chip-level raid with flexible stripe size and parity placementfor enhanced ssd reliability. *IEEE Transactions on Computers*, 2014. to appear on.

[73] E.I. Vatajelu, H. Aziza, and C. Zambelli. Nonvolatile memories: Present and future challenges. In *International Design Test Symposium (IDT)*, pages 61–66, Dec. 2014.

[74] C. Zambelli, A. Grossi, D. Walczyk, T. Bertaud, B. Tillack, T. Schroeder, V. Stikanov, P. Olivo, and C. Walczyk. Statistical analysis of resistive switching characteristics in ReRAM test arrays. In *IEEE*

Int. Conf. on Microelectronics Test Structures (ICMTS), pages 27–31, Mar. 2014.

[75] K. Takeuchi. Hybrid solid-state storage system with storage class memory and nand flash memory for big-data application. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1046–1049, Jun. 2014.

[76] S. Dubois. Crossbar Resistive RAM (RRAM): The Future Technology for Data Storage. In *SNIA Data Storage Innovation Conference*, Apr. 2014.

[77] S Bates, M Asnaashari, and L. Zuolo. Modelling a High-Performance NVMe SSD constructed from ReRAM. In *Proc. of Flash Memory Summit*, Aug. 2015.

[78] Lorenzo Zuolo, Cristian Zambelli, Rino Micheloni, Salvatore Galfano, Marco Indaco, Stefano Di Carlo, Paolo Prinetto, Piero Olivo, and Davide Bertozzi. SSDExplorer: a Virtual Platform for Fine-Grained Design Space Exploration of Solid State Drives. In *Design, Automation Test in Europe (DATE), 2014*, pages 1–6.

[79] Open Nand Flash Interface (ONFI) revision 4.0. `www.onfi.org/~/media/onfi/specs/onfi_4_0-gold.pdf?la=en`.

[80] Sean Barry. All Flash Array Data Protection Schemes. In *Proc. of Flash Memory Summit*, Aug. 2015.

[81] Doug Rollins. Simplification: Get All Flash Performance Easily, Gradually, As Your Needs Grow. In *Proc. of Flash Memory Summit*, Aug. 2015.

[82] Erik Ottem. All Flash Arrays in Healthcare. In *Proc. of Flash Memory Summit*, Aug. 2015.

[83] Walter Amsler. All Flash Array Customer Case Study. In *Proc. of Flash Memory Summit*, Aug. 2015.

[84] Avraham Meir. File on Flash: Delivering on the Promise of Webscale, All Flash, Distributed File Systems. In *Proc. of Flash Memory Summit*, Aug. 2015.

[85] Somnath Roy. Ceph Optimization on All Flash Storage. In *Proc. of Flash Memory Summit*, Aug. 2015.

[86] The kalray multi-purpose-processing-array (mppa), 2016. http://www.kalrayinc.com.

[87] Open-channel solid state drives, 2016. http://openchannelssd.readthedocs.org/en/latest/.

[88] U.s. epa. report to congress on server and data energy efficiency. tech. rep., u.s. environmental protection agency, 2007.

[89] Matias Bjørling, Philippe Bonnet, Luc Bouganim, and Bjorn Jonsson. uflip: Understanding the energy consumption of flash devices. *IEEE Data(base) Engineering Bulletin*, 33(4), 2010.

[90] Balgeun Yoo, Youjip Won, Jongmoo Choi, Sungroh Yoon, Seokhei Cho, and Sooyong Kang. Ssd characterization: From energy consumption's perspective. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems*, pages 3–3, 2011.

[91] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, and S. Swanson. Modeling power consumption of nand flash memories using flashpower. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(7):1031–1044, July 2013.

[92] Maxim max 9938, 2011. http://www.maximintegrated.com/en/products/analog/amplifiers/MAX9938.html.

[93] M. Sako, Y. Watanabe, T. Nakajima, J. Sato, K. Muraoka, M. Fujiu, F. Kouno, M. Nakagawa, M. Masuda, K. Kato, Y. Terada, Y. Shimizu, M. Honma, A. Imamoto, T. Araya, H. Konno, T. Okanaga, T. Fujimura, X. Wang, M. Muramoto, M. Kamoshida, M. Kohno, Y. Suzuki, T. Hashiguchi, T. Kobayashi, M. Yamaoka, and R. Yamashita. 7.1 a low-power 64gb mlc nand-flash memory in 15nm cmos technology. In *IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 1–3, Feb 2015.

[94] Jie Zhang, Mustafa Shihab, and Myoungsoo Jung. Power, energy and thermal considerations in ssd-based i/o acceleration. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, pages 15–15, 2014.

[95] Javier González, Matias Bjørling, Seongno Lee, Charlie Dong, and Yiren Ronnie Huang. Application-driven flash translation layers on open-channel ssds. In *Proceedings of the 7th Non Volatile Memory Workshop (NVMW)*, pages 1–2, 2016.

[96] S Bates. Accelerating Data Centers Using NVMe and CUDA. In *Proc. of Flash Memory Summit*, Aug. 2014.

[97] Persistent memory programming, 2016. `http://pmem.io`.