[Article] SSDExplorer: a Virtual Platform for Performance/Reliability-oriented Fine-Grained Design Space Exploration of Solid State Drives

(Article begins on next page)

SSDExplorer: a Virtual Platform for Performance/Reliability-oriented Fine-Grained Design Space Exploration of Solid State Drives

Lorenzo Zuolo, Cristian Zambelli, *Member, IEEE,* Rino Micheloni, *Senior Member, IEEE,*
Marco Indaco, *Student, IEEE,* Stefano Di Carlo, *Senior Member, IEEE,* Paolo Prinetto, *Senior Member, IEEE,*
Davide Bertozzi, *Member, IEEE,* and Piero Olivo

Abstract—Currently available EDA tools for design space exploration of Solid State Drives (SSDs) are not able to assess: *i)* the device architecture inefficiencies; *ii)* the architecture overdesign for a target performance; *iii)* the performance degradation caused by the disk usage. These tools feature either an overly high abstraction modeling strategy or lack the required flexibility to perform design exploration. To overcome these problems, this paper proposes SSDExplorer, a tool for fine-grained yet reasonably fast design space exploration of different SSD architectures highlighting possible bottlenecks. To prove its accuracy SSDExplorer has been validated with two real SSDs. SSDExplorer efficiency has been assessed by evaluating the impact of the NAND Flash read retry algorithm impact on the SSD performance as a function of its internal architecture.

I. INTRODUCTION

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

Lorenzo Zuolo, Cristian Zambelli, Davide Bertozzi and Piero Olivo are with the Dipartimento di Ingegneria, Università degli Studi di Ferrara, via G. Saragat,1 - 44122 Ferrara (Italy).

Marco Indaco, Stefano Di Carlo and Paolo Prinetto are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi, 24 - 10129 Torino (Italy).

Rino Micheloni is with PMC-Sierra, Via Torri Bianche 1, 20871 Vimercate (Italy).

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 work we propose 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

TABLE I
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

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].
- *Reliability assessment of several SSD architectures and evaluation of their performance drop related to NAND Flash components wearout.* The read retry algorithm in Multi-Level cell NAND Flash is here targeted as a case study.

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.

II. 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 in modifying 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.

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, Tab. I 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. This paper extends the work in [17] to show a direct comparison between SSDExplorer and a state-of-the-art emulator [3] proving the accuracy of the proposed framework considering different SSD architectures and eventually identifying its fields of application.

III. SSDEXPLORER AT A GLANCE

A. 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 [18]. 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 [19], 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
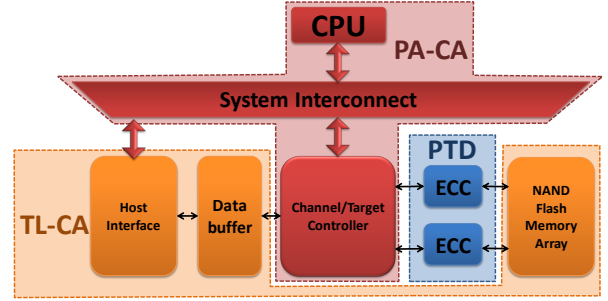

Fig. 1. Default architecture template modeled by the simulator.

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.

B. Models exploited in SSDExplorer

Fig. 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. We follow the terminology in [20] 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 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) <u>CPU</u>: SSDExplorer can implement any CPU, including both custom IP cores and advanced multicore architectures given the availability of its PA-CA model and of its proper Instruction Set Simulator. The programming model of SSDExplorer is built to support the inter-operability with state-of-the-art processors simulators such as Gem5 [21] and Open Virtual Platforms [22] in order to enable future modeling and analysis activities on the processor role in the SSDs.

2) <u>System Interconnect</u>: 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) <u>Channel/Target Controller</u>: 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) [23] 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 2.0 port and a command translator. The microarchitecture described in [24] 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, we have 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 we spawn the processes used for memory simulation 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, we can mitigate the impact of the memory subsystem on the simulation speed.

1) <u>Host Interface</u>: 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 [25] and operation timings have been accurately modeled following the SATA protocol timing directives provided in [26]. 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 [27]) protocol that significantly reduces packetization latencies with respect to standard SATA interfaces [28]. 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 [29] able to transfer data from the host interface to the data buffers and vice versa is available in SSDExplorer.

2) <u>DRAM Buffer</u>: this component is used either as a temporary storage buffer for read/write data or as a buffer [30] 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 precharging, refresh operations, detailed command timings, etc.). The data buffers of SSDExplorer are modeled with a SystemC customized version of the simulator proposed in [31]. 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) <u>NAND flash memory array</u>: 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 [32]. 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.

Fig. 2. A SSD template architecture used for FTL and WAF algorithms evaluations

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. [33]. 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 a configurable PTD model of a Bose-Chaudhuri-Hocquenghem (BCH) ECC engine. This block is composed of a fixed high-speed encoder and a multi-machine decoder. The delay of the decoder stage can be configured to mimic the behavior of a single- or a multi-threaded version as shown in [33]. 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 [34].
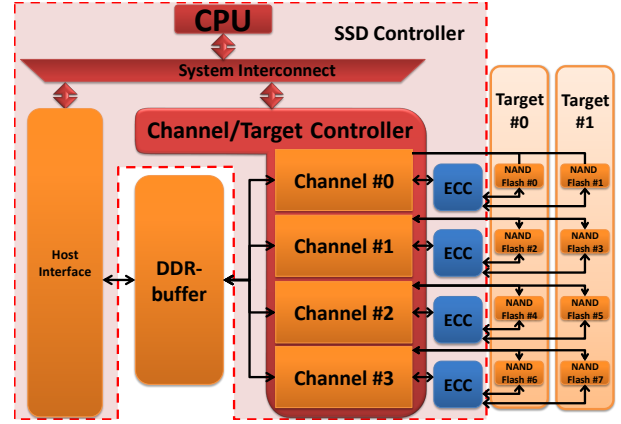
IV. FTL ALGORITHMS ABSTRACTION

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 [35], [36]). This calls for an estimation of the impact of software management algorithms usually exploited by a SSD (e.g., Garbage Collection (GC), Wear Leveling (WL), etc.) without impacting the framework complexity. 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. In this work, as discussed in section III-B, thanks to the standard programming model used by the CPU, both a realistic algorithm implementation and its abstraction through a WAF model are supported by the simulator.

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.

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. 2 has been used, configured with the values reported in Tab. II.

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 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 the actual memory size by reducing the effective cluster capacity. In such a way longer processes like

TABLE II
SSDEXPLORER CONFIGURATION FOR WAF ACCURACY ASSESSMENT.

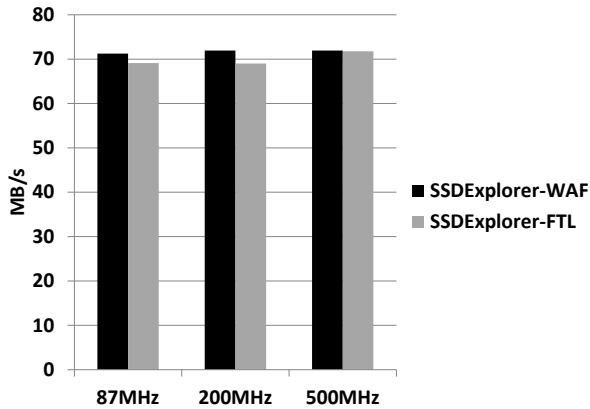| Parameter	Architecture
Host Interface	SATA II
DRAM-Buffer	1
Mock-up DRAM-size	64 kBytes
Channels	4
Targets	2
NAND Flash Dies	4
NAND Flash Planes	2
NAND Flash Blocks	16
NAND Flash Pages	4
NAND Flash Page Size	4096 Bytes
Caching	No
GC threshold	30%
GC reserved blocks	1
Over-provisioning	20%

Fig. 3. Sequential read bandwidth achieved by SSDExplorer using the WAF abstraction model and a real FTL.
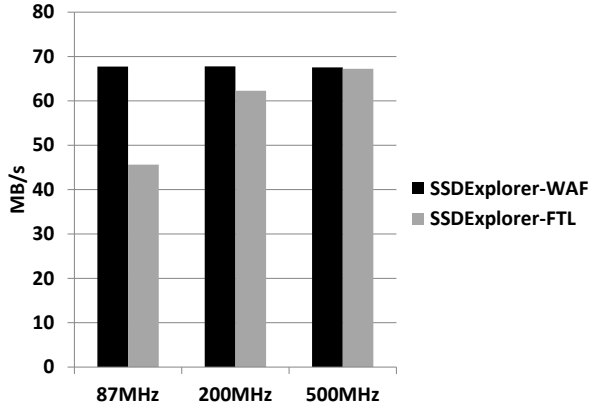

Fig. 4. Sequential write bandwidth achieved by SSDExplorer using the WAF abstraction model and a real FTL.

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. 3 shows the performance achieved with a read workload: since read transactions do not require software
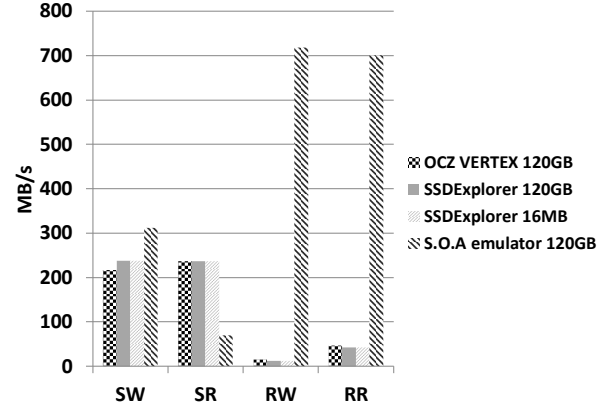

Fig. 5. Performance comparison between OCZ Vertex 120GB, SSDExplorer (both mock-up and full disk simulation), and a S.O.A emulator tool [3] in terms of throughput for Sequential Write (SW), Sequential Read (SR), Random Write (RW) and Random Read (RR).

manipulations, the FTL execution time becomes marginal, therefore not affecting the overall bandwidth. When a write workload is considered, Fig. 4 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 firmware. By increasing the SSD controller speed, the FTL components contribution not related to the GC on the SSD performance 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 [37], a WAF abstraction of the FTL represents a good simulation speed/accuracy trade-off.

V. SIMULATION RESULTS AND APPLICATIONS

A. Performance comparison with real SSD platforms

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 [38], 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 [39]: 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 [40] is justified by the fact that the latter approach would
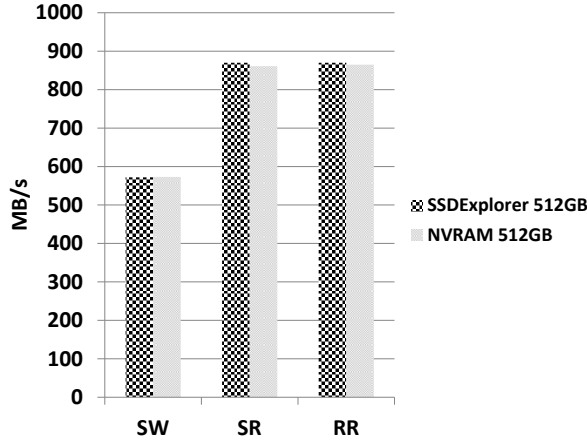
Fig. 6. Performance comparison between a 512GB NVRAM card and SSDExplorer (full disk simulation), in terms of throughput for Sequential Write (SW), Sequential Read (SR), and Random Read (RR).

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

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, we made use of 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. 5, 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. 5 confirm the accuracy provided by SSDExplorer. This is even more relevant if we consider that these low error margins can be achieved avoiding the real FTL implementation.

To ultimately prove the accuracy of the proposed framework we 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 we configured its parameters to provide the same WAF value used in SSDExplorer. As it can be seen in Fig. 5, 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 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 IV.

Enterprise Device: in this paragraph we configured SSDExplorer to track the behavior of a 512 GB NVRAM card currently exploited in cloud applications [8]. For this comparison we 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 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.6 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.

B. Optimal design point exploration

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 III 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 [41]: *write back and write through caching* and *no caching* [30]. For the former, the SSD 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
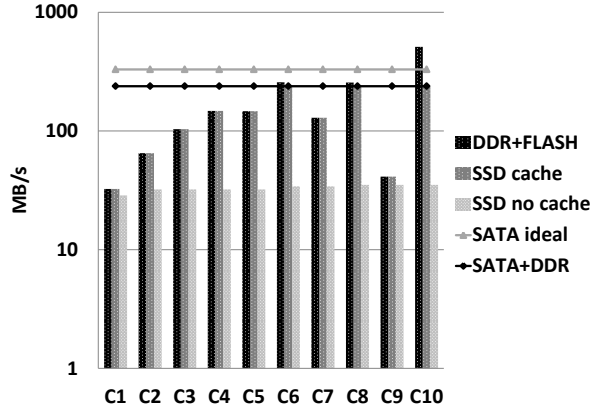
Fig. 7. Sequential Write: SATA II host interface. Comparison of the configurations proposed in Tab. III for the optimal design point exploration.

Fig. 8. Sequential Write: PCIE host interface. Comparison of the configurations proposed in Tab.III for the optimal design point exploration.

have been actually written to the NAND Flash memory. All experimental results consider a 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 [42].

Fig. 7 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. 7, 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 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
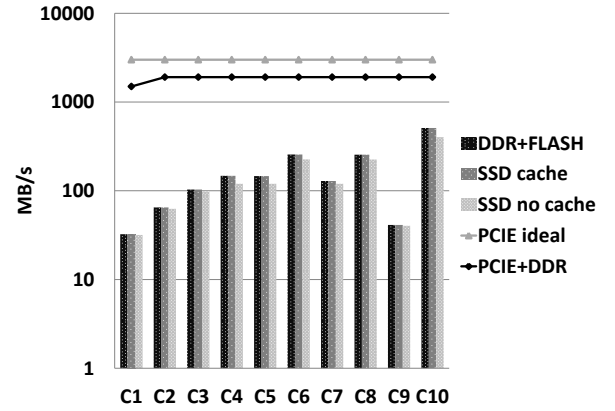
cannot acquire new commands until the current ones have been executed by the NAND Flash memories. This implies that, the internal parallelism provided by the device cannot be exploited, which becomes clear when checking the SSD performance indicated in the *DDR+FLASH* results.

To overcome this limitation and unveil the performance provided by highly parallel SSD configurations, Fig. 8 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., *C10*) is not able to saturate the interface bandwidth. However, the major result shown in Fig. 8 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.

C. A strategic SSDExplorer application: the Read Retry implications on the SSD endurance

One main limitation of current SSDs is their reliability, which is dependent upon the non-volatile NAND flash memories used as the storage medium. This reliability progressively decreases because of their intrinsic wear-out [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 [43]. Such an increase translates into the inability to correct data after a number of Program/Erase operations or after long retention times.
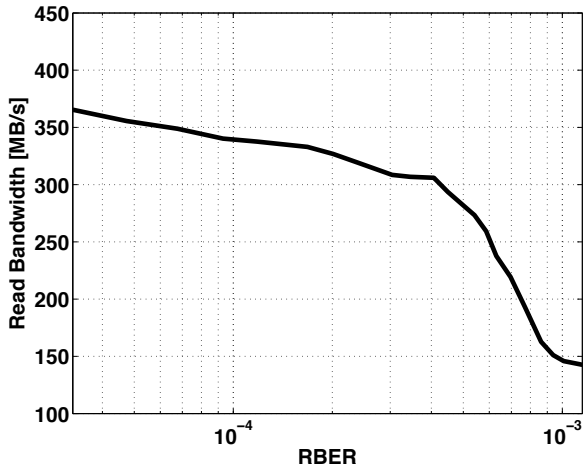
Fig. 9. SSD read bandwidth vs. RBER.

TABLE IV
SSD ARCHITECTURES CONSIDERED IN READ RETRY SIMULATIONS.

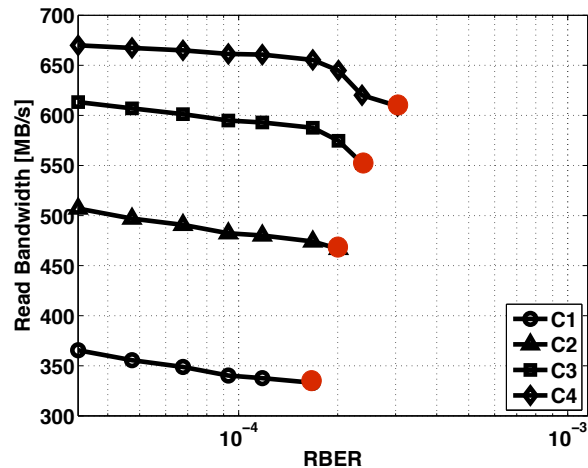| Configurations		C1	C2	C3	C4
Channels		1	2	4	8
NAND Flash Targets per Channel		16	8	4	2
ECC		1 x Channel			

Fig. 10. SSD read bandwidth vs. RBER for the four architectures described in Table IV. Curves are shown up to a 10% performance degradation with respect to the beginning of life.

To deal with increasing RBERs, NAND vendors have introduced new read techniques, such as the Read Retry (RR) algorithms, bridging the gap between the internal memory algorithms and the ECC. If, during a read operation, the number of errors in a page exceeds the ECC's correction capability, the operation is repeated with modified read parameters with the aim of reducing the number of read errors [44]. RR algorithms are needed to improve the NAND flash reliability. However, they impact on the performance and architecture of a SSD [34]. The main issues related to the RR application are:

- 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 for at the architecture level.

From a previous characterization of a 2X MLC NAND Flash technology, 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 and then fed to the modeled ECC engine in order to extract the exact delay incurred by the decoder. Fig. 9 shows the SSD read bandwidth as a function of the RBER. Since the real SSD behavior depends on a combination of Program/Erase cycles and retention times, the use of RBER as a reference metric allows us to ignore the actual degradation mechanism while generalizing the performance evaluations.

The performance degradation is caused by uncorrectable NAND Flash pages requiring the RR activation and it 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 we have repeated the analysis by modifying the number of channels and, therefore, the number of ECC

engines. Fig. 10 shows the read bandwidth behavior as a function of RBER, up to the point corresponding to a 10% degradation with respect to the beginning of SSD lifetime, for the four channels/NAND Flash targets configurations analyzed (see Table IV).

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

Another strategic activity that could benefit from SSDExplorer is the qualification of a SSD architecture. The qualification process using the standards JESD-218A and JESD-219 [45], [46] requires to determine the relationship between host writes and endured NAND Flash cycles. Operating multiple SSD architectures or SSDs with different NAND design to the desired number of Total Bytes Written (TBW) is often impractical because the time required would be excessive, then the relationship between NAND cycles and host writes could be extrapolated through CAD simulations.

Let us consider an SSD containing only one type of NAND (i.e., 2X eMLC) and no features of the drive design that would make the WAF change over the lifetime of the drive, and suppose further that the design of the WL method is expected to result in the most heavily-cycled SSD block receiving twice the average number of cycles. In that case, WAF would be a
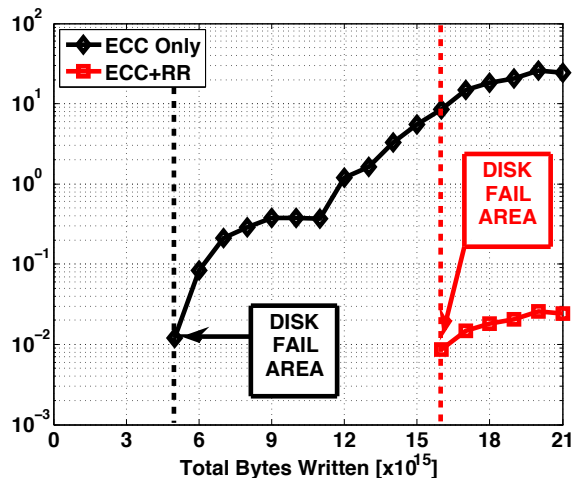
Fig. 11. SSD endurance rating for two designs: without Read Retry (i.e., ECC correction only), and with Read Retry. The disk fail area starts where the SSD's uncorrectable pages percentage is greater than zero.

constant (for a given workload), and then the estimated SSD endurance rating $f(TBW)$ is calculated with the following formula [45]:

$$f(TBW) = \frac{(TBW * 2 * WAF)}{C} \qquad (1)$$

where C is the SSD capacity. An example of this extrapolation is provided in Fig. 11, where two SSD designs are benchmarked: one including the ECC+RR correction strategy for NAND Flash and the other one using only the ECC. The results of the simulations show that by considering a disk capacity of 512 GB, and a WAF value of 2.5 (i.e., typical value for the enterprise architecture simulated in section V) it is possible to rate the SSD endurance at $5x10^{15}$ TBW and $16x10^{15}$ TBW for the ECC only and ECC+RR strategy, respectively. The rating is considered at the point where the SSD's uncorrectable pages percentage becomes greater than zero. If it is considered a SSD application scenario that performs 10 disk fill operations per day it is possible to rate the disk endurance (only related to the NAND Flash subsystem) in years: 2.67 and 8.56 for the ECC only and ECC+RR case, respectively.

VI. 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. 12 shows the kCPS achieved by SSDExplorer for 9 different SSD architectures (see Tab. V 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
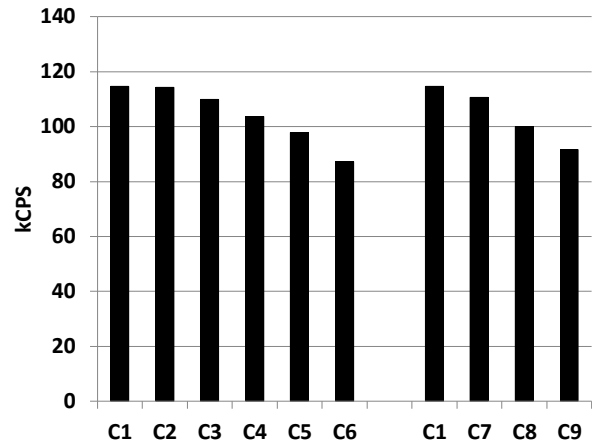

Fig. 12. SSDExplorer simulation speed with different SSD configurations using the WAF abstraction model.

NAND Flash targets, and the FTL abstraction through the WAF model is exploited.

TABLE V
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. 12, the first set of results (configuration C1 - C6) shows the simulation speed dependency on the number of instantiated channels, whereas the 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 [47].

VII. CONCLUSIONS

In this work we presented SSDExplorer, a virtual platform for fine-grained design space exploration of solid state drives. The proposed tool provides a ready to use framework able to deliver accurate performance breakdown curves of all internal SSD's components. Thanks to this feature, a SSD designer can efficiently optimize the disk architecture starting from a given target while forecasting the reliability-induced performance degradation. Moreover, the flexibility of this virtual platform allows either real or abstract FTL development, execution, and

debugging over different SSD architectures. SSDExplorer has been thoroughly validated against real devices and thanks to the wise engineering of its modeling strategy used to describe its modules, a reasonable simulation speed is still guaranteed.

VIII. Acknowledgement

References

[1] R. Micheloni, A. Marelli, and K. Eshghi, *Inside Solid State Drives (SSDs)*. Springer London, 2012.
[2] "The OpenSSD Project." [Online]. Available: http://www.openssd-project.org/wiki/The_OpenSSD_Project
[3] J. Yoo, Y. Won, J. Hwang, S. Kang, J. Choi, S. Yoon, and J. Cha, "Vssim: Virtual machine based ssd simulator," in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2013, pp. 1–14.
[4] "QEMU: open source processor emulator." [Online]. Available: http://wiki.qemu.org/Main_Page
[5] "The DiskSim simulation environment version 4.0," 2008. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/DriveChar/CMU-PDL-08-101.pdf
[6] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR*, 2009, pp. 10:1–10:9.
[7] "Ocz vertex series 120GB SSD." [Online]. Available: http://ocz.com/consumer
[8] "FLASHTEC NVRAM Drives." [Online]. Available: http://pmcs.com/products/storage/flashtec_nvram_drives/
[9] Y. 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)*, 2009, pp. 125–131.
[10] J. Lee, E. Byun, H. Park, J. Choi, D. Lee, and S. H. Noh, "Cps-sim: configurable and accurate clock precision solid state drive simulator," in *Proceedings of the ACM symposium on Applied Computing*, 2009, pp. 318–325.
[11] H. Jung, S. Jung, and Y. H. Song, "Architecture exploration of flash memory storage controller through a cycle accurate profiling," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1756–1764, 2011.
[12] E.-Y. Chung, "A Solid-State Disk Simulator for Quantitative Performance Analysis and Optimization," in *NVRAMOS*, 2009.
[13] C. Dirik and B. 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)*, 2009, pp. 279–289.
[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)*, 2010, pp. 149–155.
[15] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "Ldpc-in-ssd: Making advanced error correction codes work effectively in solid state drives," in *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 243–256.
[16] S. Lee, K. Fleming, J. Park, K. Ha, A. M. Caulfield, S. Swanson, Arvind, and J. Kim, "Bluessd: An open platform for cross-layer experiments for nand flash-based ssds," in *The 5th Workshop on Architectural Research Prototyping*, 2010.
[17] L. Zuolo, C. Zambelli, R. Micheloni, S. Galfano, M. Indaco, S. D. Carlo, P. Prinetto, P. Olivo, and D. Bertozzi, "SSDExplorer: a Virtual Platform for Fine-Grained Design Space Exploration of Solid State Drives," in *Design, Automation Test in Europe (DATE), 2014*, pp. 1–6.
[18] "Systemc 2.0.1 language reference manual," 2002. [Online]. Available: http://www.systemc.org
[19] S. Hong, S. Yoo, S. Lee, S. Lee, H. J. Nam, B.-S. Yoo, J. Hwang, D. Song, J. Kim, J. Kim, H. Jin, K.-M. Choi, J.-T. Kong, and S. 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)*, Oct 2006, pp. 235–240.
[20] S. Pasrich and N. Dutt, *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 2008.
[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[22] "Open Virtual Platforms - the source of Fast Processor Models and PlatformsOpen Virtual Platforms." [Online]. Available: http://www.ovpworld.org
[23] "Open Nand Flash Interface (ONFI)." [Online]. Available: http://www.onfi.org
[24] "Evatronix NAND Flash controller ip-core." [Online]. Available: http://www.evatronix-ip.com/ip-cores/memory-controllers/nand-flash.html
[25] *SATA revision 3.0 specifications*, Serial ATA International Organization. [Online]. Available: www.sata-io.org
[26] "SATA-IP host reference design on SP605 manual," Accessed, Apr 2013.
[27] "NVM Express," 2013. [Online]. Available: http://www.nvmexpress.org/
[28] "Nvm express 1.1 specification," 2013. [Online]. Available: http://nvmexpress.org/wp-content/uploads/2013/05/NVM_Express_1_1.pdf
[29] Open-Silicon, "SATA device controller - product brief," 2013. [Online]. Available: http://www.open-silicon.com/ip-technology/open-silicon-ip/io-controllers/sata-device-controller/
[30] "Intel X18-M X25-M SATA Solid State Drive. Enterprise Server/Storage Applications." [Online]. Available: http://cache-www.intel.com/cd/00/00/42/52/425265_425265.pdf
[31] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.
[32] M. Jung, E. Wilson, D. Donofrio, J. Shalf, and M. Kandemir, "Nand-flashsim: 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)*, 2012, pp. 1–12.
[33] Y. Lee, H. Yoo, I. Yoo, and I.-C. 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)*, Feb 2012, pp. 426–428.
[34] 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*, June 2014, pp. 4B.3.1–4B.3.5.
[35] D. Liu, Y. Wang, Z. Qin, Z. Shao, and Y. 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*, vol. 20, no. 6, pp. 1094–1107, June 2012.
[36] T. Wang, D. Liu, Y. Wang, and Z. Shao, "Ftl2: A hybrid flash translation layer with logging for write reduction in flash memory," *SIGPLAN Not.*, vol. 48, no. 5, pp. 91–100, Jun. 2013.
[37] "Intel Shows PAX Attendees SSD Overclocking." [Online]. Available: http://www.legitreviews.com/intel-shows-pax-attendees-ssd-overclocking_122557
[38] "Indilix barefoot controller." [Online]. Available: http://www.indilinx.com/solutions/barefoot.html
[39] "IOzone Filesystem Benchmark." [Online]. Available: http://www.iozone.org/
[40] "UMassTraceRepository." [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[41] "An Overview of SSD Write Caching." [Online]. Available: http://community.spiceworks.com/attachments/post/0013/5918/ssd_write_caching_tech_brief_lo.pdf
[42] "Samsung NAND Flash memory K9XXG08UXM series." [Online]. Available: http://www.arm9board.net/download/fl6410/datasheet/k9g8g08.pdf
[43] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. Nevill, "Bit error rate in NAND Flash memories," in *IEEE International Reliability Physics Symposium (IRPS)*, 2008, pp. 9–19.
[44] S. M. Jeff Yang, "High-Efficiency SSD for Reliable Data Storage Systems," in *Flash Memory Summit*, 2012.
[45] "Standard JESD218A, Solid-State Drive (SSD) Requirements and Endurance Test Method," 2011.
[46] "Standard JESD219, Solid-State Drive (SSD) Endurance Workloads," 2010.
[47] 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*, vol. 41, pp. 169–182, 2005.

Lorenzo Zuolo received the Laurea Magistrale degree (M.Sc.) in Technology for Telecommunications and Electronic Engineering from Universitá degli Studi di Ferrara respectively in 2012. Currently, he is a Ph.D student in the Dipartimento di Ingegneria of the same institution. His main research interests are focused on architectural/physical simulation of Solid State Disks (SSD) and emerging non-volatile memories.

Cristian Zambelli received the M.Sc. and Ph.D. (Hons.) degrees in electronic engineering from the University of Ferrara, Ferrara, Italy, in 2008 and 2012, respectively. He has held a Research Assistant (post-doctoral) position with the Department of Engineering, University of Ferrara, since 2012. His current research interests include the characterization, physics, and modeling of nonvolatile memories reliability and solid state drives reliability.

Rino Micheloni is Senior Principal Engineer at PMC-Sierra. Before PMC-Sierra, he was Lead Flash Technologist at IDT (Integrated Device Technology), Senior Principal for Flash and Director of Qimonda's design center in Italy, developing 36 nm and 48 nm NAND memories. From 2001 to 2006 he managed the Napoli design center of STMicroelectronics focusing on the development of 90 nm and 60 nm MLC NAND Flash. Before that, he led the development of MLC NOR Flash.

Marco Indaco is an assistant researcher of Department of Control and Computer Engineering at Politecnico di Torino. He got his Master degree in Computer Engineering from the Universit degli Studi di Napoli Federico II (Naples, Italy) and concluded in 2014 his Ph.D in Information and System Engineering at Politecnico di Torino (Turin, Italy) under the supervision of Prof. Prinetto. He has been selected for a special Ph.D program called Scuola Interpolitecnica funded by Politecnico di Milano, Politecnico di Torino and Politecnico di Bari. His research interests are mainly focused on memory reliability with particular emphasis on Solid State Drives and emerging memories. He is the recipient of different national and european awards.

Stefano Di Carlo received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been an assistant professor in the Department of Control and Computer Engineering from 2008 till 2014 and he is currently an Associate Professor in the same department. His research interests include DFT, BIST, and dependability. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.

Paolo Prinetto is full professor of Computer Engineering at the Dipartimento di Automatica e Informatica of the Politecnico di Torino, Torino (Italy), and Adjoin Professor of the University of Illinois at Chicago, IL (USA). His research activity mainly focus on Digital Systems Design & Test, System Dependability, FPGA-based Reconfigurable System Design, Assistive Technologies and ICT for Deaf people. He is also president of CINI: Consorzio Interuniversitario Nazionale per lInformatica, member of the Scientific Committee of the French "Centre National de la Recherche Scientifique" (C.N.R.S.) and Vice-Chair of the IFIP Technical Committee TC 10 - Computer Systems Technology.

Davide Bertozzi got his PhD in Electrical Engineering from University of Bologna (Italy) in 2003. Since 2005 he is Assistant Professor at University of Ferrara (Italy), where he leads the Research group on Multi-Processor Systems-on-Chip, with Networks-on-Chip as the main focus. He has been visiting researcher at international academic institutions (Stanford University) and large semiconductor companies (NEC America Labs, USA; NXP Semiconductors, Holland; STMicroelectronics, Italy; Samsung Electronics, South Korea). Bertozzi was Program Chair of the Int. Symposium on Networks-on-Chip (2008) and of the NoC track at DATE (2010-2012) conference. He is member of the Editorial Board of the IET-CDT, Springer DAEM and ACM TODAES journals. Bertozzi is a member of the Hipeac NoE and was/is involved in many Italy- and EU-funded initiatives (Photonica, Galaxy, NaNoC, vIrtical).

Piero Olivo received the Ph.D. in electronic engineering from the University of Bologna, Bologna, Italy, in 1987. He has been a Full Professor of Electronics with the University of Ferrara, Ferrara, Italy, since 1994. His research interests include the physics, the reliability and the experimental characterization of innovative non-volatile memory cells and architectures.