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

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

Lorenzo Zuolo[†], Cristian Zambelli[†], Rino Micheloni[§], Salvatore Galfano[‡],
Marco Indaco[‡], Stefano Di Carlo[‡], Paolo Prinetto[‡], Piero Olivo[†] and Davide Bertozzi[†]
[†]Dipartimento di Ingegneria, Università degli Studi di Ferrara, Ferrara (Italy)
[‡]Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino (Italy)
[§]PMC-Sierra Italy (Italy) - E-mail: lorenzo.zuolo@unife.it

***Abstract*—Solid State Drives (SSDs) are gaining particular momentum in various frameworks such as multimedia, large data centers and cloud environments. Unfortunately, efficient CAD tools for SSD design space exploration able to assess the optimization of the device microarchitecture w.r.t. the target performance are still missing. This paper tries to close this gap by proposing SSDExplorer, a tool for fine-grained and fast design space exploration of SSD devices. SSDExplorer provides unprecedented insights into the architecture behavior and subcomponent interaction efficiency, while avoiding the need for the actual implementation of an FTL or of key hardware components. This is achieved by the introduction of suitable abstractions of the different components. This is confirmed by the thorough validation of SSDExplorer against a commercial SSD device.**

I. Introduction

Solid State Drives (SSDs) are becoming popular, driven by the relentless growth of the cloud computing, and high performance gaming [1]. The development of an SSD architecture implies the analysis of important trade-offs that, if properly understood, my help in identifying optimal design points meeting target performance requirements. Although SSD hardware prototyping platforms may capture realistic storage system behaviors, they suffer from an intrinsic lack of flexibility [2]. The SSD research community is therefore increasingly relying on sophisticated software tools that enable modeling and simulation of SSD platforms. *Disk emulation tools in virtual environments* [3] use 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 abstract simulation models. Differently, *pure software simulation tools* [4] use trace driven simulators to obtain a steady state performance analysis of the device. However, they often overlook the macroscopic performance implications of key component parameters or subtle microarchitecture-level effects. A Fine-Grained Design Space Exploration (FGDSE) tool is therefore mandatory in the early development stages of an SSD architecture to avoid its over-design.

In this work we propose SSDExplorer, an ad-hoc virtual platform for FGDSE of SSDs. SSDExplorer enables to model all components of an SSD platform, selecting, for each model, the most suitable modeling abstraction level. This broadens the design space exploration capabilities provided by competing tools. Parametric models for often neglected key components such as error correctors, data compressors and NAND Flash memories are considered. SSDExplorer provides accurate performance breakdowns and identification of microarchitectural bottlenecks or unexploited parallelism in the SSD architecture. Moreover, it enables quantitative estimation of the *SSD performance drop introduced by NAND Flash components wear-out*. SSDExplorer accounts for the *performance implications of the Flash Translation Layer (FTL) without requiring its full implementation*. This is achieved exploiting the write amplification factor abstraction [5]. Similarly, the impact of key hardware blocks on the device performance can be projected without need for their actual implementation. *SSDExplorer therefore provides a relatively fast path for accurate I/O performance quantification*. As a result, functional simulation is not natively supported by the tool. Nevertheless, it can be later implemented as a device architecture refinement step with minimum incremental effort. SSDExplorer is fully implemented using the SystemC language, thus enabling to integrate, in a uniform manner, components with heterogeneous modeling abstractions, and to provide a uniform and coherent way to parameterize them. Accuracy of the analysis performed by SSDExplorer has been *validated against a commercial device* (i.e., OCZ vertex 120GB [6]).

This research has been partly supported by the 7th Framework Program of the European Union through the vIrtical Project, under Grant Agreement 288574 and the CLERECO Project, under Grant Agreement 611404.

II. Related Works

Currently, publications aiming at proposing software frameworks able to understand the behavior of an SSD mainly focus on disk *emulation* [3] and disk *trace-driven* simulation software [4], [7], [8], [9], [10], [11], [12].

Yoo et al. [3], propose a disk *emulation* strategy based on a reconfigurable framework able to emulate 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[1]. However, to achieve fast performance estimations, several components (e.g., the processor, the NAND Flash array, 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 [4] and [7] allow SSD performance and power consumption evaluation. Attempts to improve them in order to achieve real performance matching have also been proposed in [11], [12]. However, these tools are still highly abstracted, thus providing an insufficient level of simulation accuracy and realistic components description to perform real FGDSE.

To overcome this weakness, several cycle-accurate SSD simulators have been developed. Lee et al. [8] exploits a global clock simulation for hardware components description. However, it does not allow a full modeling of all the components building an SSD, thus hiding some of the bottlenecks affecting the architecture. Other methods for fast simulation have been proposed in [9], [10]; yet, they also suffer from precision loss due to lack of a complete architectural modeling. *Hardware platform* prototypes have been proposed as well [2], [13]. They enable a precise SSD behavior investigation, although their fixed architecture severely limits exploration of different design solutions (the sole internal firmware modification is allowed).

Overall, available frameworks miss a clear exploration of the performance correlation between the host interface capabilities and the non-volatile memory subsystem, going through all intermediate architectural blocks. To summarize, Table 1 shows the main characteristics

[1] http://wiki.qemu.org/Main_Page

TABLE I. COMPARISON BETWEEN SSDEXPLORER AND OTHER SSD FRAMEWORKS.

Reconfigurable parameters	SSDExplorer Platform	Emulation Platforms	Trace-driven Platforms	Hardware Platforms
Actual FTL (WL, GC, TRIM)	√	√	√	√
WAF FTL	√	No	No	No
Host IF performance	√	√	No	√
Real workload	No	√	No	√
Different Host IF	√	No	√	No
DDR timings	√	No	No	No
Multi DDR buffer	√	No	No	No
Way: Shared bus	√	√	√	√
Way: Shared control	√	No	√	No
NAND architecture	√	√	√	No
NAND timings	√	√	√	√
NAND latency aware	√	No	No	√
ECC timings	√	No	No	√
Compression	√	No	No	No
Interconnect model	√	No	No	√
Core model	√	No	No	√
Real firmware exec	√	No	No	√
Multi Core	√	No	No	No
Model refinement	√	No	No	No
Simulation Speed	Variable	High	High	Fixed

of SSDExplorer in the context of previous work in this field by comparing relevant features of the available simulation frameworks.

III. SSDEXPLORER AT A GLANCE

One of the key concepts that have driven the development of SSDExplorer is the possibility for users to experience a unified, reconfigurable and multi-abstraction simulation environment. To achieve this goal, each block has been written and integrated using the SystemC modeling and simulation environment[2]. SystemC allows designers to cover in a single language several model refinement layers, ranging from the Timed Functional up to the Register Transfer Layer. Thanks to this feature, if a specific block has to be thoroughly investigated, a more accurate model can be easily developed for it, and plugged into the simulation environment without changing any other component. Since the SystemC simulation speed scales inversely to the description level, the accuracy of each model must be wisely selected in order to maximize simulation efficiency.

A. Main simulator goals

SSDExplorer primary goal is accurate I/O performance characterization of an SSD device, capturing the dependency between the SSD performance figures and those of its sub-blocks and their interaction effects. From this perspective, HW/SW components of an SSD that logically belong to the control path require high modeling accuracy, while components belonging to the data path can be just modeled in terms of the introduced processing/storage delays. While this choice limits the functional simulation capabilities of the virtual platform, it ultimately mitigates the impact of FGDSE on simulation speed. Moreover, a detailed implementation of all SSD components might not be available when the SSD architecture design space is explored. For instance, developing a full FTL for the SSD is a time consuming and non-trivial design problem. Its implementation might not be available to hardware designers during FGDSE, thus preventing a realistic assessment of the overall device performance. Similarly, key hardware components such as the error corrector or the compressor may be more conveniently treated as parameterizable or backannotated black-box models during FGDSE. In fact, their impact over SSD performance can be easily projected based on highly abstract quality metrics such as a computation latency or a compression ratio. As a consequence, not only the final component implementation might not be available during FGDSE, but this is not even strictly needed in some cases. Detailed implementations become necessary later in the design flow, when the focus shifts from theoretical I/O performance characterization to actual functional simulation. For the above reasons, SSDExplorer has been designed in order to select the most suitable modeling style for each SSD component to guarantee an accurate quantification of I/O performance, and to tolerate lack of precise implementations of specific HW/SW components without affecting performance estimation accuracy. Fig. 1 shows the SSD architecture template modeled by the SSDExplorer. Three macro-architectural abstraction domains can be identified: *Register Transfer Level models* (RTL), *Cycle Accurate models* (CA) and *Parametric Time Delay models* (PTD). In the next paragraphs, a description of all simulator components is reported, and the motivations behind the selected modeling abstraction in each macro-area of the simulator are also discussed.

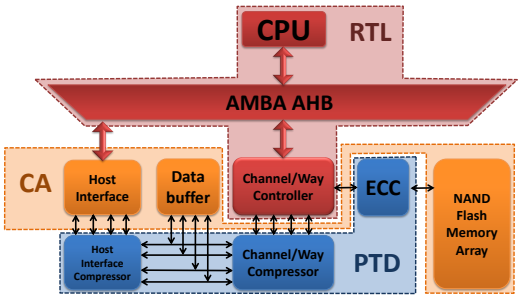

Fig. 1. **Default architecture template modeled by the simulator.**

B. Register Transfer Level models

SSDExplorer tries to offer the highest level of accuracy in modeling those control tasks that determine internal transfer rates within the SSD. The key components that take part in the management of the data flow are the CPU, the system interconnect and the channel/way controller. All these components are involved in the real execution of the SSD firmware (if available) or of its abstracted behavior. As a consequence, an RTL-equivalent design abstraction is mandatory for these components since optimization techniques implemented at this level may highly impact macroscopic SSD performance. This requirement has been partially relieved only for the CPU, for which a pipeline-, pinout- and cycle-accurate modeling style has been considered enough. Even if this solution overall affects the simulation speed, it allows precise modeling of real data and command handling between the SSD components. Moreover, the firmware cost in terms of overall performance drop can be easily collected.

1) CPU: This component models an ARM7TDMI core[3] featuring a 16MB SRAM and a DMA running at 200MHz. It is responsible for the SSD's firmware execution providing an environment for custom FTL development. Thanks to this feature a full SSD firmware can be implemented and interchanged in a plug & play way, which is a very useful feature for platform refinement. The virtual platform is open to the integration of other instruction set architectures, provided the proper wrapper with the standard socket interface supported by the system interconnect is developed.

[2] http://www.systemc.org

[3] http://www.arm.com/products/processors/classic/arm7/index.php

2) System Interconnect: The interconnect model used in SSDExplorer is an AMBA v. 2.0 AHB bus[4] running at the same CPU frequency, a well established standard in nowadays SSD devices [6]. It is configured to support 16 masters and 16 slaves and the arbiter policy is round-robin. For future architectures development, SSDExplorer can also instantiate evolutions of the baseline bus protocol and topology such as the Multi-Layer AMBA AHB interconnect as well as the AMBA AXI protocol. Currently, they are not used in the platform instances under test since they would be over-designed solutions with respect to current SSD requirements. To meet the AMBA AHB protocol requirements in optimized platform instances, both the split and burst transactions are supported, thus hiding wait states and arbitration penalties as much as possible. In many modeling frameworks the bus protocol is abstracted away through behavioral models, hence severely limiting the accuracy on the maximum achievable SSD performance.

3) Channel/Way Controller: To perform read/write operations on the NAND Flash memory array, 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)[5] standard has been exploited for the NAND memory subsystem. From an architectural point of view, the channel/way controller is composed of five macro blocks: an AMBA AHB slave program port, a Push-Pull DMA (PP-DMA) controller, a SRAM cache buffer, an Open NAND Flash Interface 2.0 (ONFI) port and a command translator. The microarchitecture described in [14] has been chosen to mimic realistic functionalities of a channel/way controller in industry-relevant designs. SSDExplorer can be configured with a flexible channel/way interconnection scheme based on state-of-the-art solutions such as the shared bus gang and the shared control gang [15].

C. Cycle Accurate models

The host interface, the DRAM buffers and the NAND Flash memory array have been described at a higher abstraction level. The *key rationale* behind this choice is that these components typically affect macroscopic SSD performance not due to their specific hardware implementation (which is pretty consolidated), but rather through their parameter setting and their latency/throughput figures (which may be even workload dependent). In fact, in an SSD environment, different combinations of the parameters of these blocks may lead to a large performance fluctuation. At the same time, significant performance deviations are incurred if timing accuracy is lost in modeling component behavior. To track in a realistic manner such dependencies, a cycle-accurate description level for these SSD components has been used loosing the signal accuracy. The consequent burdening on the speed of the simulation framework is therefore mitigated.

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: SATA and PCI Express. All SATA protocol layers[6] and operation timings have been accurately validated following the SATA protocol timing directives provided in [16]. Native Command Queuing (NCQ) support has been implemented featuring arbitrary queue length up to 32 commands. The PCI Express interface enables to significantly boost sequential and random operation throughput, and is currently exploited in enterprise SSDs [1]. Fast operations are achieved through the NVMe (Non Volatile Memory Express[7]) protocol that significantly reduces packetization latencies with respect to standard SATA interfaces[8]. All PCIe configurations (i.e., from gen 1 up to gen 3 with variable lane numbers) can be modeled, thus ensuring accurate latency matching [17]. Both interfaces include a command/data trace player which parses a file containing the operations to be performed. During simulation the Host Interface model parses the trace file and triggers operations for the following components accordingly.

To ease the interchange between different host interfaces, a common control architecture based on an AMBA AHB slave port and an external DMA controller[9] able to transfer data from the host interface to the data buffers and vice-versa, is available in SSDExplorer.

2) Data Buffers: This component is used as a temporary storage buffer for read/write data. 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 SDExplorer are modeled with a SystemC customized version of the simulator proposed in [18]. The number of buffers available 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, acting upon a simple text configuration file, which abstracts internal modeling details. Without lack of generality, the results of this work are modeled after a DDR2 SDRAM interface.

3) NAND flash memory array: The fundamental component of an 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. Moreover, the command and the data interfaces of the memory include timing variability as well. To accurately take into account all these effects, a cycle accurate NAND Flash simulator has been exploited [19]. All the experimental results of this work are obtained by modeling a Multi-Level Cell technology whose main characteristics are a t_PROG which ranges from 900 μs to 3 ms, a t_READ of 60 μs and a t_BERS which ranges form 1 ms to 10 ms [20].

D. Parametric Time Delay models

The microarchitectural blocks presented in this section have a twofold feature. On the one hand, they strictly depend on the design choices of SSD vendors (software, hardware, or even mixed solutions can be implemented). On the other hand, their behavior and impact on SSD I/O performance can be easily abstracted by means of well-defined quality metrics. For these reasons, they have been described using parametric time delay models capable of accurately capturing their impact on SSD performance. While this choice enables accurate I/O performance characterization, it prevents functional simulation when such components are instantiated. Nevertheless, at the early design stage, when the internal SSD architecture is defined, functional simulation is actually not required, since priority is given to the

[4] http://www.arm.com/products/system-ip/amba/amba-open-specifications.php
[5] http://www.onfi.org
[6] www.sata-io.org
[7] http://www.nvmexpress.org/
[8] http://nvmexpress.org/wp-content/uploads/2013/05/NVM_Express_1_1.pdf
[9] http://www.open-silicon.com/ip-technology/open-silicon-ip/io-controllers/sata-device-controller/

delivery of target I/O performance with a matched and resource-aware architecture configuration. Only later, when the design is refined, parametric delay models can be replaced by cycle-accurate or even RTL-equivalent models, thus restoring the functional simulation capability of SSDExplorer.

1) Compressor: Nowadays SSD architectures are increasingly using compression in order to reduce the effective amount of data written to NAND Flash memories for wear-out minimization [21]. Since the performance of this component, usually defined as the compression ratio and output bandwidth, is univocally associated with the specific implementation, a parametric timing model can be exploited. SSDExplorer is able to reproduce the timing of a hardware GZIP engine[10] starting from a chosen compression placement. Compressors can be placed either between the host interface and the DRAM buffer (i.e., Host interface compressor) or between the DRAM buffer and the channel/way controller (i.e., Channel/Way compressor).

2) Error Correcting Code (ECC): In state-of-the-art SSD simulators the presence of this component 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. The virtual platform of this work allows the user to choose between different ECC solutions such as BCH and adaptive BCH [22], [23], miming their operation latencies.

E. Simulator Flexibility

SSDExplorer aims to be accurate, hence it requires the precise specification of RTL implementations, CA models or PTD depending on the selected abstraction for each component. Nonetheless, the simulator should not be viewed as hardwired for a specific implementation. In fact, when considering the use of SSDExplorer for modeling platforms of different vendors/users, parametric and cycle-accurate models can be freely replaced to reflect performance/behavior of vendor-specific components while leaving their interfaces unchanged. Also, the high degree of platform parameterization enables to model several architectures without deep changes in the simulation framework. The only high-impact choice would be the replacement of the system interconnect model and of the controller, given their RTL-equivalent modeling styles. While this is unavoidable for the controller, which represents the true IP of each vendor's platform, the currently modeled family of system interconnects represent a *de-facto* standard in industry. So, it is likely to be the right choice across many platforms.

F. Write Amplification Factor (WAF) abstraction

The FGDSE of a large set of SSD architectures is critical when the FTL must be taken into account. Estimating the impact of the FTL software management algorithms (e.g., garbage collection, wear leveling, etc) without developing a custom FTL and therefore burdening on the framework complexity is a complex task. A lot of research effort is reported in this field, and [5] tackles this problem by introducing a lightweight algorithm able to evaluate the blocking time introduced by those algorithms in terms of a Write Amplification Factor (WAF). Thanks to the standard programming model used by the considered CPU model (see Section III-B1), SSDExplorer enables both an actual FTL implementation and its abstraction through a WAF model can. This flexibility is provided to match the incremental development requirements of actual users. For the sake of virtual platform validation, the former approach introduces a very high degree of complexity, and usually several optimizations are hidden for non-commercial environments, hence preventing simulation validation. Therefore, in this work, the latter method has been followed and a reconfigurable WAF algorithm based on greedy policy [5] has been embedded inside the validated SSDExplorer instance.

[10]http://www.inomize.com/index.php/content/index/gzip-hw-accelerator

G. Performance comparison against real SSD

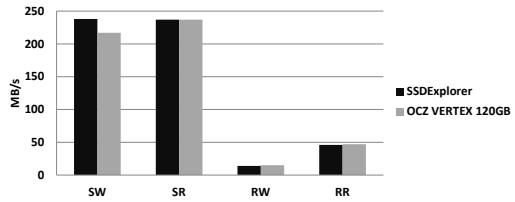

Fig. 2. **Performance comparison between OCZ Vertex 120GB and SSDExplorer in terms of throughput for Sequential Write (SW), Sequential Read (SR), Random Write (RW) and Random Read (RR).**

This section reports a direct comparison between SSDExplorer and an OCZ Vertex 120GB [6], a widely adopted device. This device is based the well-know and documented barefoot controller[11]that can be easily simulated. We resorted to standard IOZone[12] synthetic benchmarks to quantify the I/O performance of SSD devices. A sequential and a random write/read workload with a block size of 4KB is injected inside the simulated disk.
As shown in Fig. 2, for a strictly sequential workload, SSDExplorer tracks the real device performance with an error margin of about 8% for the write operation and 0.1% for the read operation. Differently, when a random traffic is used, the performance deviation from the real disk is of 6% and 2% for writes and reads, respectively. These differences are due to the approximations that are behind the WAF theory [5]. In both cases, the closely matching results reported in Fig. 2 confirm the accuracy of the illustrated methodology. This is even more relevant when we consider that this low error margins can be achieved by SSDExplorer while avoiding the actual FTL implementation, i.e., with a good simulation performance (documented in section IV-C). From this view point, it is useful to remark that SSDExplorer achieves FGDSE capabilities without having all components with RTL-equivalent accuracy.

IV. Experimental Results

A. Optimal design point exploration

TABLE II. SSD configurations.

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

This section shows the SSDExplorer capability in finding the optimal SSD design point (i.e., minimum resource allocation) for a

[11]http://www.indilinx.com/solutions/barefoot.html
[12]http://www. iozone.org/

given target performance. Without lack of generality, in this work the target performance is set by the host interface bandwidth limits. Table II shows a set of representative design points used for this purpose. All results are computed using a workload composed of a sequential write trace and the payload of each single host interface transaction is fixed to 4KB. Moreover, all data have been collected using two different DRAM buffer management policies, which are typically exploited in both consumer and enterprise environments [24]: *caching* and *no caching*. In the former, the SSD controller notifies the end of each transaction to the host system when the data have been moved from the host interface to the DRAM buffers. In the latter the notification is triggered only when all data have been actually written to the NAND Flash memory.

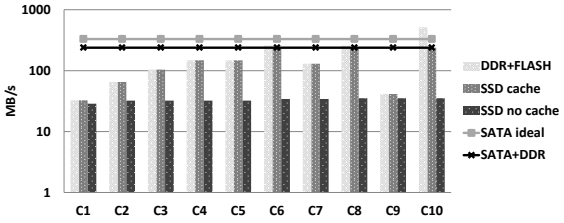

Fig. 3. **Sequential Write: SATA II host interface.**

Fig. 3 shows how the aforementioned architectures exploit the performance of a SATA II host interface. The *SATA ideal* contribution is referred to the ideal throughput achievable by the host interface in a stand-alone fashion. On the contrary, *SATA+DDR* shows a real metric for the host interface performance since it incorporates the time spent by its internal DMA to transfer data from the host system to the DRAM buffers. The *SATA+DDR* performance is considered to be the best-case performance of the SSD architecture as a whole. Starting from this consideration, the best architecture equalization is achieved only by the configurations able to match the *SATA+DDR* contribution with a perfect balancing between the *DDR+FLASH* column (i.e., the time spent by the Flash memory to flush the DRAM buffer and write the data) and the *SSD* column (i.e., the overall disk performance).
When a *cache* algorithm is used, the *SSD cache* column indicates *C6*, *C8* and *C10* as the best candidates since they reach the target performance and saturate the host interface bandwidth. However, when the architecture is taken into account, it is clear that only *C6* represents the right choice since it is the only configuration able to reach the host interface limit with the lower resource consumption. On the contrary, when a *no cache* algorithm is used, the overall disk performance (the *SSD no cache* column) is bounded in spite of the high internal memory parallelism. In such a scenario there is no configuration able to reach the target performance and so, the search for the optimal design point falls on *C1*.
The *key rationale* behind the above performance flattening can be found inside the SATA interface and more precisely, into its limited command depth. In fact, the SATA protocol is able to manage only a maximum of 32 commands at once, and in an SSD exploiting a *no cache* policy, the host interface cannot acquire new commands until the current ones have been written inside the non-volatile memory. In such a way, the internal parallelism provided by the device cannot be exploited, which becomes clear when checking the SSD performance with the *DDR+FLASH* column.
To overcome this limitation and unveil the performance provided by highly parallel SSD configurations, an high speed PCIE host interface encapsulating the NVMe protocol has been explored. Fig. 4 shows

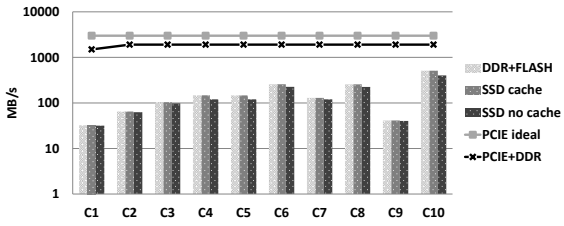

Fig. 4. **Sequential Write: PCIE host interface.**

the achieved results when a PCIE-Gen2 featuring 8 lanes and the NVMe protocol is exploited. Due to the high PCIE speed, the first consideration is that the host interface no longer represents the SSD performance bottleneck. In fact, even the most parallel configuration (i.e., *C10*) is not able to saturate the interface bandwidth. However, the major contribution showed in Fig. 4 can be evidenced by looking at the *SSD no cache* columns. In this case, since the NVMe protocol can handle up to 64K-commands, the SSD internal parallelism can be unveiled and fully exploited. In such a way the performance of these SSDs closely track the overall bandwidth showed by *SSD cache* disks. Clearly, between these configurations a performance gap still exists. Indeed, in the latter the time spent to flush the incoming data to the NAND Flash memories is hidden. Finally, 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 the performance-cost trade-off is optimized, then *C6* becomes again the best candidate since it shows higher performance with lower resource allocation.

B. Performance over NAND Flash wear-out

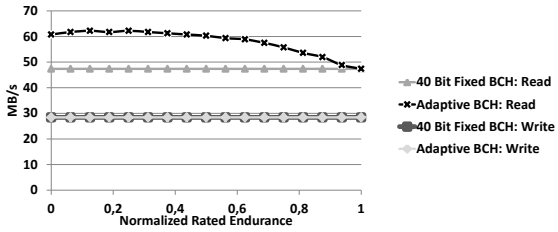

Fig. 5. **Performance drop with respect to Normalized Rated Endurance.**

A key feature of SSDExplorer is the capability to analyze the SSD performances over NAND Flash wear-out. As Flash memory pages wear out, both timing and reliability are influenced thus affecting performance of the whole SSD. In particular reliability loss due to Flash wear out requires the use of complex ECC subsystems representing one of the killing factors for the SSD performance. SSDExplorer is able to simulate ECC activities allowing the user to choose between two ECC schemes: a fixed BCH ECC with correction capability fixed to the worst case condition and an adaptable BCH ECC with correction capability that adapts with the aging of the Flash pages. This second choice exploits a static correction table that correlates the target correction capability with the memory page wear-out, measured by Program/Erase (P/E) cycles. Every time a new page is written, based on the current P/E count the proper correction capability is selected from the table. In order to evaluate the effects of Flash memory wear-out and ECC design choices, the throughput was measured by applying sequential writes and reads on two SSD configurations, having both 4 channels 2 ways and 4 dies,

and differing for the ECC adaptability: the first features a fixed BCH ECC which is able to correct 40 bits whereas the second one features an adaptable BCH ECC which is also able to correct up to 40 bits.
The trends of throughput (Fig. 5) show that, except for the end-of-life, adaptable BCH achieves a remarkable read throughput gain w.r.t. fixed BCH: this depicts adaptable BCH ECC as the best solution between the two choices.
Those throughput differences are mainly caused by ECC. In fact, ECC correction capability influences the read and write operations latency overhead. The encoding operation latency, which is involved in writing operation, is not substantially affected by the correction capability choice. The decoding operation latency, which incurs while reading from the disk, instead, heavily grows with employed correction capability, thus limiting read throughput. In adaptable BCH ECC, correction capability across memory wear-out is lower than the worst case fixed BCH ECC correction capability, thus resulting in lower overhead and higher read performances.

C. Simulation Speed

Since simulation speed is one of the main targets of SSDExplorer, it has been evaluated on 8 different SSD architectures reported in Tab. III. Fig. 6 shows the Kilo-Cycles Per Seconds (KCPS) achieved on an Intel(R) Xeon(R) CPU E5520 clocked @ 2.27GHz with 12GB of RAM, on which a Redhat x86-64 Linux operating system runs. The simulation speed scales inversely to the number of resources instantiated inside the framework.
The sustained simulation speed for reasonable design points (39,7 KCPS for the C4 configuration adopted in [6] [2]) prove definitely the effectiveness of using different hardware refinement models.

TABLE III. SSD CONFIGURATIONS.

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

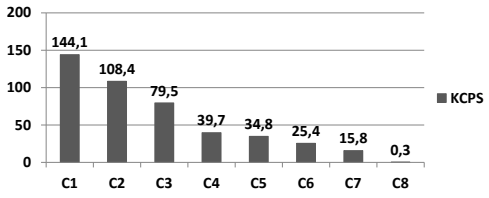

Fig. 6. **SSDExplorer simulation speed with different SSD configurations.**

V. CONCLUSIONS

In this work we presented SSDExplorer, a virtual platform for fine-grained design space exploration of solid state disks. The main aim of SSDExplorer is to provide a ready to use framework which is able to deliver accurate performance breakdown curves of all internal SSDs components. SSDExplorer enables both abstract FTL models and real FTL implementations and is built on top of the standard SystemC programming language. SSDExplorer has been successfully validated against a real device and thanks to the wise engineering of the abstraction levels used to describe its modules, a relative high simulation speed is still guaranteed thus paving the way for its future integration in a complete virtual platform environment.

REFERENCES

[1] R. Micheloni, A. Marelli, and K. Eshghi, *Inside Solid State Drives (SSDs)*, ser. Springer series in advanced microelectronics. Springer London, Limited, 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 *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, 2013, pp. 1–14.

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

[5] 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: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 10:1–10:9.

[6] "Ocz vertex series 120GB SSD." [Online]. Available: http://ocz.com/consumer

[7] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "Flashsim: A simulator for nand flash-based solid-state drives," in *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, 2009, pp. 125–131.

[8] 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 2009 ACM symposium on Applied Computing*, ser. SAC '09. New York, NY, USA: ACM, 2009, pp. 318–325.

[9] H. Jung, S. Jung, and Y. H. Song, "Architecture exploration of flash memory storage controller through a cycle accurate profiling," *Consumer Electronics, IEEE Transactions on*, vol. 57, no. 4, pp. 1756–1764, 2011.

[10] E.-Y. C. Y. University, "A Solid-State Disk Simulator for Quantitative Performance Analysis and Optimization," in *NVRAMOS, Operating System Support for Next Generation Large Scale NVRAM*, 2009.

[11] 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 *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 279–289.

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

[13] 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.

[14] "Evatronix NAND Flash controller ip-core." [Online]. Available: http://www.evatronix-ip.com/ip-cores/memory-controllers/nand-flash.html

[15] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70.

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

[17] A. Athavale, "Implementing pci express designs using fpgas," Accessed, Aug 2013. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1274512

[18] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

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

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

[21] "Data Compression in the Intel Solid-State Drive 520 Series." [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/ssd-520-tech-brief.pdf

[22] C. Zambelli, M. Indaco, M. Fabiano, S. Di Carlo, P. Prinetto, P. Olivo, and D. Bertozzi, "A cross-layer approach for new reliability-performance trade-offs in MLC NAND flash memories," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 881 –886.

[23] M. Fabiano, M. Indaco, S. Di Carlo, and P. Prinetto, "Design and optimization of adaptable BCH codecs for NAND flash memories," *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, vol. 37, pp. 407–419, 2013.

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