
Lorenzo Zuolo1, Cristian Zambelli1, Rino Michielon1, Salvatore Galfano1, Marco Indaco1, Stefano Di Carlo2, Paolo Prinetto2, Piero Olivo1 and Davide Bertozzi1

1Dipartimento di Ingegneria, Università degli Studi di Ferrara, Ferrara (Italy), 44122
2Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino (Italy), 10129
3PMC-Sierra, Milano (Italy), 20100

Abstract—Solid State Drives (SSDs), based on NAND Flash memory technology, are gaining particular momentum as a storage medium in various frameworks such as multimedia, large data centers and cloud environments. Unfortunately, the current offering of CAD tools for design space exploration of SSDs is not able to assess whether the device microarchitecture features inefficiencies and/or is overdesigned for a target performance. This is due to the necessary architecture abstraction needed to target full system functional simulation, or steady-state performance analysis. This paper bridges this gap by proposing SSDExplorer, a tool for fine-grained yet reasonably fast design space exploration of SSD devices. The tool delivers unprecedented insights into the architecture behavior and subcomponent interaction efficiency, while avoiding the need for actual implementation of an SSD architecture. SSDExplorer therefore provides a relatively fast path for accurate I/O performance quantification. As a result, functional simulation is not required at the analysis step. Nevertheless, it can be later implemented as a device architecture refinement step with minimum incremental effort.

1. INTRODUCTION

Nowadays, Solid State Drives (SSDs) are becoming popular, driven by the relentless growth of the computing, and high performance gaming [1]. The development of an SSD architecture implies the analysis of important trade-offs that, if properly understood, may contribute to lighten the design space, and to reduce the prototyping effort. Although SSD hardware prototyping platforms [2] may capture realistic storage system behaviors, they suffer from an intrinsic lack of flexibility with respect to SSD design choices. To tackle this challenge, and to identify optimal design points meeting target performance requirements, 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 in virtual environments [3], and pure software simulation tools [4]. 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 abstract simulation models. The latter category uses trace driven simulators to obtain a steady state performance analysis of the device. However, it often overlooks the macroscopic performance implications of 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. A Fine-Grained Design Space Exploration (FGDSE) tool is then mandatory in the early development stages of an SSD architecture to avoid its overdesign when trying to meet the target I/O performance requirements.

In this work we propose SSDExplorer, an ad-hoc virtual platform for FGDSE of SSD devices. SSDExplorer complements modeling and simulation capabilities of state-of-the-art tools targeting the four main innovations:

• Modeling of all components of an SSD platform, thus broadening 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 in SSDExplorer.

• Delivering unprecedented insights into the performance quantification within the SSD architecture, by means of suitable abstractions. This is confirmed by the thorough validation of SSDExplorer against commercial SSD devices.

SSDExplorer is fully implemented using the SystemC language. SystemC makes it possible to integrate, in a uniform manner, components with heterogeneous modeling abstractions, and to provide a uniform and coherent way to parameterize them.

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 [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 [13]. 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] exploit 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 in [12], [14]. 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.
TABLE I. COMPARISON BETWEEN SSDExplorer AND OTHER SSD FRAMEWORKS.

<table>
<thead>
<tr>
<th>Reconfigurable parameters</th>
<th>SSDExplorer Platform</th>
<th>Simulation Platforms</th>
<th>Trace-driven Platforms</th>
<th>Hardware Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual FTL (WL, GC, TRIM)</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Host IF performance</td>
<td>V</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Real workload</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Buffer size</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DDR timings</td>
<td>V</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi DDR buffer</td>
<td>V</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Way: Shared bus</td>
<td>V</td>
<td>V</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Way: Shared control</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NAND architecture</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NAND timings</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NAND awareness</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOC timings</td>
<td>V</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Compression</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Core model</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Real firmware exec</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Multi Core</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Model refinement</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Simulation Speed</td>
<td>Variable</td>
<td>High</td>
<td>High</td>
<td>Fixed</td>
</tr>
</tbody>
</table>

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, going through all intermediate architectural blocks. Currently, the performance equalization of these chained components is oversized. To summarize, Table 1 shows the main characteristics 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 [15]. 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. However, it is worth highlighting that, since the simulation speed of the SystemC framework scales inversely to the description level, the accuracy of each model must be wisely selected in light of simulator goals 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 datapath 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 FGDSF on simulation speed. Moreover, a detailed implementation of all SSD components might not be available when 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 FGDSF, 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 FGDSF. 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 FGDSF, 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: i) select the most suitable modeling style for each SSD component in light of the goal of an accurate quantification of I/O performance; ii) tolerate lack of precise implementations of specific HW/SW components without affecting performance accuracy to a significant extent by providing suitable modeling abstractions.

Fig. 1 shows the SSD architecture template modeled by the SSDExplorer. Three macro-architectural domains can be identified based on the selected modeling abstraction: 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.

B. Register Transfer Level models

One of the main goals achieved in SSDExplorer is 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 has been considered mandatory for these components. In fact, optimization techniques implemented at this level may highly impact macroscopic SSD performance. This requirement was partially relieved only for the CPU, for which a pipeline-, pinout- and cycle-accurate modeling style was 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 [16] 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) System Interconnect: The interconnect model used in SSDExplorer is an AMBA AHB bus [17]. This solution represents a well established standard in nowadays SSD devices [6]. The modeled bus features the revision 2.0 of the AMBA protocol and runs at the same frequency of the CPU. 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 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. Finally, to meet the AMBA AHB protocol requirements in optimized platform instances, both the AMBA AXI 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 channel/way controller dedicated to these blocks controlled by the CPU with a proper protocol. The Open NAND Flash Interface (ONFI) [18] 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 [19] has been chosen to mimic real-world 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 [20].

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 typify SSD system microarchitecture and their performance is strongly impacted by 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 parameters for DRAM/Memory timings and SSD architecture can impact 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. Modeling and managing a protocol is a complex task, especially for lower protocol layers. Two types of interfaces are implemented in SSDExplorer: SATA and PCI Express. 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. The features of the available host interfaces are:

- **SATA**: all SATA protocol layers [21] and operation timings have been accurately validated following the SATA protocol timing directives provided in [22]. Native Command Queuing (NCQ) support has been implemented featuring arbitrary queue length up to 32 commands.

- **PCI Express**: this host 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 [23]) protocol that significantly reduces packetization latencies with respect to standard SATA interfaces [24]. All PCIe configurations (i.e., from gen 1 up to gen 3 with variable lane numbers) can be modeled, thus ensuring accurate latency matching [25].

To ease the interchange between different host interfaces, a common control architecture based on an AMBA AHB slave port and an external DMA controller [26] 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 SSDExplorer are modeled with a SystemC customized version of the simulator proposed in [27]. The number of buffers available in a SSD architecture is not fixed, but is instead determined 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 of the same order. To account for all these effects, a cycle accurate NAND Flash simulator has been exploited [28]. All the experimental results of this work are obtained by modeling a Multi-Level Cell technology whose main characteristics are a L_\text{LATENCY} of 60 \mu s and a T_{\text{READ}} of 30 \mu s, a T_{\text{WRITE}} of 5 \mu s and a T_{\text{ERASE}} of 3 ms [29].

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, with the internal SSD architecture not yet defined, functional simulation is actually not required, since priority is given to the 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 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 mitigation [30]. 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 an hardware GZIP engine [31] 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), with different performance implications that will be covered in the experimental results. To capture realistic behaviors, a cycle-accurate parameterized model is needed.

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 [32], [33], 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 be viewed as hard-wired 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 architecture options 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 theoretically infinite solid state disk architectures exposes the drawback in developing a custom FTL tailored for each component [4]. At this point, an interesting challenge lies ahead: how to estimate the impact of the software management algorithms usually exploited by a SSD (e.g., garbage collection, wear leveling, etc) without developing a custom one and burdening on the framework complexity. In literature 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). In this work, as discussed in Section III-B1, thanks to the standard programming model used by the processor, both an actual algorithm implementation and its abstraction through a WAF model can be carried out. 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.

G. Performance comparison against real SSD

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 transaction. In the former, the SSD controller notifies the end of each transaction by the host interface to the DRAM buffers. In the latter the notification is triggered only when all data have been effectively written to the N&amp;N Flash memory.

IV. EXPERIMENTAL RESULTS

A. Optimal design point exploration

In this section we put SSDExplorer at work trying to find the optimal SSD design point for a given target performance. The goal is to achieve a matched performance between all SSD components, which ultimately means to provide the target performance with the minimum resource allocation. Without lack of generality, in this work the target performance is set by the host interface bandwidth limits, and all provided results aim at showing which hardware configuration hits this limit. 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 [36]: 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 effectively written to the NAND Flash memory.
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 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 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.

B. Compression

In solid state disks, the compressor has been introduced in order to minimize the NAND Flash wear-out due to massive write operations [30]. However, this technique hides another interesting behavior, i.e., the reduction of the WAF. Indeed, especially for random write workloads, the compressor can impressively boost the overall SSD throughput. By building on the wide design space exploration capabilities of SSDExplorer, the aim of this Section is only to show the macroscopic performance effects introduced by such block. For this purpose, a random write workload has been applied to an SSD with a no cache configuration, and the two compressor positioning modes proposed in Section IV-B have been explored. Starting from the same baseline SSD memory configuration (i.e., 4 channels, 2 ways and 4 dies per way), Fig. 5(a) and Fig. 5(b) show how the overall disk performance scales with respect to the compression ratio, which depends on the characteristics of the workload at hand. By carefully observing such results, four different effects can be evidenced.

The first one is the WAF reduction and the throughput amplification with respect to the compression ratio. The key rationale behind this behavior can be found inside the WAF theory. Indeed, as the compression ratio increases, the amount of written data decreases, thus de-facto limiting the additional write operations that must be performed by the garbage collection algorithm.

The second implication can be evidenced in the performance gap introduced between a compression ratio of 8 and 16. This macroscopic fluctuation is introduced by the latency manager of the NAND Flash memory subsystem since the reduced amount of data does not spread over many blocks. In fact, writing data on different block offsets may significantly impact on the throughput of the device due to the variation of the address decoding timings.

The third effect is the throughput drop experienced in the channel/way compression as shown in Fig. 5(b). This lowering is due to the fixed granularity of NAND Flash program operations. In fact, since only 4KB at once can be written in such devices, a high compression ratio is achieved, the compressor must iterate several times before building a standard 4KB transaction. In this case, the compression delay becomes significant, thus introducing an effective blocking time. It is worth to highlight that when host interface compression is used, as shown in Fig. 5(a), the above limitation does not occur since the DRAM memory can accept any data size for write operations.

Finally, the last implication is the difference between the maximum throughput achievable in the two proposed configurations. The key rationale behind this discrepancy can be found in the used DRAM buffer size. In fact, the main motivation, which justifies a host interface compressor, is the possibility to shrink the DRAM buffer size since the compression reduces the effective amount of data to store. Consequently, as the DRAM size reduces, its access time decreases accordingly, thus enhancing the overall SSD performance. On the contrary, when a channel/way compression is used, the DRAM buffer has to store all incoming data from the host interface, therefore a large buffer has to be employed. Despite the obvious conclusion which promotes the host interface compressor as the best solution, also the channel/way compressor is usually exploited since the compression property of a given data chunk cannot be known in advance. In fact, if the former is used with non-compressible data, the caching capabilities of the SSD are drastically reduced, thus causing frequent blocking times due to DRAM flushing to the NAND Flash memory subsystem.

C. Performance over NAND Flash wear-out

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

![Fig. 4. Sequential Write: PCIe host interface.](image)

![Fig. 5. Random Write throughput wth respect to compression ratio.](image)
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 44 bits whereas the second one features an adaptable BCH ECC which is also able to correct up to 44 bits. The trends of throughput (Fig. 6) show that, except for the end-of-life, adaptable BCH achieves a remarkable read throughput gain w.r.t. fixed BCH; this depicts an 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.

D. Simulation Speed

One of the objectives of the proposed framework is to allow a fine-grained design space exploration of solid state disks by building an ad-hoc virtual platform capable of reasonably high simulation speed. In Tab. III, 8 different SSD architectures have been simulated. Fig. 7 shows the Kilo Cycles Per Seconds (KCPs) achieved on an Intel(R) Xeon(R) CPU E5520 clocked at 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.

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. Thanks to the feature, for the first time an SSD designer can efficiently outline the disk architecture starting from a given target performance and avoid resources misallocation. Moreover, in order to speedup the evaluation phase without lack of generality on performance figures introduced by the SSD firmware, an abstract FITL model based on the WAF has been feed into the simulator. Furthermore, since a standard programming model is exploited, an actual FTL can be still developed and easily interchanged thus making de-facto SSDExplorer the ultimate framework for SSD storage system design. In this work, we want to highlight that SSDExplorer is validated against a real device and thanks to the wise engineering of the abstraction layers 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.

TABLE III. SSD CONFIGURATIONS.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SSD Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1-16-Ram-12-Chan: X4/WAY:1-DIE</td>
</tr>
<tr>
<td>C2</td>
<td>1-DDR-Buf-1-Chan: X1-WAY:1-DIE</td>
</tr>
<tr>
<td>C3</td>
<td>1-DDR-Buf+1-Chan: X1-WAY:1-DIE</td>
</tr>
<tr>
<td>C4</td>
<td>1-DDR-Buf+2-Chan: X2-WAY:1-DIE</td>
</tr>
<tr>
<td>C5</td>
<td>1-DDR-Buf+2-Chan: X2-WAY:2-DIE</td>
</tr>
<tr>
<td>C6</td>
<td>4-DDR-1-Chan: X2-WAY:4-DIE</td>
</tr>
<tr>
<td>C7</td>
<td>4-DDR-1-Chan: X2-WAY:8-DIE</td>
</tr>
<tr>
<td>C8</td>
<td>32-DDR-1-Chan: X2-WAY:16-DIE</td>
</tr>
</tbody>
</table>

REFERENCES


Fig. 6. Performance drop with respect to NAND Flash P/E cycles.

Fig. 7. SSDExplorer simulation speed with different SSD configurations.