Performance and Reliability Analysis of Cross-Layer Optimizations of NAND Flash Controllers[1]

BERTOZZI DAVIDE, Università di Ferrara
STEFANO DI CARLO, SALVATORE GALFANO, and MARCO INDACO, Politecnico di Torino
PIERO OLIVO, Università di Ferrara
PAOLO PRINETTO, Politecnico di Torino
CRISTIAN ZAMBELLI, Università di Ferrara

NAND flash memories are becoming the predominant technology in the implementation of mass storage systems for both embedded and high-performance applications. However, when considering data and code storage in non-volatile memories (NVMs), such as NAND flash memories, reliability and performance become a serious concern for systems' designer. Designing NAND flash based systems based on worst-case scenarios leads to waste of resources in terms of performance, power consumption, and storage capacity. This is clearly in contrast with the request for run-time reconfigurability, adaptivity, and resource optimization in nowadays computing systems. There is a clear trend toward supporting differentiated access modes in flash memory controllers, each one setting a differentiated trade-off point in the performance-reliability optimization space. This is supported by the possibility of tuning the NAND flash memory performance, reliability and power consumption acting on several tuning knobs such as the flash programming algorithm and the flash error correcting code. However, to successfully exploit these degrees of freedom, it is mandatory to clearly understand the effect the combined tuning of these parameters have on the full NVM sub-system.

This paper performs a comprehensive quantitative analysis of the benefits provided by the run-time reconfigurability of an MLC NAND flash controller through the combined effect of an adaptable memory programming circuitry coupled with run-time adaptation of the ECC correction capability. The full non-volatile memory (NVM) sub-system is taken into account, starting from the characterization of the low level circuitry to the effect of the adaptation on a wide set of realistic benchmarks in order to provide the readers a clear figure of the benefit this combined adaptation would provide at the system level.

Categories and Subject Descriptors: C.4 [**PERFORMANCE OF SYSTEMS**]: Design studies

General Terms: Reliability, Performance, Design

Additional Key Words and Phrases: Adaptable memory controllers, ECC, NAND flash memories

ACM Reference Format:
Davide Bertozzi, Stefano Di Carlo, Salvatore Galfano, Marco Indaco, Piero Olivo, Paolo Prinetto, and Cristian Zambelli, 2013. Performance and reliability analysis of Cross-Layer Optimizations of NAND Flash Controllers Providing Differentiated Flash Access Modes for Adaptive Computing *ACM Trans. Embedd.*

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

1. INTRODUCTION

The application of the NAND flash memory technology [Atwood et al. 1997] has faced a surprising increment, far beyond what was expected when it was originally introduced. One popular example are the solid state disks (SSD) that feature the advent of multi-level cell (MLC) NAND flash memories to store a large amount of data in flash [Ouyang et al. 2014]. However, with the flash memory storage capacity that roughly doubles every 18 months, designers face challenging performance and reliability problems [Micheloni et al. 2010; Mielke et al. 2008; Lee et al. 2003; Bez et al. 2003; Irom and Nguyen 2007]. MLC flash memories require high programming time and provide reduced endurance when compared to old single-level cell (SLC) devices. The raw bit error rate (RBER) of a MLC flash memory is around 10^{-6} [Cooke 2007], at least two orders of magnitude worse than the one of a SLC device [Dan and Singer 2003]. These problems are further amplified by the flash file systems that are often stressed by frequent write requests of small amount of data [Di Carlo et al. 2011].

State-of-the-art NAND flash memories are tightly cost-optimized. The internal operations of the memory are mostly defined at design-time based on worst-case scenarios able to comply with the industry standards (i.e., ONFI [ONFI Workgroup 2012]). However, a fixed system configuration based on worst-case scenarios leads to waste of resources in terms of performance, power consumption, and storage capacity. This is clearly in contrast with the request for run-time reconfigurability, adaptivity, and resource optimization in nowadays computing systems [Cardoso and Hübner 2011]. New mobile usage models in today's complex embedded systems require the execution of multiple use cases on the same device and require to adapt to often non-predictive behaviors of today's complex applications [Henkel et al. 2011]. They require seamless integration of safety/time-critical functionalities with non-critical functionalities, each one demanding for different requirements from the storage system [Sampson et al. 2013]. Moreover, NAND flash reliability is not constant; it changes over the device lifetime. This must be taken into account to enable high optimization of these devices. We clearly see a trend toward supporting differentiated access modes in flash memory controllers for MLC NAND flash devices, each one setting a differentiated trade-off point in the performance-reliability optimization space. This trend is confirmed by some commercial devices that already enable additional levels of flexibility selectable by the user at boot-time. Mainly, these devices enable speed/power consumption optimization by changing the memory bus interface speed (e.g., Micron MT29F16G08ABABA NAND Flash), and the storage model (i.e., choosing SLC or MLC writing schemes [Samsung 2012]). Concurrently with these solutions, several researchers focused on the optimization of the flash write algorithms [Liu et al. 2012] to guarantee performance/reliability trade-off and on the use of adaptive ECC schemes to trade-off storage space and performance for higher error correction capability [Chen et al. 2009; Fabiano et al. 2013]. Some studies also investigated how mechanisms that enable applications to store data approximately enable to improve performance whenever high-precision storage is not required [Sampson et al. 2013]. However, a clear analysis of the benefits of combining optimized write algorithms with run-time adaptable ECC schemes is still missing in the literature thus preventing a clear understanding of the benefits a MLC NAND flash controller would achieve by implementing this higher level of configurability. Only a few studies tried to work in this direction. Pan et al. [2011] presented a first attempt of analyzing joined limited adaptation of the flash programming step voltage coupled with programmability of the ECC, while a very preliminary study presented by Zambelli et al. [2012] introduced increased adaptation at the flash physical layer.

The goal of this paper is to perform a comprehensive analysis of the benefits achievable exploiting the run-time reconfigurability of an MLC NAND flash controller through the combined effect of an adaptable memory programming circuitry coupled with run-time adaptation of the ECC correction capability. Run-time reconfigurability is implemented through the definition of a set of access modes, each one setting a specific trade-off between read throughput, write throughput, uncorrectable bit error rate (UBER) and power consumption. Rather than focusing on the architectural implementation of the considered access modes, this paper focuses on the characterization of the performance/reliability tuning range achievable with them. To the best of our knowledge, this is the first time that run-time adaptation is extended to the full non-volatile memory (NVM) sub-system and a comprehensive study to quantitatively analyze the effect of this adaptability considering a wide set of benchmarks is carried out.

The paper first analyzes the trade-off that can be achieved by considering the write algorithm adaptivity and the ECC adaptivity in isolation and then analyzes the combination of the two adaptation mechanisms. For this purpose, a comprehensive modeling and simulation framework has been set up for both the analog and the digital parts of an MLC NAND flash memory sub-system in an homogeneous 45nm industrial technology substrate. Furthermore, the benefit of the defined access modes on the software stack have been evaluated within a simulation framework based on the YAFFS2 (Yet Another Flash File System version 2) flash file system. A set of different software benchmarks, with different requirements in terms of storage system have been simulated, highlighting the benefits they can achieve in terms of performance (i.e., read throughput and write throughput on the flash memory) and power consumption of the full NAND Flash memory subsystem.

The paper is organized as follows: Section 2 and Section 3 respectively propose the result of the characterization of different flash programming algorithms and of an adaptive ECC sub-system in isolation. Section 4 explores the trade-offs proposed by the cross-layer optimization in the NAND memory controller. Section 5 shows the performance of the proposed system on a set of real-life applications and finally Section 6 summarizes the main contributions of this work and concludes the paper.

2. CHARACTERIZATION OF ADAPTIVE FLASH PROGRAMMING ALGORITHMS

The physical management sub-system of the memory controller interacts with the high-voltage (HV) analog circuitry of the NAND flash memory, which is the block in charge of generating the voltage waveforms required to read/program/erase the memory cells. It issues commands to a control FSM or to an embedded microcontroller in the flash device. Without lack of generality in this work we target a 2-bit per cell NAND flash memory [Mielke et al. 2008]. It stores two bits of information per cell by defining a set of threshold voltage levels (V_{TH}) identified by the statistical distributions L0-L3 of Fig. 1.

An erase operation places all cells of a block at level L0. The following program operations set the threshold voltages of the selected cells at one of the three levels L1-L3 according to the data that must be programmed. The cell programing operation is achieved through a standard algorithm named Incremental Step Pulse Programming Standard Verify (ISPP-SV) [Micheloni et al. 2010]. A voltage pulse of predefined amplitude and duration is applied to the gate of each programmed cell. Afterwards, a verify operation takes place. It verifies whether the V_{TH} of the programmed cells exceeds a predefined verify level V_{VFY} as depicted in Fig. 2. Since we work with a 2-bit per cell MLC architecture three verify levels (i.e., VFY1, VFY2, VFY3) are defined and must be verified. If the verify is successful, the cells have reached the desired distribution level and they are excluded from the following pulses through the so-called program-inhibition technique [Micheloni et al. 2010]. Otherwise, another cycle of ISPP is applied after incrementing the programming voltage of ΔISPP.

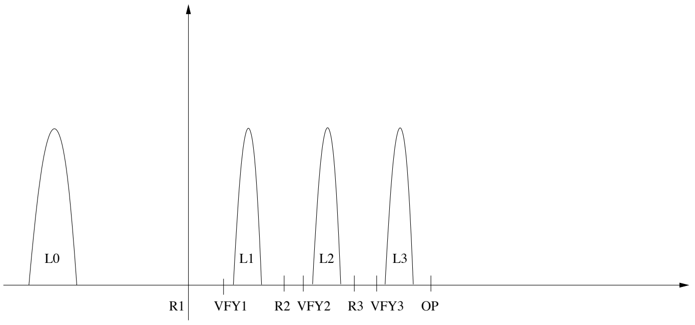


Fig. 1. Threshold voltage distributions in a MLC NAND flash. Read levels (R1, R2, and R3), Verify levels (VFY1, VFY2, VFY3), and over-programming level (OP) are pointed out.

Due to the technological variations, V_{TH} is not perfectly related to the amplitude of the ISPP pulse. There are "fast" cells that reach the verify level with few program pulses and "slow" cells that require more pulses. Both behaviors represent a threat for the reliability of the program operation. In fact, the threshold voltage distributions of the L1-L3 levels significantly deviate from an ideal Gaussian shape. They often cross the distribution read levels (R1-R3) and cause bit errors. A solution for increasing ISPP programming accuracy is the ISPP Double Verify (ISPP-DV) algorithm presented by Micheloni et al. [2010] and Miccoli et al. [2011]. The bit-line voltage of the selected cells is modulated in order to partially decrease the ΔISPP step using a prior verify level with slightly lower voltage than the original verify level. As a result, a more compact threshold voltage distribution can be obtained (see Fig. 2).

Another concern of MLC architectures is to decrease the gap in terms of write-throughput with respect to SLC memories. Both the ISPP-SV and to a larger extent the ISPP-DV require a large number of verify operations per single ISPP step. An interesting solution to avoid unnecessary verify operations is to use the ISPP Reduced Verify (ISPP-RV) write algorithm proposed by Micheloni et al. [2010]. In the ISPP-RV algorithm, the number of verify operations is initially small and is automatically increased based on the number of ISPP steps the algorithm performs (see Fig. 2). This algorithm is able to provide increased programming speed at the cost of reduced robustness against page-errors.

Currently, the flash programming algorithm is set at fabrication time in the flash controller. It is stored in a code-ROM integrated in the same memory die, and executed by an embedded microcontroller. Implementing more than one write algorithm simply requires to store more than one algorithm in the code-ROM by slightly increasing its capacity. Having more than one programming algorithm stored in the code-ROM requires also a mechanism to select the desired algorithm for a transaction or a set of transactions. The ONFI 3.0 standard for NAND Flash memories [ONFI Workgroup 2012] envisions the possibility of implementing both new vendor-specific commands and special commands in case of development of innovative writing methodologies. It could therefore be exploited to implement the write algorithm selection through three dedicated commands such as: 0x80 = Program with ISPP-SV, 0x81 = Program with ISPP-DV, and 0x82 = Program with ISPP-RV. The choice of the programming algorithm can be also implemented through dedicated configuration registers. This approach is consistent with the methodologies exploited in today's NAND flash memory controllers to provide reconfiguration options (e.g., changing the DDR protocol timings [Evatronix 2012], or the storage paradigm [Samsung 2012]).

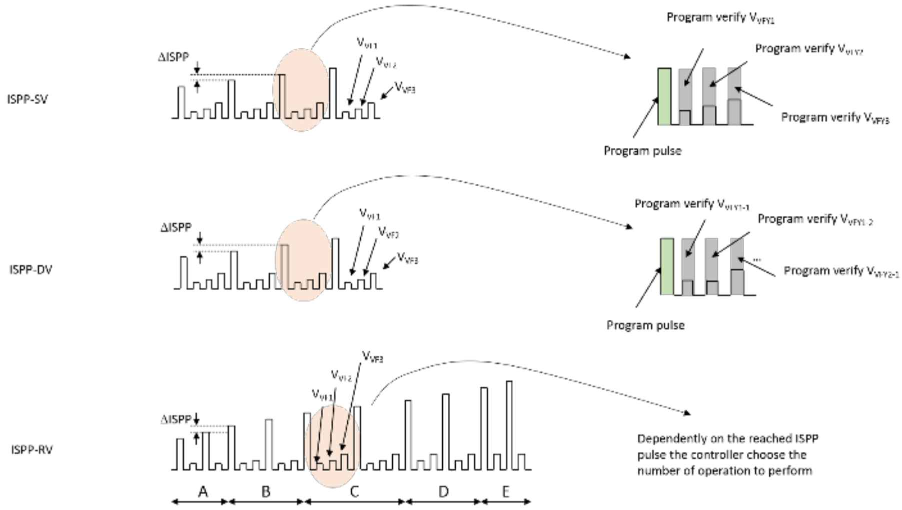

ISPP-SV
ΔISPP
V_{VF1}
V_{VF2}
V_{VF3}
Program verify V_{VFY1}
Program verify V_{VFY2}
Program verify V_{VFY3}
Program pulse
ISPP-DV
ΔISPP
V_{VF1}
V_{VF2}
V_{VF3}
Program verify V_{VFY1-1}
Program verify V_{VFY1-2}
...
Program verify V_{VFY2-1}
Program pulse
ISPP-RV
ΔISPP
V_{VF1}
V_{VF2}
V_{VF3}
Dependently on the reached ISPP pulse the controller choose the number of operation to perform
A
B
C
D
E

Fig. 2. Comparison of the different ISPP algorithms exploited to expose NAND Flash reliability/performance trade-offs

It is worth to mention here that the scaling of the flash cell geometry poses several threats to the reconfigurability of the NAND flash programming algorithms that must be carefully considered. Especially when moving to ultra-scaled devices it is easier to incur in reliability side-effects. However, for technology nodes such as the 4X nm (considered in this work) and the 3X nm it is still possible to leverage on the number of verify pulses to expose trade-offs in the reliability/performance domain. When moving to the 2X nm technology nodes and beyond a refinement of the entire programming algorithms is required due to an increased complexity of the internal NAND flash micro-controller. However, even for these devices it is common practice for the industry to provide on the same chip more than one programming, erasing, and even reading algorithm to be chosen by the NAND flash controller manufacturer acting upon results from test modes that are not accessible by the final users [Micheloni et al. 2010]. This suggests that programming algorithm adaptation will be exploitable also for these devices.

The next sections illustrate the models and the simulations performed to characterize how different programming algorithms impact the raw bit error rate (RBER) and the power consumption of the memory [2].

2.1. Compact and accurate NAND flash Model

Our case study targets a 2-bit per cell NAND flash memory featuring a 45 nm manufacturing process designed for low-power applications. The simulation environment includes two modules: (1) the *high-voltage (HV) sub-system* exploited to generate the voltages required for the programming algorithms (including the verify stage), and (2) a *compact model of the NAND flash memory* with array simulation capability.

[2] The programming of MLC NAND flash also depends on the strategy adopted for loading the data to write into the memory. Without loss of generality, we chose to investigate and explore the ISPP full sequence strategy [Micheloni et al. 2010] instead of the two-rounds one since it reduces the simulation time and provides faster post-processing of the experimental results

The HV module is the analog core of a NAND flash memory. Modifying or reading the number of electrons stored into the floating gate requires the generation of a set of bias voltages with a desired precision, timing and granularity. Moreover, since many voltages have a value larger than the NAND power supply, several charge pumps are required. To obtain highly accurate estimations of the energy consumption of each ISPP algorithm considered in this work, we simulated the program charge pump, the inhibit charge pump, the verify charge pump and the regulators/limiting systems according to the guidelines proposed by Kang et al. [2008]. All blocks have been implemented in HSPICE using STM-45nm technology library [CMP 2012]. The power consumption of each pump extracted from the SPICE simulation during the various stages of the ISPP algorithms has been then fed into a NAND flash power modeling framework based on the equation set provided by Mohan et al. [2010]. As input parameters of the model, we assumed a low-power NAND flash supplied with V_{DD} = 1.8V using an ISPP algorithm starting from 14V to 19V, using $\triangle$ISPP steps of 250mV. The same settings hold for all considered programming algorithms. The simulated HV sub-system has been designed to work with all algorithms. In fact, in a NAND flash device, the timing and sequence of the analog circuitry operations are driven by the embedded microcontroller/FSM by means of a set of interface registers required to generate the enable signals for the charge pumps. Switching from one ISPP algorithm to another does not require a modification of the HV sub-system. It only implies a different sequence of enable signals notified through the same register interface.

Together with the HV module we developed a compact model of the NAND flash cells partially based on Spessot et al. [2010]. It includes variability effects typical of nanoscaled memories and it allows us to simulate array functionalities during a page programming operation. The considered variability effects include: width and length geometrical variations of FG-MOS transistors; non-homogeneity of tunnel oxide and substrate doping; tunneling caused by the electron injection granularity process into the cells floating gate; cell-to-cell interference caused by cross-talk between adjacent floating gates; aging effects due to repeated program/erase cycling, which typically degrades the RBER. All these effects contribute to significantly broaden the gaussian distributions related to the programmed threshold voltage levels within the array, negatively impacting the RBER. A comprehensive description of the considered model is provided in the Appendix of this paper. The model has been validated by fitting it against experimental data collected from Spessot et al. [2010] as showed in Fig. 3, where the cell voltage threshold is plotted during an ISPP operation for a 41nm NAND flash technology. The experimental data provided by Spessot et al. [2010] consider the study of the cell threshold voltage evolution after the application of an ISPP that ranges between 6V and 24V using $\triangle$ISPP=1V, which is different from the specification of our flash memory. Nevertheless, our ISPP range (14V - 19V) represents a subset of the one proposed in Spessot et al. [2010]. Moreover, modifications of $\triangle$ISPP do not change the physical structure of the memory cells [Cooke 2007], thus allowing to fit our model against the data provided Spessot et al. [2010].

It is worth to mention here that the proposed model does not take into account reliability issues due to retention errors. Retention errors are one of the major contributors in NAND flash memory reliability. However, the guidelines provided by JEDEC Solid State Technology Association [2011] about the modeling of the NAND RBER during retention clearly indicate that this value is an offset of the RBER obtained at a specific program/erase cycle according to the following model:

$$\mathrm{RBER}\left(PE, t_{ret}\right) = \mathrm{RBER}_{wr}(PE) + B_o\left(PE^n \cdot t_{ret}\right)^m \tag{1}$$

where t_{ret} is the page retention time measured in hours, PE is the instantaneous program/erase cycles count of the page, m is a coefficient whose numerical value is usually

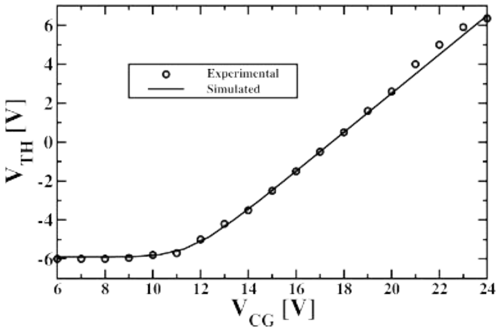


Fig. 3. Fitting results of the NAND flash compact model with experimental data during an ISPP-SV operation featuring 7μs pulses, 1V $\triangle$ISPP.

between 1 and 2, n is a power-law coefficient for program/erase cycles, RBER_{wr} is the error rate observed at $t_{ret} = 0$, and B_o is a scale factor, which depends on the target technological process. Therefore, in order to capture the impact of retention errors a prior characterization of RBER_{wr} is required. Adding the retention contribution will introduce an offset that would not significantly change the definition of the trade-off points while increasing the computational complexity of the model. Moreover, the simulations performed in this paper consider continuous benchmarks that feature short relaxation time between read and write operations, therefore limiting the impact of the retention errors.

2.2. Characterization of programming algorithms

Power consumption, RBER and the average page write time of the flash when using the ISPP-SV, the ISPP-DV, and the ISPP-RV algorithms have been characterized by means of the models presented in Section 2.1. Such parameters are derived as a function of the program/erase cycles of the memory and reported in Fig. 4. Fig. 4a, 4c and 4e provide average results obtained by simulating a random write pattern on a 4KB memory page, while Fig. 4b, 4d and 4f show equivalent results considering all cells of the page programmed at one of the three threshold voltage distributions presented in Fig. 1. This allows us to highlight how the flash performance changes depending on the written data.

Fig. 4a and 4b show the RBER estimated when programming the considered memory page. The two figures clearly show that the choice of a particular programming algorithm produces a significant modification (up to one order of magnitude) of the RBER of the page. The power consumption of the memory device during a program operation with different programming algorithms is instead reported in Fig. 4c and 4d. Power measures do not include I/O pins and digital portions of the flash, which are irrelevant in the comparative analysis. The most power demanding write strategy is the ISPP-DV. It introduces a 4% power consumption increment with respect to the standard ISPP-SV algorithm.This is due to the increased usage of the read charge pump circuitry in the HV sub-system. Nevertheless, it does not represent a major source of power drain in the overall system consumption context. The least power demanding write strategy is instead the ISPP-RV, since the HV circuitry is enabled for a shorter lapse of time due to the reduced number of verify operations. The page write time has been calculated considering a fixed cell verify time (i.e., page read operation) of 30μs. Results reported in Fig. 4e and Fig. 4f show that the fastest algorithm is the ISPP-RV due to the reduced number of verify operations, which generally slow down the writing process. It is worth to point out that the average page write time decreases as the aging increases due to the fastest programming behavior of the memory cells [Micheloni et al. 2010]. This effect is tightly coupled with a reduction of the overall memory reliability since bit errors tend to be more frequent [Mielke et al. 2008]. Table I summarizes the main results obtained from the characterization of the selected device.

Table I. NAND Flash simulation parameters.

Page write time (AVG) @ cycle 1	600us (RV) 800us (SV) 1400us(DV)
Page read time	75us
Block erase time	4ms
Maximum considered P/E cycles	100000
Page Size	4 KB + parity

Note: Programming timings are provided at cycle 1

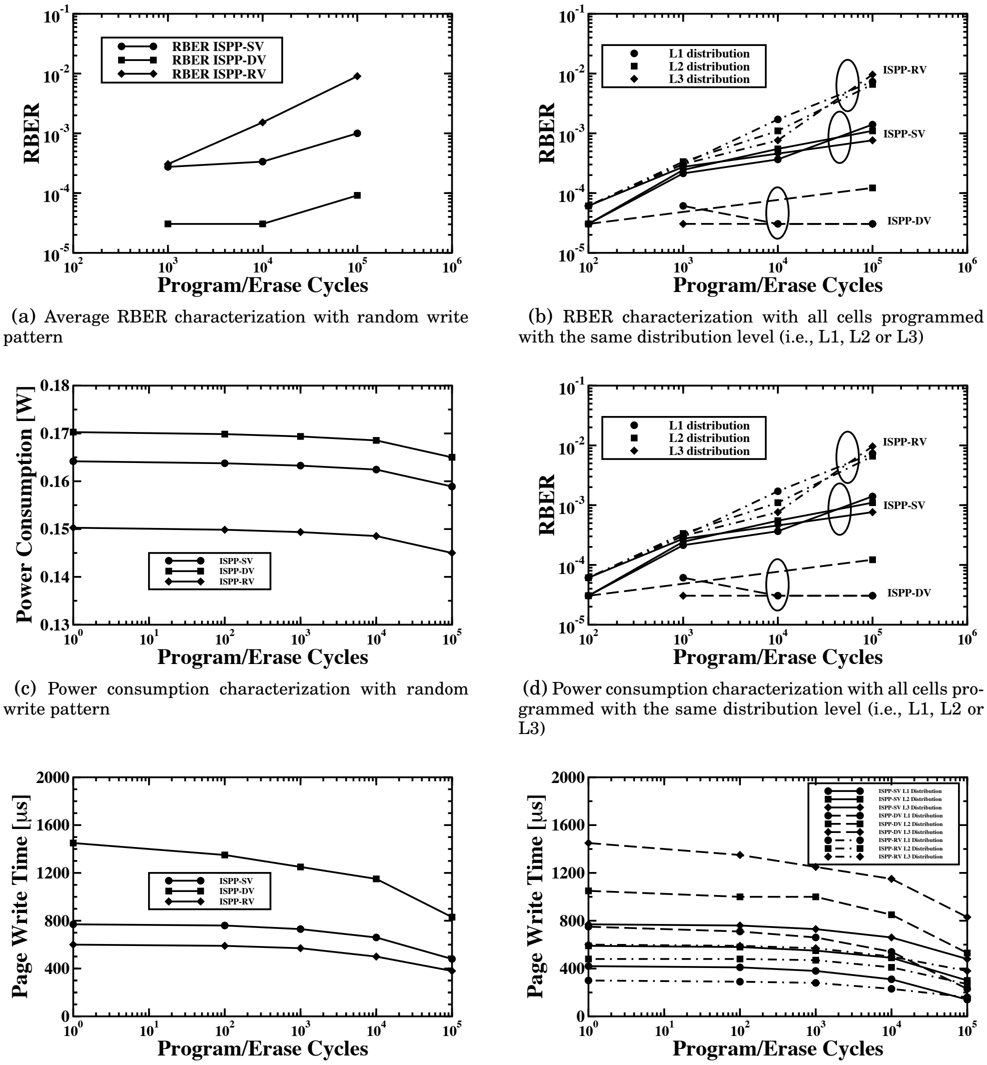
The presented plots clearly show the potentials of an adaptive flash programming algorithm. By selecting a programming algorithm among ISPP-SV, ISPP-DV, and ISPP-RV we can easily trade-off among RBER, power, and write throughput, with only incremental complexity of the memory controller architecture.

3. CHARACTERIZATION OF AN ADAPTIVE ECC SUB-SYSTEM

An adaptive ECC sub-system, enabling to modify the ECC correction capability in a selected range is another way of trading-off NAND flash memory performance, reliability and power consumption. In current SSDs, the ECC calculation time is often successfully hidden when the portion of errors is irrelevant compared the effective dimension of the disk (e.g., at the beginning of disk lifetime). However, when the number of errors due to aging of the NAND flash starts to increase, therefore forcing the usage of policies such as the read-retry to ease the role of the correction codes, the ECC becomes the real system bottleneck. In this situation, multiple errors need to be corrected on multiple NAND flash devices impacting both on SSD read and write throughput [Micheloni et al. 2013].

The ECC sub-system exploited in this paper implements the adaptable Bose-Chaudhuri-Hocquenghem (BCH) ECC architecture presented by Fabiano et al. [2013]. BCH codes belong to the larger class of cyclic codes, which have efficient decoding algorithms due to their strict algebraic architecture [Bose and Ray-Chaudhuri 1960]. BCH codes perform correction over single-bit symbols and better perform when bit errors are not correlated, or randomly distributed. Several studies have reported that NAND flash memories manifest non-correlated or randomly distributed bit errors over a page [Yaakobi et al. 2009]. BCH codes are therefore a perfect choice for correcting errors in these devices. The construction of a BCH code is based on Galois field GF(2^m). Given a finite Galois field $GF\left(2^m\right)$ (with $m \geq 3$), a t-error-correcting BCH code, denoted as $BCH\left[n,k,t\right]$, encodes a k-bit message to a n-bit codeword by adding r parity bits to the original message. The value of m is selected by finding the minimum value that solves the inequality $n-k \leq m \cdot t$, where $n = 2^m - 1$ and $r = m \cdot t$. Whenever $n = k + r < 2^m - 1$, the BCH code is called shortened or polynomial. In a shortened BCH code the codeword includes less binary symbols than the ones the selected Galois field would allow. The missing information symbols are imagined to be at the beginning of the codeword and are considered to be 0.

The hardware BCH architecture exploited in this paper enables to encode/decode a full memory page selecting the desired correction capability t in a given range of values. A detailed description of the employed ECC hardware architecture is out of



(a) Average RBER characterization with random write pattern

(b) RBER characterization with all cells programmed with the same distribution level (i.e., L1, L2 or L3)

(c) Power consumption characterization with random write pattern

(d) Power consumption characterization with all cells programmed with the same distribution level (i.e., L1, L2 or L3)

(e) Average page write time characterization with random write pattern

(f) Average page write time characterization with all cells programmed with the same distribution level (i.e., L1, L2 or L3)

Fig. 4. RBER, power consumption and average write time characterization for ISPP-SV, ISPP-DV, and ISPP-RV algorithms for a page of the flash. For each measured parameter two characterizations are performed based on the type of data programmed in the page. The left column reports an average characterization obtained writing a random pattern in the memory page, while the right column reports a characterization obtain programming all cells of the page at one of the three available distribution levels (L1, L2 and L3) in order to highlight the effect of the data pattern on the measured parameters.

the scope of this paper and can be found in [Fabiano et al. 2013]. The equation that governs the relation among t, RBER and UBER of the flash device for a selected code is:

$$UBER = \frac{1}{n} \sum_{i=t+1}^{n} \binom{n}{i} \cdot RBER^{i} \cdot (1 - RBER)^{n-i} \tag{2}$$

In our specific design, the ECC sub-system has been implemented to work on a full page of the flash (i.e., $k = 4KB$). We considered a target UBER equal to 1E-13, as in Mielke et al. [2008]. Based on equation (2), Table II reports the correction capability required to achieve the target UBER considering the RBERs of the various programming algorithms characterized in Section 2. Clearly the correction capability required to satisfy the target UBER constraints increases over time. As expected, from the reliability standpoint, the worst performance is provided by the ISPP-RV algorithm. This algorithm requires at the end of the life of the device a correction capability of 399 errors per page. This value would require a considerable amount of hardware and performance resources that leads to the conclusion that memory pages using the ISPP-RV algorithm necessarily provide a reduced endurance. This result is in line with current trends for MLC memories endurance that report a typical limit of 10,000 or less program/erase cycles [Yaakobi et al. 2010; Grochowski and Fontana 2012]. For this reason we selected a target maximum correction capability of 93 errors per page corresponding to the requirement of the ISPP-RV algorithm at the end of life. Given the selected value of k and t the resulting code is designed over GF(2^{16}) (i.e., $m = 16$).

Table II. Correction capability required by the ECC to achieve a target UBER=1E-13.

Alg/progr. cycles	**1**	**100**	**1,000**	**10,000**	**100,000**
ISPP-RV	1.000E-06 / 4	6.104E-05 / 13	3.052E-04 / 31	**1.526E-03 / 93**	9.0332 E-03 / 399
ISPP-SV	1.000E-06 / 4	1.000E-06 / 4	2.747E-04 / 30	3.357E-04/ 33	1.000E-03 / 70
ISPP-DV	1.000E-06 / 4	1.000E-06 / 4	3.052E-05 / 10	3.052E-05 / 10	9.155E-05 / 17

Note: Every element of the table reports the memory RBERs for the different programming algorithms (random data pattern) as characterized in Section 2, and the required correction capability.

In the remaining of this section the ECC sub-system will be characterized to show the different trade-offs offered by its programmability. It is worth to mention here that our ECC implementation features a 8-bit parallelism to meet the I/O parallelism of the target flash, and a 8-bit parallelism of the Chien machine allowing 8 evaluations per clock cycle to speed-up the decoding process. Moreover, both the encoder and the decoder are pipelined with the flash memory in order to optimize the performance of the block. Table III reports the area required for this block synthesized using Synopsys DesignCompiler with the STM-45nm [CMP 2012] technology library. The full design works at 100MHz clock frequency.

Table III. ECC encoder and decoder area footprint.

	Area (μm^2)
Encoder	179586.97
Decoder	543625.58

Note: Synthesis has been performed using the STM-45nm technology library.

Let us start with the evaluation of the amount of redundancy (i.e., parity bits) introduced by the ECC. In the worst case (e.g., $t = 93$) the code requires to store $m{\cdot}t = 16{\cdot}93 = 1488\,\text{bits} = 186\text{B}$. This accounts for about 83% of the spare area available on our device that corresponds to 224B per page. ECC parity bits, are not the only extra information stored in a flash memory. High-level functions such as filesystem management, bad blocks management and wear-leveling need to save considerable amount of information. When the spare are is not enough, a certain amount of pages of the flash must be reserved, thus reducing the overall flash capacity. As an example, YAFFS2, the filesystem selected for our analysis, requires to save 18 bytes of information for each data chunk of 2KB. Every page of our flash can store 2 chunks and requires 36 spare bytes. With UBER=1E-13 and $t = 93$, there are 38 spare bytes available to the filesystem that is just enough for the YAFFS2 requirement. If additional functions such as the wear leveling need to be managed, or increased reliability is required, the spare are may become too small. Looking at Fig. 5, if reduced correction capability is required, either because the device is in the early stage of its life, or because a more reliable programming algorithm is applied, the spare area occupation can be reduced up to 78% (4.46% occupation for $t = 4$). This provides a high degree of freedom for the flash memory controller. It is worth to mention here that this does not represent the main parameter to take into account when optimizing the ECC subsystem, nevertheless it is worth to be considered in the optimization of the full flash sub-system design.

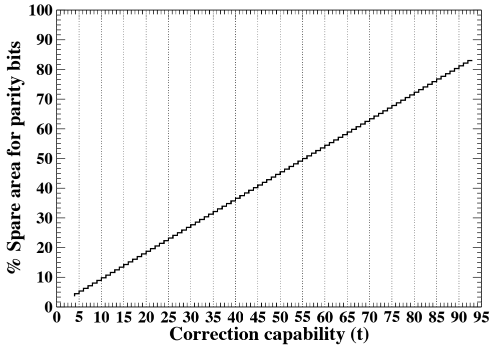


Fig. 5. Percentage of spare area dedicated for storing parity bits as a function of the selected correction capability.

The choice of t also makes it possible to tune the ECC latency. Fig. 6 shows that, carefully tuning the correction capability, the ECC subsystem can significantly save in decoding time compared to the worst case ($t = 93$). Simulations have been performed in the worst case conditions, i.e., t errors injected into the last bits of the page to make sure that the full page must be checked in order to find the corrupted bits. The encoding latency is instead almost constant regardless of the selected correction capability.

Similarly to the ECC latency also the ECC power consumption can be traded-off by carefully selecting the correction capability. Fig. 7 shows that, also in this case, we can save up to $\sim$55% of decoding power consumption when reducing the correction capability.

To conclude the characterization of the designed programmable ECC sub-system, Fig. 8 reports the relation between UBER and RBER for the selected correction mode obtained by plotting equation (2). The figure shows an additional degree of freedom the controller can achieve in which also the UBER can be tuned together with the other parameters.

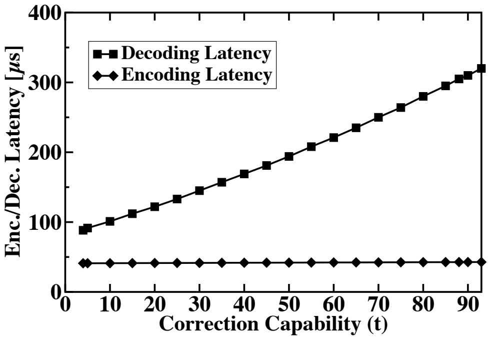


Fig. 6. Worst case ECC encoding and decoding latency. Simulations have been performed at a clock frequency of 100MHz.

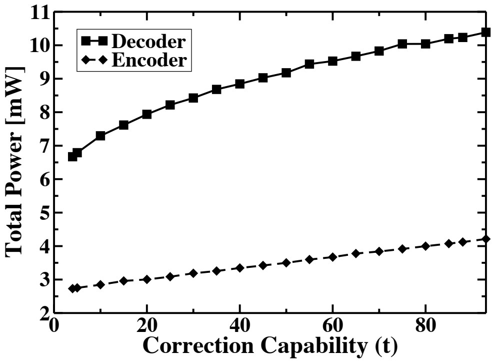


Fig. 7. Worst case ECC power consumption.

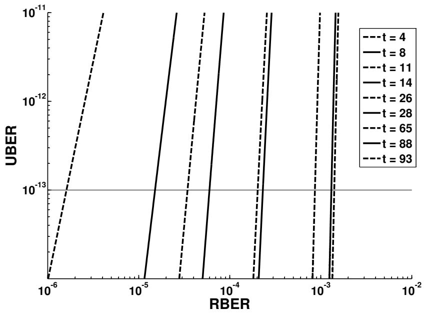
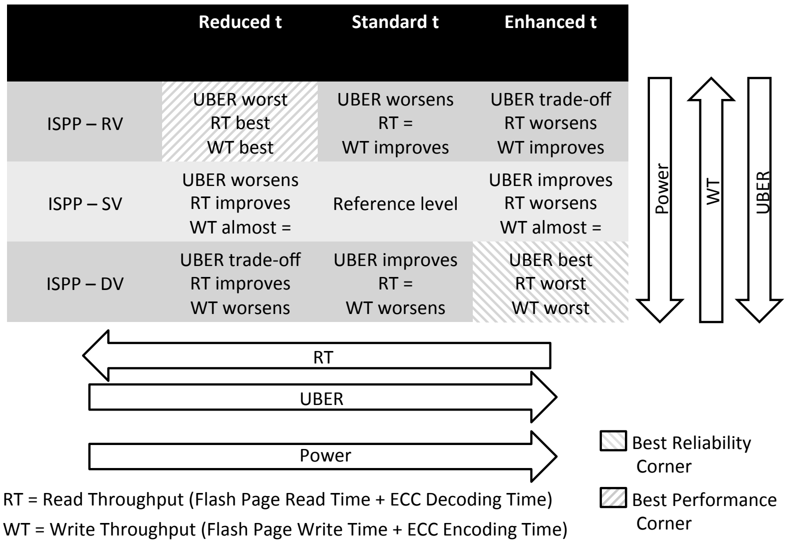


Fig. 8. RBER vs. UBER relationship for the selected code and selected correction modes.

4. CROSS-LAYER OPTIMIZED NAND FLASH ACCESS MODES

So far we have considered the flexibility and the trade-offs that can be achieved by reconfiguring the flash programming algorithm and the ECC sub-system in isolation. However, acting upon their parameters at the same time we want to show that it is possible obtain higher optimization in terms of reliability, performance and power consumption, thus identifying a set of differentiated access modes that can be configured in the memory controller and made available to the software stack.

Fig. 9 provides an overview of how the NAND flash sub-system reacts when selecting different programming algorithms and ECC correction capabilities. Three main parameters of the flash are considered in Fig. 9: (1) the UBER of the flash, (2) the read throughput (RT), i.e., the number of page read requests per second the system is able to serve, and (3) the write throughput (WT), i.e., the number of write requests per second the system is able to serve.



	Reduced t	Standard t	Enhanced t
ISPP – RV	UBER worst RT best WT best	UBER worsens RT = WT improves	UBER trade-off RT worsens WT improves
ISPP – SV	UBER worsens RT improves WT almost =	Reference level	UBER improves RT worsens WT almost =
ISPP – DV	UBER trade-off RT improves WT worsens	UBER improves RT = WT worsens	UBER best RT worst WT worst

▧ Best Reliability Corner

▨ Best Performance Corner

RT = Read Throughput (Flash Page Read Time + ECC Decoding Time)

WT = Write Throughput (Flash Page Write Time + ECC Encoding Time)

Fig. 9. Set of access modes provided when tuning the programming algorithm and the ECC correction capability in a cross-layer adaptation framework.

If we consider the ISPP-SV programming algorithm with an ECC designed for UBER of 1E-13 as a reference operating point, the following behaviors can be foreseen:

- UBER worsens when lower values of t, or programming algorithms with reduced verifications are used.
- WT is mainly affected by the programming algorithm. As pointed out in Fig. 6 the ECC encoding time is almost constant regardless the selected correction capability.
- RT is mainly affected by the selected ECC correction capability that directly affects the ECC decoding time (see Fig. 6). It increases if a lower t is used.
- The combination of reduced t and ISPP-RV represents the best performance corner, but offers the worst reliability.
- The combination of increased t and ISPP-DV represents the best reliability corner, but offers the worst performances.
- In the bottom-left and upper-right access modes of the table, the UBER is adapted acting on the correction strength and the chosen algorithm.

4.1. Access modes characterization

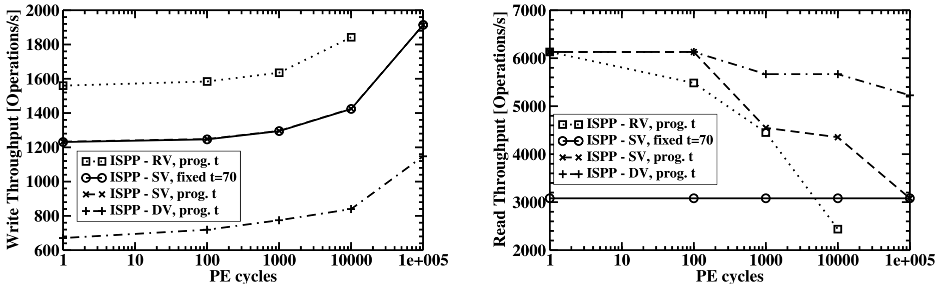
An example of the optimization that can be achieved by selecting the considered access modes is reported in Fig. 10. It shows how the modulation effect of the RT and WT (for a target UBER=10^{-13}) achievable by changing the programming algorithm and the

ECC correction capability, varies over time along with memory aging. The correction capability of the ECC is adapted as aging increases according to Table 3 to preserve the target UBER in spite of memory aging. For the sake of comparison, the figure shows the performance of a non-adaptive controller using the ISPP-SV programming algorithm and a fixed correction capability $t = 70$ required to meet the target UBER at the end of the memory lifetime.

Table IV. Adaptation of the ECC correction capability to the flash aging for different programming algorithms and target UBER

		PE cycles				
UBER	ISPP alg.	1	10^2	10^3	10^4	10^5
10^{-11}	RV	3	11	28	88	>93
	SV	3	3	26	29	65
	DV	3	3	8	8	14
10^{-13}	RV	4	13	31	93	399
	SV	4	4	30	33	70
	DV	4	4	10	10	17
10^{-15}	RV	4	17	34	>93	>93
	SV	4	4	34	37	74
	DV	4	4	12	12	20

Fig. 10a clearly shows that, acting on the programming algorithm, we can modify the write performance of the flash with 30% improvement obtained with *ISPP-RV prog. t* used instead of *ISPP-SV fixed t* at the end of life of the flash. Moreover WT modulation capability is preserved over memory cycling. The non-adaptive and the adaptive ISPP-SV solutions are almost overlapped because the encoding latency is barely affected by the ECC correction capability. Fig. 10b instead shows that we can tune the RT of the system by using ISPP-DV as opposed to ISPP-RV. In these cases the RT can be improved by 83% or degraded by 20% calculated at program/erase (PE) cycle 10k, respectively, and compared to the reference adaptive ISPP-SV solution. Of course, the RT degradation of ISPP-RV is the price to pay for its WT improvement. The comparison of the *ISPP-SV prog. t* curve with the baseline *ISPP-SV fixed t* curve shows that tuning the ECC correction capability over the life of the flash enables a significant improvement of the RT with no penalty on the WT.



(a) Write throughput

(b) Read throughput

Fig. 10. WT and RT comparison among different configurations of the controller for a target UBER=10^{-13}

Fig. 10b also shows that, in the early stage of the memory life, the modulation capability of the RT is marginal. The reason lies in the similar RBER figures of the programming algorithms in fresh devices. On the one hand this means that the RT improvement with respect to the reference case will be achieved only after hundreds of PE cycles. On the other hand, this also means that in fresh devices the WT can be broadly modulated at marginal RT penalty. Overall, Fig. 10 shows a usage model of the access modes: the correction capability is used to preserve a target UBER over the flash life, whereas the programming algorithm is used to trade the WT with the RT. At a given PE cycle a higher RT can be achieved by switching the programming algorithm (i.e., from *ISPP-SV prog. t* to *ISPP-DV prog. t*), and the ECC correction capability (since ISPP-DV needs a lower t to preserve the target UBER with respect to ISPP-SV). The WT can be traded-off similarly. Regardless the selected programming algorithm, Fig. 10b clearly shows that for most of the memory life the non-adaptive approach produces a significant device under-utilization from the RT standpoint.

Other usage models are clearly feasible. For instance, switching from *ISPP-SV prog. t* to *ISPP-DV prog. t*, while keeping t unchanged, minimizes the UBER beyond 10^{-13} leaving the RT unaltered at the cost of the WT. Similarly, switching to *ISPP-RV progr. t* achieves a WT improvement. If at the same time we decrease t the UBER is largely degraded while the RT is improved. Otherwise with a constant t the UBER is degraded to the lower extent but RT is unaltered. Finally, the upper-left access mode in Fig. 9 can be used in those cases where an ultra-low power operating mode is required while, at the same time, largely degrading UBER and therefore application-perceived low reliability are accepted. Approximately storage of data to improve performance whenever high-precision storage is not required has been already investigated in previous studies [Sampson et al. 2013] and the considered service represents a very efficient way for its implementation. In contrast, the lower-right access mode in Fig. 9 provides the best achievable reliability at the cost of increased power consumption and largely degraded performance.

Fig. 11 summarizes the way UBER can be tuned by selecting different ECC correction capability or programming algorithm. Values in the figure are computed considering the RBER of the flash at 10,000 PE cycles, i.e., quite late in the flash lifetime. Similarly to the performance characterization, Fig. 11 shows that we can achieve important trade-offs in the reliability of the access modes, with the possibility of varying the UBER of the NVM system of several orders of magnitude.

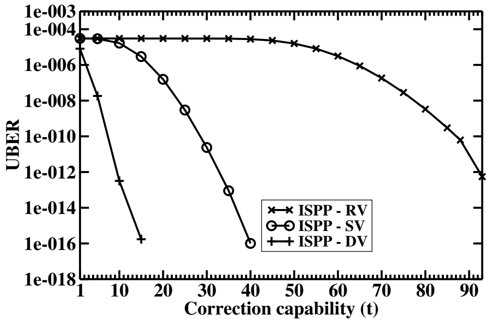


Fig. 11. Trade-off on the storage reliability by selecting different programming algorithms and different ECC correction capability. UBER is computed at 10,000 PE cycles of the flash.

4.2. Implementation of the access modes

In order to properly exploit the advantages provided by the combined adaptation of the flash programming algorithm and the ECC correction capability, a strategy to decide which memory access mode to use at run-time is mandatory. While a complete discussion of this topic is out of the scope of this paper a set of preliminary insights can be provided here. There are essentially two factors that must be considered, at run-time, to properly select the optimal flash storage options: (i) the application reliability/performance/power requirements, and (ii) the memory aging.

The first factor is static for a given application or for selected portions of data of an application. Even if not straightforward, applications can be carefully profiled in order to assign different reliability/performance/power requirements to the different set of data they manage. The application profile can be then exploited to choose the best storage service for each type of information.

We envision in this paper to split the flash memory into different partitions providing different storage services according to Fig. 12.

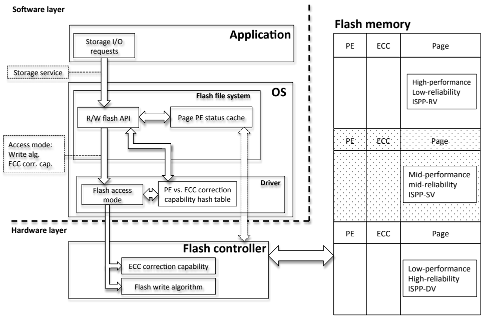


Fig. 12. Exporting storage services to the software layer.

The flash filesystem can therefore be extended in order to provide dedicated API to request different classes of storage services and to properly redirect the data to the partition implementing the requested access mode. Each application can be then instrumented in order to request for each flash memory access the storage service that is more suited for the specific data that is going to be accessed. A single application can therefore benefit from data stored in different partitions with different services in order to optimize the overall reliability/performance. Moreover, considering a different scenario, the choice of the target service may be also handled by the operating system to shield the user from details of the hardware implementation and to avoid erroneous selection of the target service. The operating system may be delegated to select different access modes for an application by exploiting routines that continuously analyze the behavior of the application in order to determine the optimum performance/reliability/power trade-off configuration for the problem, and supervise the program execution. Using program instrumentation gives the programmer flexibility in choosing the system configuration needed for a particular non-functional requirement, while, the implicit approach reduces programming effort and speeds up program development.

While for a given access mode the selected programming algorithm is in general constant over the memory life-time, the ECC correction capability must be continuously tuned at run-time to compensate for the memory aging. Several models in the literature correlate the RBER of a page to the number of performed PE cycles [Sun et al. 2011], and enable to build models fitted on experimental data to compute the best ECC correction capability to apply when a page is programmed. If the PE count is constantly tracked during flash operations it can be exploited to adapt the ECC correction capability according to the selected aging model. In this context, one of the most efficient and easy solutions is to demand this operation to the flash file system in cooperation with the flash driver. At each programming operation the PE count of the target page is incremented and stored together with other file system related information. This value can be then exploited at run-time to select the best correction capability every time the page is programmed. The PE count of each page is cached in RAM using a common practice implemented by the flash file system to store management information. Every time a page must be programmed, this value is retrieved from the cache and it is used to search the best correction capability to apply into a correction capability hash table stored in the flash drivers and containing aging information related to the specific flash technology. This value is then used to encode the target page. Similarly, whenever a page must be read, the same procedure is used to retrieve the correction capability used to encode the page and this information is used during the ECC decoding phase.

5. STORAGE SERVICES AT WORK

To appreciate the benefits of differentiated flash access modes on the execution of a set of real applications we constructed a simulation environment running under the Linux operating system using YAFFS2 (Yet Another Flash File System version 2) as flash file system. The Linux Memory Technology Device (MTD) driver has been instrumented to emulate operations on a NAND flash memory with 4096 blocks of 128 pages, with a page size of 4 kB for a total of 2GB of available storage. YAFFS2 has been also instrumented to trace the list of operations performed through the MTD. Read, write and erase operations have been traced. The log essentially contains information about the sequence of operations, the target page address and the timing. To obtain unbiased measurements of the flash performances the YAFFS2 caches have been disabled.

Several file system benchmarks are available on the Internet (e.g. IOzone [iozone.org 2012], Postmark [Katcher 1997], SPEC benchmarks [Standard Performance Evaluation Corporation 2013], Filebench [Wilson 2008], etc.). In this paper, we selected the Filebench benchmark [Wilson 2008] that provides a large variety of behaviors that can be exploited for our analysis. They either perform simple file I/O operations, or emulate complex I/O activities. Among the available benchmarks we selected three applications:

- *varmail*: has different threads performing create-append-sync, read-append-sync, read and delete operations on the files (representing emails) contained in a single directory;
- *webserver*: opens, reads and closes multiple files in a directory tree while appending data in log file;
- *videoserver*: reads a file set containing videos that are actively served, and writes another file set containing videos that are available but currently inactive.

One of the main characteristic that differentiate the three selected benchmarks is the ratio between the number of read operations ($\#R$) and the number of write operations ($\#W$). This is a critical parameter that influences the type of access mode required by the application to maximize its performance. Table V summarizes this information. It reports the $\#R/\#W$ ratio for each benchmark, as well as the average number of actual read and write operations generated by each benchmark during the simulations. *varmail* is a typical example of write intensive application requiring fast programming of the flash. On the contrary *videoserver* is a read intensive application requiring fast read access to the data stored in flash. Finally *webserver* is between the other two benchmarks and performs a more equalized set of read and write operations to the flash.

Table V. $\#R/\#W$ ratios of different Filebench personalities

Personality	$\#R/\#W$	Avg. $\#R$	Avg. $\#W$
varmail	32.5%	48,536	149,081
webserver	153.5%	100,708	65,581
videoserver	1077.9%	457,138	42,410

Fig. 13,14 and 15 show the opportunities the controller programmability provides to the three applications for a target UBER=10^{-13}. All figures report the overall application throughput, i.e., number of operations (read or program operations) performed on the flash per unit of time. Comparison is again performed with a non-adaptive controller using the ISPP-SV programming algorithm and fixed ECC with t=70. Simulations have been performed in order to emulate a steady state with all flash pages written at least once. This generates an average of one erase operation every 128 programmed pages corresponding to the number of pages in a block.

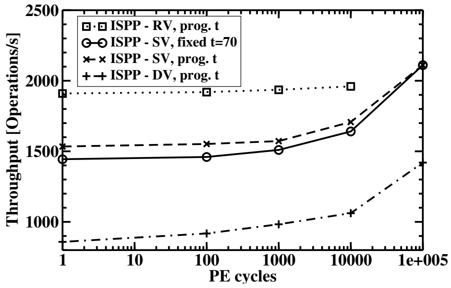


Fig. 13. Varmail throughput for a fixed UBER=10^{-13}

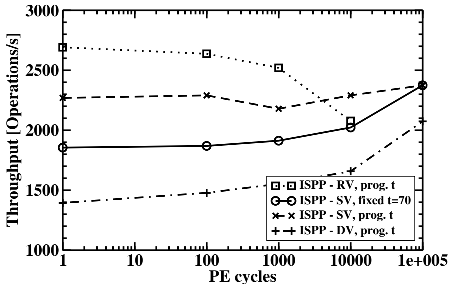


Fig. 14. Webserver throughput fixed UBER=10^{-13}

Looking at Fig. 13, that reports the throughput of *varmail*, it is evident that *ISPP-RV prog. t* enables a significant improvement of the overall performance of the application. This improvement comes however with a reduced endurance of the flash due to the high RBER introduced by this programming algorithm when the number of PE cycles exceeds 10,000.

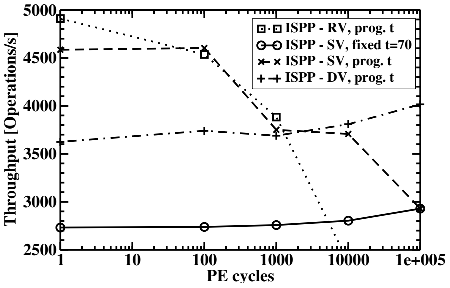


Fig. 15. Videoserver throughput fixed UBER=10^{-13}

If we move instead to the opposite application profile represented by the read intensive *videoserver* reported in Fig. 15 we can notice an interesting result. Looking at the overall flash lifetime, the *ISPP-SV prog. t* seems the best option for this application even if looking at Fig. 10b we could expect better performances from *ISPP-DV prog. t*. The main motivation for this behavior is that the flash programming time is dominant over the flash read time and therefore negatively influences the overall application performances. This opens new opportunities for the proposed controller. In fact, Fig. 15 suggests that not only the ECC correction capability must be adapted to compensate for page aging. In this specific application profile, the ISPP-DV can be selected when the flash reaches more then 10,000 PE cycles to sustain the overall performance and reliability level.

The last situation represented by *webserver* (Fig. 14) obviously provides an intermediate behavior. In this situation *ISPP-DV prog. t* reduces the overall performances and is therefore not a good choice for the application. However, both *ISPP-SV prog. t* and *ISPP-RV prog. t* introduce significant performance improvements.

The analysis performed so far highlights how the proposed adaptation strategy improves the performance of selected applications when mapped to dedicated access modes. The same programmability can be also exploited to provide access modes with different reliability levels as reported in Fig. 16, 17, and 18 for the *videoserver* application. In this comparison we considered a standard reliability service (UBER=10^{-13}), an enhanced reliability service (UBER=10^{-15}) and a reduced reliability service (UBER=10^{-11}).

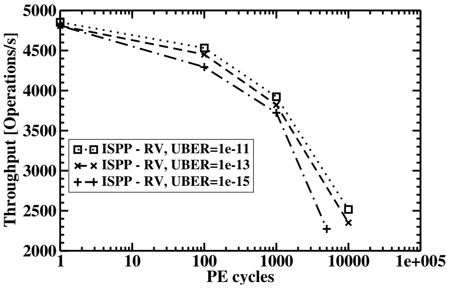


Fig. 16. Videoserver throughput with ISPP-RV program. t at different target UBER

When analyzing the results reported in Fig. 16, 17, and 18 it is important to take into account that the *videoserver* application is a read intensive application. When

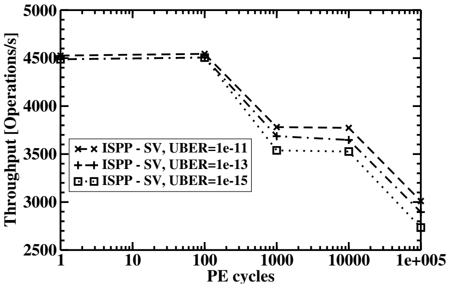


Fig. 17. Videoserver throughput with ISPP-SV program. t at different target UBER

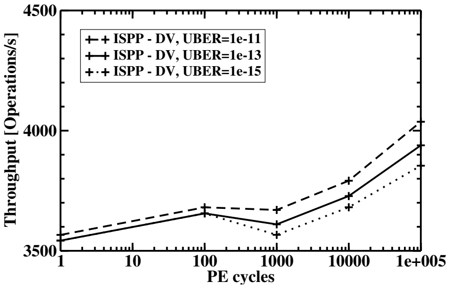


Fig. 18. Videoserver throughput with ISPP-DV program. t at different target UBER

exploiting the *ISPP-RV prog. t* (Fig. 16) and the *ISPP-SV prog. t* (Fig. 17) writing algorithms, that provide reduced reliability compared to the *ISPP-DV prog. t* algorithm, the ECC subsystem is particularly stressed to guarantee error-free data during the intensive read activity of the application. Since the ECC correction capability must be increased with the flash aging, the throughput of the application with these two algorithms decreases over time. Differently, when considering the *ISPP-DV prog. t*, the high reliability of this algorithm strongly relaxes the ECC requirements. This strongly improves the read throughput of the flash at the cost of a decreased write throughput. Write operations become therefore critical for this operation mode and overall the throughput of the application decreases. Nevertheless, it is interesting to note that since the write performance of the flash increases with aging (see Fig. 4e) we observe a slight improvement in the performance of the application at the end of the flash lifetime. Considering the increased reliability service the target choice will be between *ISPP-SV prog. t* and *ISPP-DV prog. t*. In both cases switching to a higher reliability level does not introduce major penalties in the performances. However, *ISPP-DV prog. t* guarantees performances that are more constant over the full flash lifetime. This could be a benefit especially when real-time applications are considered. When moving to the reduced reliability service, instead the choice can be between the *ISPP-RV prog. t* and *ISPP-SV prog. t*. In this case however the choice is a trade-off between performance and memory endurance.

Finally, Fig. 19 reports how the reliability of the memory sub-system can now be traded for the reduced power consumption. In power savings scenarios the functionalities of the system need to be preserved in order to either prolong battery life for portable and embedded systems or to reduce cooling issues in high performance computing systems. Under such conditions the quality of service (QoS) of a target application (i.e., video playback) can be degraded to a minimum acceptance level. This is the case of the *ISPP-RV prog. t* access mode, which can significantly reduce the memory energy consumption by a 10% factor at the beginning of the memory lifetime with respect to the non-adaptive ISPP-SV case.

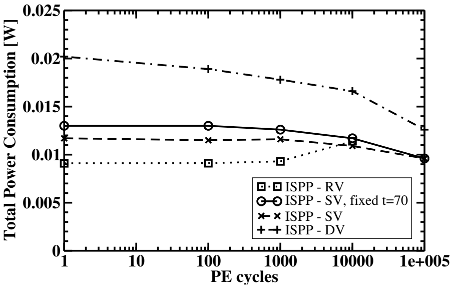


Fig. 19. Average power per operation during the execution of the videoserver benchmark

6. CONCLUSION

In this paper, we demonstrated that combining the selection of different flash programming algorithms, with run-time adaptation of the ECC correction capability in an MLC NAND flash sub-system holds promise of exposing interesting trade-offs between performance, reliability and power for memory access. This enabled us to define differentiated access modes for ultra-high performance, for ultra-high reliability or intermediate trade-off requirements. When put at work for real-life workloads, the user-selectable access modes prove capable of better adapting to application requirements than non-adaptive controllers. By modeling memory endurance effects, we pointed out that the most suitable access mode for each application is not the same through the entire memory lifetime. Based on the results of this paper, the RTL coding of the run-time reconfigurable memory controller will be our future work in order to obtain an adaptive NVM sub-system that can complement the current ongoing efforts in adaptive computing.

7. AUTHORS CONTRIBUTIONS

D. Bertozzi, P. Olivo, and C. Zambelli mainly contributed one the characterization of the flash programming algorithms while S. Di Carlo, S. Galfano, M. Indaco and P. Prinetto focused on the characterization of the adaptive ECC performance and on the setup of the software framework for the benchmark simulations. All authors contributed to the overall storage service analysis.

APPENDIX

The flash model developed in this work is a SPICE-based compact model devised for Monte Carlo simulation of a floating gate transistor. The model captures the threshold voltage evolution of a NAND Flash cell during the ISPP algorithm within a memory array, by adding to the calculated cell's threshold voltage, at each time step of the writing algorithm, the following variability sources:

— *Geometrical variability*: since the transistors within the array do not feature the same geometrical parameters, mainly due to lithographic concerns, a displacement on the channel length (L) and channel width (W) from their nominal values σ_L and σ_W is considered in each Monte Carlo run. These latter parameters feature a Gaussian distribution with mean value equal to 1nm and standard deviation of 0.2 nm. Since the geometry of the transistor affects also the threshold voltage evolution, these parameters are calculated before the definition of the transistor structure to be simulated, therefore affecting the final cell's threshold voltage.

— *Cell-to-Cell Coupling*: the SPICE compact model for the NAND array includes parasitic capacitive couplings between each cell and its first neighbors along the same word- and bit-line. The capacitances are derived from 3D-TCAD simulations, and feature the typical values for a 45 nm technology (i.e., roughly about 20 aF). The cell's threshold voltage calculated at each ISPP step takes into account that the electron tunneling current, and the channel potential of the transistor, deviates from the nominal value by adding a ΔV_{TH} to the voltages exploited for the writing operation.

— *Injection statistics*: the discrete nature of the electronic flow charging the floating gate represents an additional variability source to be considered when dealing with the program operation of nanoscale cells since the statistical process ruling discrete electron injection into the floating gate introduces fluctuations in cell V_{TH} after the application of a writing pulse [Spessot et al. 2010]. On this basis we introduced this additional variability contribution in our compact model for the program operation by adding a displacement from the cell's threshold voltage having the following spread:

$$\sigma_{\Delta V_T} = \sqrt{\frac{q}{\gamma C_{PP}} \left(1 - e^{-\gamma\left(\overline{\Delta_{V_T}}\right)}\right)} \tag{3}$$

where q is the electronic charge, γ is the slope of the tunneling characteristic of the floating gate transistor, C_{pp} is the floating gate capacitance calculated with geometrical variability and $(\Delta_{V_T})^{-}$ is the voltage step magnitude of the ISPP algorithm.

— *Random Dopant Fluctuation (RDF)*: The atomistic nature of substrate doping has been clearly shown to result into a fundamental threshold voltage spread for MOS field effect transistors (MOSFETs) given by:

$$\sigma_{RDF} = 3.19 \times 10^{-8} \times \left(\frac{t_{ox}\,(N_A)^{0.4}}{\sqrt{WL}}\right) \tag{4}$$

where t_{ox} is the tunnel oxide thickness subjected to geometrical variability and equal to 7.5 nm + 0.1 nm, and N_A is the substrate doping of the cell which follows a profile retrieved by TCAD simulations.

— *Oxide Trap Fluctuation (OTF)*: Referring to traps placed at the substrate/oxide interface (where they have the strongest impact on cell V_{TH}) and assuming a Poissonian fluctuation of their number due to process variability, a spread in cell V_{TH} results according to the following:

$$\sigma_{OTF} = K_{OX} \times t_{ox} \times \frac{\sqrt{Q_{OX}}}{\sqrt{WL}} \tag{5}$$

where Q_{ox} is the surface density of traps assumed equal to $10^{-11}\ cm^{-2}$, tox is the tunnel oxide thickness, and K_{ox} is a constant equal to $10^{-6}V \times cm$.

— *Aging effect*: The threshold voltage of a memory cell increases due to charge trapping with the number of write cycles. There are two types of traps that form in the tunnel oxide: interface traps and bulk traps, both of which contribute to the increase in the threshold voltage. It has been shown that both these traps have a power-law relation to the number of cycles on the memory cell [Spessot et al. 2010] as:

$$\Delta Nit = A \times cycle^{0.62} \tag{6}$$

$$\Delta Not = B \times cycle^{0.30} \tag{7}$$

where A and B are fitting constants, cycle is the number of write cycles on the cell, and the terms ΔN_{it} and ΔN_{ot} are the interface and bulk trap densities respectively. In addition to providing this power-law relationship. The authors calculated the values of constants A and B to be 0.08 and 5, respectively for the considered technology. The total threshold voltage increase due to trapping is divided into interface trap voltage shift (ΔV_{it}) and bulk trap voltage shift (ΔV_{ot}), by using the following equations.

$$\Delta Vit = \frac{\Delta Nit \times q}{C_{ox}} \tag{8}$$

$$\Delta Vot = \frac{\Delta Niot \times q}{C_{ox}} \tag{9}$$

where C_{ox} is the capacitance of the tunnel oxide.

All these variability sources contributes to the final threshold voltage value approximately with the following percentile values: geometrical variability (15%), oxide trap fluctuations (15%), random dopant fluctuation (25%), parasitic coupling capacitances, injection statistics, and aging (45%).

REFERENCES

G. Atwood, A. Fazio, D. Mills, and B. Reaves. 1997. Intel StrataFlash memory technology overview. *Intel Technology Journal Q* 4 (1997). https://noggin.intel.com/content/intel-strataflash-memory-technology-overview

R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti. 2003. Introduction to flash memory. *Proc. IEEE* 91, 4 (April 2003), 489–502.

R. C. Bose and D. K. Ray-Chaudhuri. 1960. On a class of error correcting binary group codes. *Information and Control* 3, 1 (1960), 68–79.

J. M. P. Cardoso and Michael Hübner. 2011. *Reconfigurable computing: From FPGAs to hardware/software codesign*. Springer, Germany.

T.-H. Chen, Y.-Y. Hsiao, Y.-T. Hsing, and C.-W. Wu. 2009. An adaptive-rate error correction scheme for NAND flash memory. In *Proceedings of the 27th IEEE VLSI Test Symposium (VTS)*. IEEE, USA, 53–58.

CMP. 2012. CMP Project. web available resource. (2012). http://cmp.imag.fr/

J. Cooke. 2007. The inconvenient truths of NAND flash memory. Flash Memory Summit. (2007). http://download.micron.com/pdf/presentations/events/flash_mem_summit_jcooke_inconvenient_truths_nand.pdf

R. Dan and R. Singer. 2003. Implementing MLC NAND flash for cost-effective, high-capacity memory. M-Syst. White paper. (2003). http://tinyurl.com/o2443mh

S. Di Carlo, M. Fabiano, P. Prinetto, and M. Caramia. 2011. *Design Issues and Challenges of File Systems for Flash Memories*. InTech, Croatia, Chapter 1, 3–30.

Evatronix. 2012. Evatronix NANDFLASH-CTRL NAND Flash Memory Controller. Web available resource. (2012). http://www.evatronix.pl/products/docs.html?id=10\&product=TkFOREZMQVNILUNUUkw=

M. Fabiano, M. Indaco, S. Di Carlo, and P. Prinetto. 2013. Design and optimization of adaptable BCH codecs for NAND flash memories. *Microprocessors and Microsystems* 37, 4–5 (2013), 407–419.

E. Grochowski and R. E. Fontana. 2012. Future technology challenges for NAND flash and HDD products. Flash Memory Summit. (2012). http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120821_S102A_Grochowski.pdf

J. Henkel, L. Bauer, M. Hübner, and A. Grudnitsky. 2011. i-Core: A run-time adaptive processor for embedded multi-core systems. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. ERSA-ADN Publishing, USA. http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.217.7593

iozone.org. 2012. IOzone file system benchmark. Web available resource. (2012). http://www.iozone.org

F. Irom and D.N. Nguyen. 2007. Single event effect characterization of high density commercial NAND and NOR nonvolatile flash memories. *IEEE Transactions on Nuclear Science* 54, 6 (December 2007), 2547–2553.

JEDEC Solid State Technology Association. 2011. Failure mechanisms and models for semiconductor devices (JEP122G). Web available resource. (2011). http://www.jedec.org/standards-documents/docs/jep-122e

Y.H. Kang, J.K. Kim, S.W. Hwang, J.Y. Kwak, J.Y. Park, D. Kim, C.H. Kim, J.Y. Park, Y.T. Jeong, J.N. Baek, and others. 2008. High-voltage analog system for a mobile NAND flash. *IEEE Journal of Solid-State Circuits* 43, 2 (February 2008), 507–517.

J. Katcher. 1997. PostMark: a new file system benchmark. Network Appliance Tech Report TR3022. (Oct. 1997). https://communities.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp-tr3022.pdf

J.D. Lee, J.H. Choi, D. Park, and K. Kim. 2003. Data retention characteristics of sub-100 nm NAND flash memory cells. *IEEE Electron Device Letters* 24, 12 (December 2003), 748–750.

R.-S. Liu, C.-L. Yang, and W. Wu. 2012. Optimizing NAND flash-based SSDs via retention relaxation. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, USA. http://static.usenix.org/event/fast12/tech/full_papers/Liu.pdf

C. Miccoli, C. Monzio Compagnoni, A. S. Spinelli, and A. L. Lacaita. 2011. Investigation of the programming accuracy of a double-verify ISPP algorithm for nanoscale NAND Flash memories. In *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*. IEEE, USA, MY.5.1–MY.5.6.

R. Micheloni, L. Crippa, and A. Marelli. 2010. *Inside NAND flash memories*. Springer Verlag, Germany.

R. Micheloni, A. Marelli, and K. Eshghi. 2013. *Inside Solid State Drives (SSDs)*. Springer, Germany.

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

V. Mohan, S. Gurumurthi, and M. R. Stan. 2010. FlashPower: A detailed power model for NAND flash memory. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, USA, 502–507.

ONFI Workgroup. 2012. Open NAND Flash Interface. Web available resource. (2012). http://onfi.org

J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, and Y. Wang. 2014. SDF: Software-defined Flash for Web-scale Internet Storage Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*. ACM, New York, NY, USA, 471–484. DOI:http://dx.doi.org/10.1145/2541940.2541959

Y. Pan, G. Dong, and T. Zhang. 2011. Exploiting memory device wear-out dynamics to improve NAND flash memory system performance. In *Proceedings of the 9th USENIX conference on File and storage technologies (FAST)*. USENIX Association, Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=1960475.1960493

A. Sampson, J. Nelson, K. Strauss, and L. Ceze. 2013. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 25–36.

Samsung. 2012. Samsung KFG4GH6x4M 4Gb Flex-OneNAND M-die Datasheet. (2012).

A. Spessot, A. Calderoni, P. Fantini, A. S. Spinelli, C.M Compagnoni, F. Farina, A. L. Lacaita, and A. Marmiroli. 2010. Variability effects on the VT distribution of nanoscale NAND Flash memories. In *IEEE International Reliability Physics Symposium (IRPS)*. IEEE, USA, 970–974.

Standard Performance Evaluation Corporation. 2013. SPEC Benchmarcls. Web available resource. (2013). http://www.spec.org

H. Sun, B. Wood, and P. Grayson. 2011. Qualifying reliability of solid-state storage from multiple aspects. In *Proceedings 7th IEEE International Workshop on Storage Network Architecture and Parallel I/O (SNAPI)*. IEEE, USA. http://storageconference.org/2011/Papers/SNAPI/1.Sun.pdf

A. Wilson. 2008. The New and Improved FileBench. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST)*. USENIX, USA. https://www.usenix.org/legacy/events/fast08/wips_posters/wilson-wip.pdf

E. Yaakobi, J. Ma, A. Caulfield, L. Grupp, S. Swanson, P. H. Siegel, and J. K. Wolf. 2009. Error correction coding for flash memories. Flash Memory Summit. (2009). http://www.bswd.com/FMS09/FMS09-201-Yaakobi.pdf

E. Yaakobi, J. Ma, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf. 2010. Error characterization and coding schemes for flash memories. In *Proceedings of the IEEE GLOBECOM Workshops (GC Wkshps)*. IEEE, USA, 1856–1860.

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