RZ 3795 (# Z1101-001) 01/14/11
Computer Science 11 pages

Research Report

OFC: An Optimized Flash Controller for Enterprise Storage Systems

Roman A. Pletka[1], Maria Varsamou[2], Mathias Bjoerkqvist[1], Theodore Antonakopoulos[2], Peter Müller[1], Robert Haas[1] and Evangelos S. Eleftheriou[1]

[1]IBM Research – Zurich
8803 Rüschlikon
Switzerland
E-mail: {rap,mbj,pmu,rha,ele}@zurich.ibm.com

[2]Department of Electrical and Computer Engineering
University of Patras
26500 Patras
Greece
E-mail: {mtvars,antonako}@upatras.gr

OFC: An Optimized Flash Controller for Enterprise Storage Systems

Roman A. Pletka[†], Maria Varsamou[‡], Mathias Bjoerkqvist[†], Theodore Antonakopoulos[‡], Peter Müller[†], Robert Haas[†], and Evangelos S. Eleftheriou[†]

[†]IBM Research - Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland
{rap,mbj,pmu,rha,ele}@zurich.ibm.com
[‡]Department of Electrical and Computer Engineering, University of Patras, 26500 Patras, Greece
{mtvars,antonako}@upatras.gr

Abstract

NAND Flash technology has become the prime candidate for future high-performance enterprise storage applications. However, the architecture and design of Flash controllers have been primarily geared to consumer-market requirements, so that they typically neither sustain high write IOPS and short latencies nor offer the endurance and reliability required for enterprise storage applications. We present a flexible Flash controller architecture called Optimized Flash Controller (OFC) that fulfills all enterprise storage requirements and also serves as a generic platform to further improve the performance of Flash management algorithms. We show the interplay between hardware and firmware and its performance impact on a Flash controller. So far, the relationship between performance behavior resulting from the dependency on prior workloads and the actual controller design could only be observed from a black-box perspective. Further, our results for sequential and pseudo-random read/write workloads show how the number of relocated pages affects the overall performance. They are consistent with existing write amplification models and can be used to assess other Flash management algorithms.

The OFC is built on an FPGA evaluation board and includes Flash-channel controllers, an embedded processor, and SDRAM. Owing to its flexible architecture, new Flash-management algorithms can easily be developed in software and tested on real Flash memory devices. The controller firmware is built as a Linux kernel module, avoiding expensive context switches. Moreover, we developed a Flash-channel simulator that allows kernel code to be tested in user space using User-Mode Linux.

The current OFC version supports four Flash channels and two pipelined dies per channel, reaching a total capacity of 32 GiB. Measurements on our prototype show that we can achieve a maximum sustained sequential throughput of 115 MB/s reading and 72 MB/s writing. Moreover, we achieve a maximum sustained 4 KiB random performance of 24 KIOPS reading and 8.5 KIOPS writing.

I. Introduction

In the past, NAND Flash memory technology has mainly been driven by the consumer market so that today price and performance characteristics make it an interesting candidate to bridge the growing gap between DRAM and HDDs in enterprise storage systems. Especially the introduction of multi-level cells (MLC), which, in contrast to single-level cells (SLC), allows to store more than one bit in a cell led to a boost in storage density and concomitant increase in the capacity of such devices. The absence of moving parts in Flash enhances its ruggedness and latency, and reduces power consumption; however, the operation and structure of Flash pose specific issues in terms of reliability and management overhead, which can be addressed in the hardware or software of a Flash controller.

Typically, NAND-based Flash memories are organized in terms of blocks, with each block consisting of a fixed number of pages. Unlike on HDDs, a Flash memory location must be erased before a write operation. Writes can be done on a page or sub-page level, but erase operations can only be performed on entire blocks. Moreover, manufacturers recommend to program the pages in a block consecutively to reduce write disturbance. The limited endurance in terms of program/erase (P/E) cycles, today on the order of 3 k to 10 k for MLC and 50 k to 100 k for SLC devices, requires careful monitoring of their distribution over the entire device. For illustration, Figure 1 shows the typical raw bit error rate (RBER) of the worst page in a block when writing random data for SLC NAND Flash memories from two different manufacturers.

The need to erase before writing and the high variance in the update rate of pages led to the introduction of a write out-of-place policy that entails a mapping table from logical to physical addresses. Accordingly, higher-level Flash management functions, such as address mapping, garbage collection, write-page allocation, and wear leveling, which can differ depending on the targeted application area. On the other hand, reliability issues caused by the limited endurance and retention as well as cell disturb characteristics are rather well understood [1], [2],

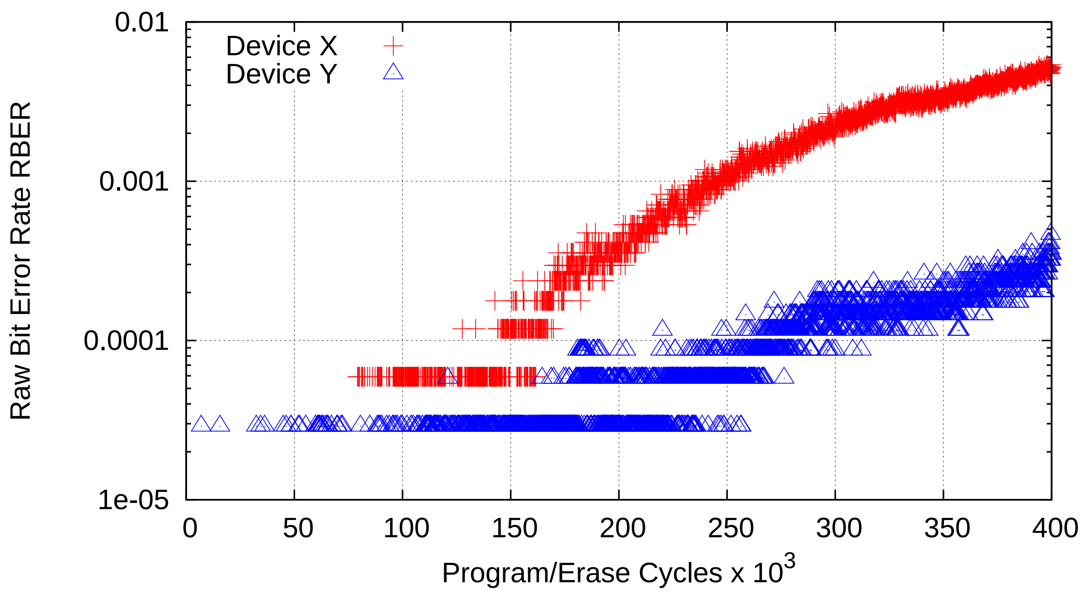


Fig. 1. RBER for the worst page in a block after erasing and writing random data as a function of the P/E cycle count. Device X has a page size of 2112 Bytes, and device Y 4320 Bytes. Note that many pages in a block are behaving significantly better than the worst page.

and can be addressed, to a certain extent, with a first-level error-correction coding (ECC) scheme directly built in hardware, Finally, the standardized interfaces to access Flash chips, such as the Open NAND Flash Interface (ONFI) 1.0/2.x [3], allow a hardware implementation that is agnostic to vendor-specific technology.

In this paper, we present a combination of hardware and software that results in high performance and provides the necessary flexibility to address different application areas, ranging from SSD-like tier-0 storage to cache applications.

The remainder of this paper is structured as follows. First, a brief overview of related work is given in Section II. Then Section III introduces the OFC hardware architecture. Section IV presents the Flash controller firmware of the OFC and highlights general trade-offs in such a firmware design. Section V presents the OFC performance under various workload conditions. The findings and conclusions are then discussed in Section VI.

II. Related Work

In [4], Birrell et al. studied high-level design choices for SSD architectures that advocate page-level mapping of logical to physical addresses. SSD architectures have been evaluated with simulations to analyze the sensitivity to workloads [5], [6], the benefits from internal parallelism [7], and the analysis of write amplification [8]. Flash management algorithms have also been studied by means of competitive analysis [9].

Some work is based on real hardware: Simple Flash test platforms for the evaluation of Flash chip characteristics such as reliability and latency have been introduced in [2], [10], [11]. These platforms are not suitable for a high-speed SSD controller as they are lacking Flash management capabilities. In [12], a NAND Flash controller architecture that uses two channels and command interleaving and reaches a random write performance of up to 3 KIOPS has been shown. Another dual-channel architecture has been presented in [13], showing that database transaction processing can be accelerated by up to one order of magnitude when using Flash instead of disks to improve the random read performance. Another approach is the Flash research platform [14] which utilizes Flash DIMMS connected to FPGA's acting as a controller. The Flash is operated at low frequencies and ONFI2 is not supported. Flash management algorithms are currently being implemented in user space on the host. No random/sequential read/write performance results has been shown so far.

Flash-based storage devices targeting the enterprise level typically utilize a PCIe interface to reduce latency. Note however that with a block device interface the intrinsic Flash properties cannot be fully exploited from the application level. The FusionIO ioDrive [15] includes Flash channel controllers and processes the data path in hardware. It has no controller to execute Flash management tasks. Hence the control path completely runs in a device driver on the host. This has the advantage that available resources from the host, typically built from commodity hardware, can be utilized instead of the limited resources in the embedded environment, but requires significant processor and

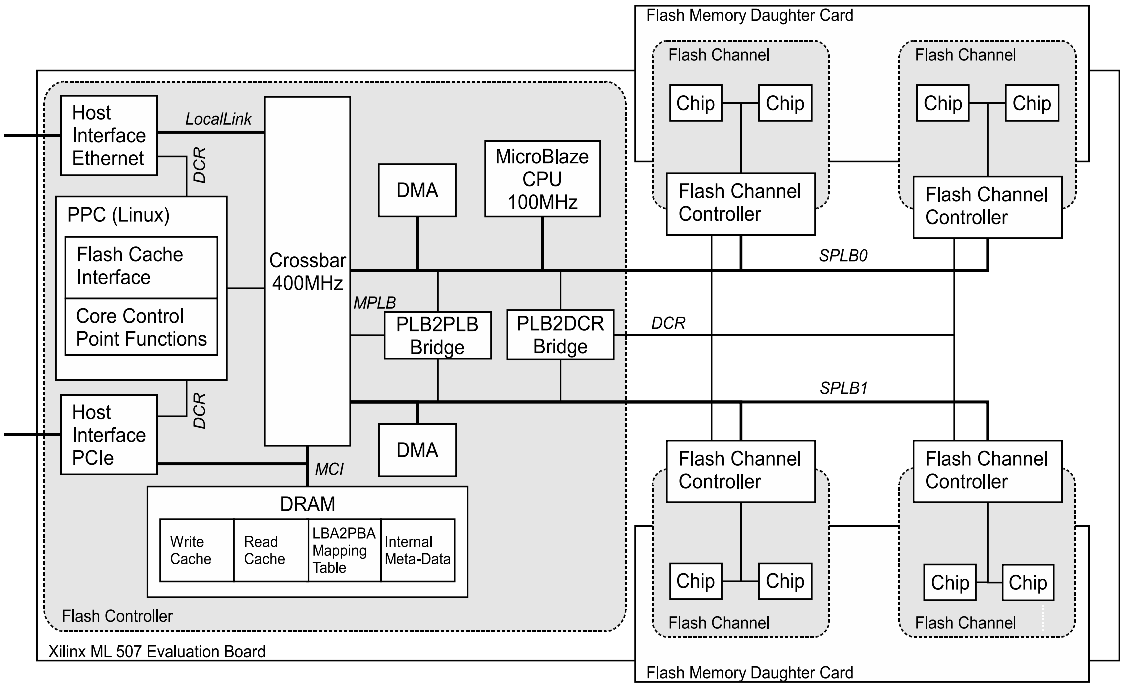


Fig. 2. OFC hardware overview.

memory resources from the host. In contrast to the ioDrive, the TMS RamSan-20 [16] has an embedded processor which performs all Flash management algorithms directly in the device.

III. OFC Hardware Architecture

The basic unit of the OFC hardware architecture is the Flash channel. Each Flash channel consists of a number of Flash chips and a Flash Channel Controller (FCC). Each Flash chip is attached to a Flash channel bus that is based on the Flash interface used (e.g., ONFI), and the FCC is used to interact with the various Flash chips, to read/write to/from their internal data buffers and to access the OFC internal bus. To increase the maximum data rate supported by the OFC, four parallel Flash channels are used. All Flash channels are interconnected using the Peripheral Local Bus (PLB) of the CoreConnect architecture. As shown in Figure 2, the OFC uses two separate PLB busses and a high-speed crossbar switch for data transfers between the OFC's main memory (DRAM) and the various Flash channels. Data transfers and information exchange with the OFC firmware are supervised by a 32-bit RISC microcontroller (Microblaze), whereas data transfers are performed by dedicated DMA engines, one per PLB. In the current setup, the OFC is configured with SLC NAND Flash memory chips with 8192 blocks per die, each block having 128 pages with 4320 Bytes. Two dies are used per channel to reach the total capacity of 32 GiB.

The maximum data rate per Flash channel is mainly determined by the Flash technology used. If R is the data rate at the Flash interface, B the number of bytes per Flash page, T_W and T_R the time required for writing/reading a page to/from the Flash cells, the maximum data rates for read and write are $\frac{RB}{B+RT_R}$ and $\frac{RB}{B+RT_W}$, respectively. Typical values for SLC Flash chips are: $B = 4320$ bytes, $R = 166$ MB/s, $T_W = 200$ μs and $T_R = 25$ μs. As the time required for accessing the Flash cells is comparable to or higher than the data transfer time (e.g., $T_W \gg T_R \geq B/R$), one method used for improving the maximum data rate per Flash channel even further, especially during write, is pipelining. When pipelining is used, several commands are executed simultaneously on different Flash chips on the same Flash channel, as long as non overlapping data transfers are guaranteed.

Each FCC consists of various Finite State Machines (FSMs) that support the ONFI-1.0 and ONFI-2.0 interfaces. For all high-speed data transfers, the ONFI-2.0 FSMs are used exclusively. A local FIFO is used for adapting the different data rates between the PLB bus and the Flash channel. The OFC has been prototyped using the Xilinx Virtex-5 (XC5VFX70T) FPGA on the ML507 board [17]. Table I presents the complexity of each module of the OFC architecture.

The maximum supported data rate at the entire system level is determined by the maximum transfer rate of the internal bus. Because of the speed grade of the FPGA used in the prototyping setup, the maximum data rate achieved

TABLE I
MODULE COMPLEXITY

Module	FPGA Slices
ONFI 1.0 (4 channels)	1000
ONFI 2.0 (4 channels with pipeline)	2680
Microblaze	94
DDR2 SDRAM	2884
CoreConnect and DMAs	1067

at each PLB bus is 100 MB/s, which limits the maximum achievable data rate. Measurements under saturation loading conditions revealed that a maximum raw rate of 30 kIOPs for read and 20 kIOPs for write operations at 4 KiB can be achieved, when the additional processing overhead is minimized. It has to be emphasized, that because of its modular structure, the OFC architecture can easily be extended to support a much larger number of Flash channels, which results in much higher data rates supported. Typical implementations of PLB on high-speed FPGAs support rates of up to 400 MB/s, whereas in ASIC implementations, 800 MB/s have already been demonstrated.

IV. FLASH CONTROLLER FIRMWARE

This section starts with an overview of the Flash controller firmware and then discusses design decisions that are crucial for achieving a high random write performance: Specifically, the logical-to-physical address mapping granularity, the allocation of free pages for write requests, garbage collection, and wear leveling are addressed.

Figure 3 illustrates the software architecture of the Flash controller firmware. A Linux v2.6.30 kernel runs on the embedded PPC 440, and the Flash controller firmware has been implemented as a loadable kernel module to minimize context switches and to have direct access to hardware resources. We removed a small memory region at the end of the DRAM space (48 MiB) from the kernel-managed memory. The controller firmware can only access this region using IO remapping. The largest part of this region is dedicated to the actual data being transferred from/to the host interface and will therefore not be accessed by the controller firmware at all. Another region is

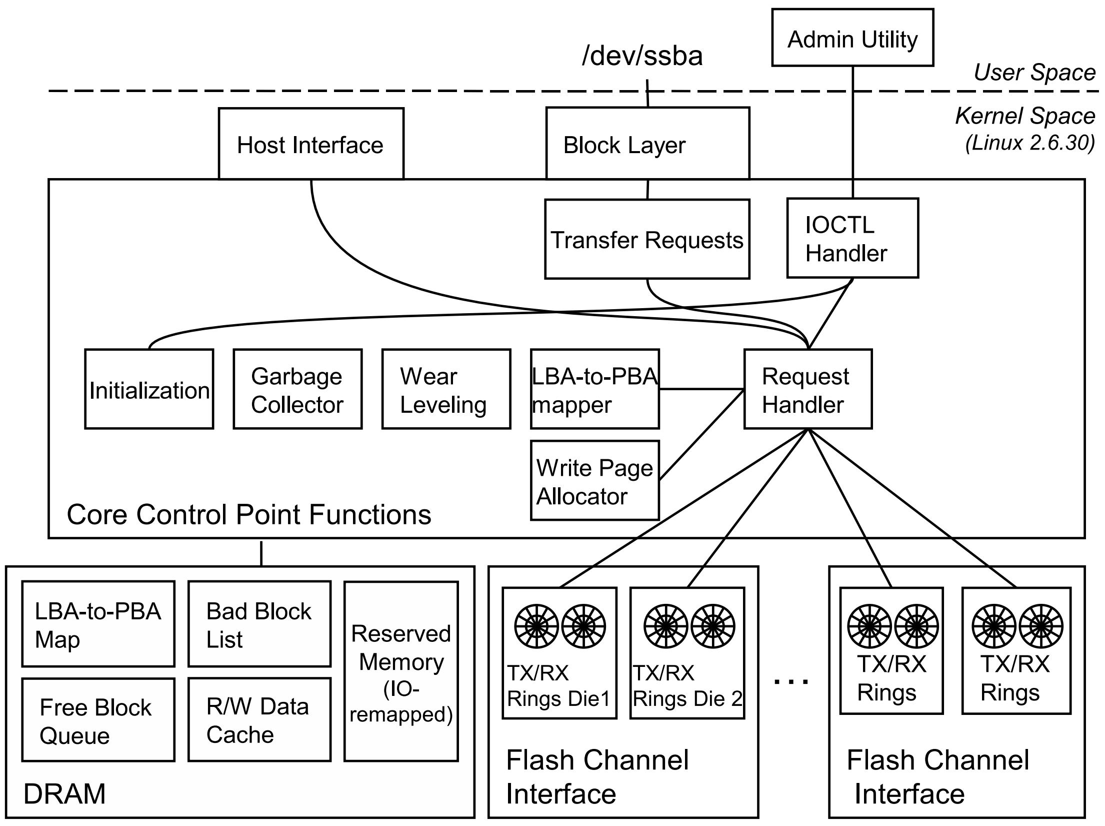


Fig. 3. OFC firmware architecture.

used for the additional meta-data regions of Flash pages as well as special pages holding controller meta-data that needs to be transferred from/to Flash.

There are several ways how Flash memory can be accessed using the Flash controller firmware: The normal access method for applications is the host interface. The block device interface `/dev/ssba` allows the logical address space of the device to be accessed. As data has to be copied from/to the memory, this entails the penalty of executing a memory transfer which is acceptable for testing purposes. Similarly, special IO-control operations on the block interface provide direct access to physical pages or can trigger Flash management functions (e.g., garbage collection and reinitialization of the device). The request handler can also be accessed directly from another kernel module. This is useful for benchmarking the OFC without host interface.

A. Request Handler

The request handler is the common denominator for all access methods to Flash. In case of a read request, it uses the logical address to determine the physical location of the data on Flash and sends a request to the corresponding Flash channel controller. In case of a write operation, a new free page is determined by the write page allocator, then the Flash page meta-data is assembled before a request is sent to the Flash channel controller. The communication to the Flash channel controller is done by means of ring buffers. There are two ring buffers, transmit (TX) and receive (RX), for each die in the system. While the Flash channel controller is polling for new requests on the TX rings, the controller firmware is informed by an interrupt about completed requests in the RX rings. The request information maintained by the controller firmware includes the entry point in the receive path which is used when completed requests from the RX rings are processed. This function is scheduled by the interrupt handler and depends on the actual type of the request, such as normal, relocation, or meta-data read/write operations or erase operation.

B. Logical-to-Physical Address Mapping

The host interface and the block device use a block size that corresponds to the Flash page size, but does not include page meta-data and ECC information. This enables all write operations to be processed at maximum speed. Therefore, we decided to support full page-level address mapping instead of a block-level or hybrid mapping. Write page allocation and the mapping table are hence simplified, but additional techniques are needed when the mapping table will no longer fit into the available DRAM. Currently, this is not needed in OFC because the entire mapping table fits into DRAM, but the code has been prepared to employ a scheme such as DFTL [18], in which the full mapping table resides in Flash and a partial table is maintained in a DRAM cache.

C. Write Page Allocator

The write page allocator in OFC transforms any type of write stream into a sequential write pattern to Flash. This can, for example, be achieved using a round-robin allocation scheme in which in each round from each channel a page of a given die is allocated, and in the next round from each channel a page from a next die. Such a scheme can therefore fully utilize the maximum write bandwidth provided by the hardware. The sustained write performance is primarily determined by the number of valid pages that need to be relocated when freeing space.

Towards the end of the lifetime of the Flash, however, regions that expose too many errors have to be retired. Either full blocks or individual pages can be retired. Retiring full bocks simplifies the process and greatly reduces the size of meta-data structures to be kept in DRAM and/or Flash. Maintaining an ordinary bad-block table fulfills the task, and hence no additional logic is needed in the write page allocator. However, when the first pages reach the maximum number of acceptable errors in a block, many other pages in that block will still be significantly below that error rate. This is illustrated in Figure 4 for an SLC NAND Flash memory device. Those pages could still be used for some time making the retirement of individual pages appealing.

Several options can be envisaged. For example, pages in a block can be retired individually and independently from each other using a bitmap for each partially used block. In the worst case, all blocks are partially used resulting in 64 MiB of bitmaps for a 256 GiB device using 4 KiB pages. To reduce memory consumption, one can limit the number of partially used blocks.

Another approach is based on the observation that for a given NAND Flash chip technology, a group of pages per block can potentially be used longer than others. For instance, it has been observed that there is a significant difference between even and odd pages in some devices and that other devices exhibit differences between the first and the second half of the pages in a block [2].

Finally, a third approach is to allocate pages based on the current raw bit error rate as well as the actual write pattern to be written as, for example, writing zeros degrades cells more than writing ones.

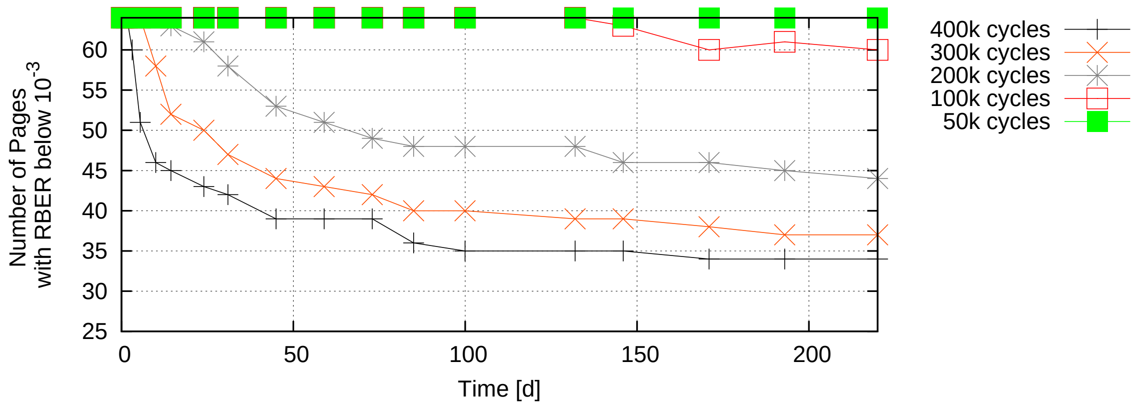


Fig. 4. Number of pages with a RBER below 10^{-3} when being read after having been aged by a given number of P/E cycles and kept for a given amount of time in a given block of device A. The device has 64 pages per block. Even at 400k P/E cycles (4x the manufacturers specification), half of the pages in the block can still be used.

The retirement of pages and blocks has an impact on the write page allocator. As the number of usable pages or blocks can vary from die to die, the initial goal to distribute the write load equally among all dies, preferably in a round-robin manner, has to be relaxed, which will ultimately result in a write performance degradation towards the end of the lifetime of the device.

D. Garbage Collection

Depending on how garbage collection is done, a higher or lower number of valid pages need to be relocated, which affects write amplification. Currently, we have implemented two different garbage collection schemes. The cyclic buffer scheme always selects the oldest block written as the first one for garbage collection. This scheme has the advantage of being simple, but can cause the relocation of many pages. An improvement to the cyclic buffer scheme is the greedy window algorithm. Instead of always selecting the oldest block for garbage collection, it selects the one with the most invalid pages from a window of oldest blocks. Other schemes that can distinguish between hot and cold data. Independently of the scheme, garbage collection can be accelerated by operating in parallel across channels or dies.

Another aspect of garbage collection is the placement of page meta-data which typically does not fit into DRAM and has to be maintained on Flash. In OFC, we have the option of placing page meta-data into the last page of a block similar as in [4], [5]. Although this wastes Flash capacity, the information necessary for garbage collection of a block can be obtained by a single read operation. Moreover, it has the advantage that the entire spare region in a Flash page can be used for ECC information.

E. Wear Leveling

Wear leveling tries to balance the write workload in such a way that all blocks in the system experience similar wear in terms of P/E cycle counts. There are two kinds of wear leveling. Dynamic wear leveling writes new data to that block from the free block pool that has the least wear. Static wear leveling erases the block with the least wear to allocate new pages to be written. However, pages that are still valid have to be relocated. Therefore, write page allocation and garbage collection are closely related to wear leveling and can help simplify the wear leveling process. Clearly, the cyclic buffer scheme needs no additional wear leveling.

V. Performance Measurements

In this section we present the performance results of our OFC. The goal is to validate the expected performance when the hardware and software are integrated and how it will be affected by different sequential and random workloads as well as the history of the write workload the device experienced. Moreover, we quantify the performance improvements provided by over-provisioning and the reduction of page relocations under a random write workload using the simple cyclic buffer garbage collector. Based on these results, one can easily derive the expected performance of other Flash management algorithms provided their impact on the number of relocated pages per block is known.

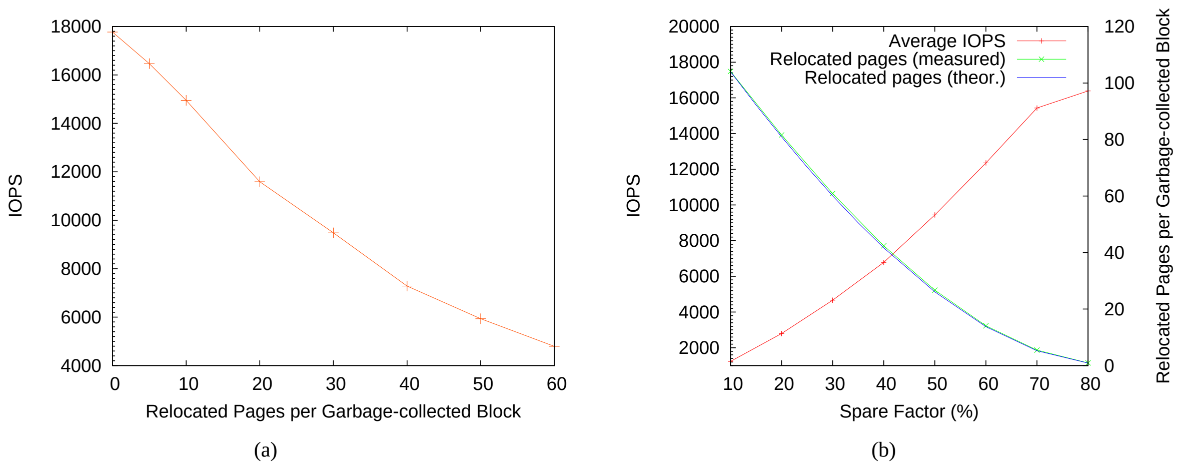


Fig. 5. Random write performance as a function of the number of relocated pages and spare factor.

A. Description of the Setup

To assess the performance under sustained workloads that match the bandwidth of the Flash controller, we added a kernel module that acts as traffic generator and measurement tool and plays the role of the host. Note that this workload generator consumes some, but only minimal resources. Therefore the fact that in this setup the workload generator and the system to be measured run on the same processor has a negligible impact on the measurement. To keep this impact minimal, we made several key design decision: First, the number of requests outstanding at any point in time is limited. Second, we use a simple pseudo random number generator called the Super Fast Hash [19], which turned out to be reasonably good for our purpose. Third, the data pages to be written reside in the reserved memory area and can be initialized to different patterns, such as random data, all zeros, or all ones. During a test the data pages are not touched. The test kernel module directly accesses the request handler in the Flash controller firmware to send requests to the Flash, i.e., in the same way as the implementation of the host interface is using it.

TABLE II
SYSTEM PARAMETERS

Parameter	Value
Spare factor	45%
Pages per block	128 (incl. one meta-data page)
Blocks per die	1024
Dies per channel	2
Channels	4
IOs per experiment	5,500,000

Unless otherwise stated, the following performance tests all use the parameters defined in Table II. The spare factor denotes the fraction of the total Flash capacity reserved for Flash management and therefore not available to the host. We set it to 45%, which is in the range of typical values found in commercially available products. We limited the number of blocks used for our experiments to 8192 Flash blocks, resulting in a total capacity of 4 GiB. This is a reasonable size to reduce the execution time of experiments and still allows an adequate evaluation of the Flash management functions. The number of IOs for a test allows the entire physical address space being used to be overwritten five times. Writing the whole device multiple times gives us ample opportunity to study the characteristics and performance of garbage collection. In addition, we collected statistics in intervals of roughly one second and used the Linux jiffies value to calculate the exact rates during each interval.

B. Random Write Performance as a Function of Page Relocations

To accommodate new write requests from the host, already written blocks need to be recycled when free blocks become scarce. Any relocation of valid pages that needs to be done during the garbage collection of a block causes additional overhead for the system and leads to a performance drop. If the system determines that there are no valid pages in that block, no relocation needs to be performed and the block is simply erased. On the other hand, if there are valid pages left in the block subject to garbage collection, the system needs to read those valid data pages and write them elsewhere before the block can be erased, hence causing write amplification.

To analyze the impact the number of invalid pages per block has on the performance, we modified the garbage-collection scheme to select an arbitrary fixed number of pages to be relocated from the oldest block independently of the actual number of still valid pages in that block. The number of relocated pages is one of the key parameters affecting the performance. Hence, it can be used as a figure of merit to assess other implementations of Flash management functions, in particular in terms of garbage collection and wear leveling.

Figure 5(a) illustrates the random write performance seen by the host as a function of the number of relocated pages. The performance drops by 50% when the number of relocated pages per garbage collected block increases from zero to 30. The drop in performance is due to the fact that the number of total writes in the system for each user write increases with the number of pages that need to be relocated. As a result, the increased amount of work caused by the relocation operations degrades the performance as shown in the graph. In summary, the number of relocated pages directly and significantly affects the overall system performance.

C. Random Write Performance as a Function of Spare Factor

One way to reduce the number of pages that need to be relocated is to exploit additional physical Flash space that is not directly visible to the host by increasing the spare factor. As the spare factor is increased, the average number of valid pages per physical block decreases, the blocks now stay longer in the system until they get garbage collected. This leads to fewer valid pages to be relocated per block whenever garbage collection is initiated and hence reduces write amplification. In Figure 5(b), we show the random write performance as a function of the spare factor. The random writes are uniformly distributed over the entire LBA space, and the cyclic buffer garbage collector is being utilized.

The average values for the IOPS and the number of relocated pages per garbage-collected block are calculated from the values observed after the device is in a steady state, i.e., the physical blocks of the device have all been overwritten several times and the values for the parameters being studied appear stable. By increasing the spare factor from 30% to 60% (meaning that the space on the device available to the user decreases from 70% to 40%, i.e., a drop of 40%), the performance more than doubles (from 6 kIOPS to 13 kIOPS). At the same time, the number of relocated pages per garbage-collected block drops from 45 to 15. So there is a clear trade-off between performance and the user capacity determined by the spare factor.

To verify this result, we can use the following probabilistic model presented in [20]: As the writes are uniformly distributed over the entire LBA space S_L, the probability that a single write has an LBA address $L = x$ is $p(L = x) = 1/S_L$. Any subsequent host write is independent of the preceding write. Hence, the probability that a page written with $L = x$ is still valid after a subsequent write is $1 - p(L = x)$. In a cyclic buffer, the first page in a block will be garbage collected once the entire PBA space S_P has been written. When a block with S_B pages is garbage collected, the expectancy E of still valid pages is

$$E = \sum_{i=0}^{S_B-1} \left(1 - \frac{1}{S_L}\right)^{S_P - \frac{ES_P}{S_B} - iD - S_B R} = \sum_{i=0}^{S_B-1} \left(1 - \frac{1}{S_L}\right)^{\left(\frac{S_L}{1-A}\right)\left(1-\frac{E}{S_B}\right) - iD - S_B R}, \qquad (1)$$

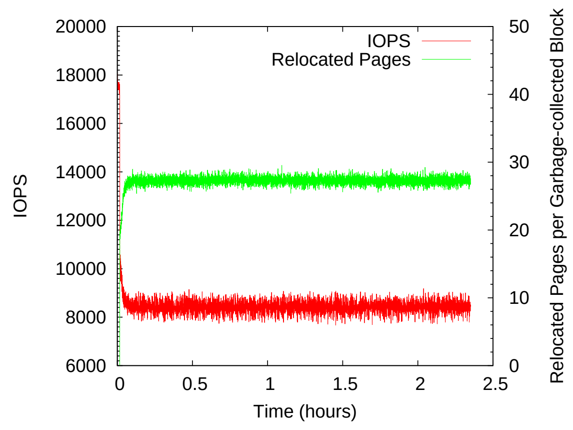
where S_L and S_P are expressed as number of pages, D denotes the total number of dies, adn R the number of reserved blocks. The factor D is used here because of the round-robin scheme of the write page allocator. The term ES_P/S_B denotes the number of pages being relocated, which have to be subtracted because they are valid pages. The spare factor A corresponds to $(S_P - S_L)/S_P$. Knowing that $S_P >> S_B$ Equation (1) can be further simplified:

$$E \approx S_B \left(1 - \frac{1}{S_L}\right)^{\left(\frac{S_L}{1-A}\right)\left(1-\frac{E}{S_B}\right)}. \qquad (2)$$

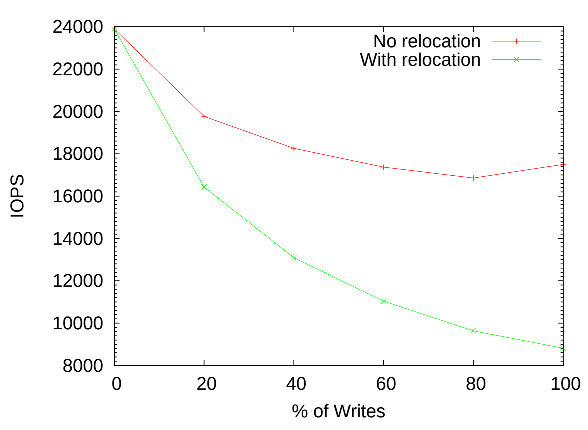
This equation can be solved using the fixed point method. Our measurements closely match the theoretical value shown in Figure 5(b) and are consistent with the write amplification results in [8] and [6]. For a spare factor of 45% (which corresponds to 82% over-provisioning), the write amplification factor is 0.5.

D. Long-term Random Write performance

To verify that our platform works reliably and predictably over longer periods of time, we run a longer term experiment consisting of random writes. Figure 6(a) illustrates the long-term random write performance. The physical blocks of the device have all been written once after a few minutes, and we can see that after that the random write performance is stable at 8.5 kIOPS. The number of relocated pages per garbage collected block also stays at 27 pages, which is consistent with the results in Section V-C.



(a) Long-term random write performance.



(b) Mixed random read/write performance.

Fig. 6. Long-term and mixed read/write performance

E. Mixed Random Read/Write performance

To investigate the performance under mixed workloads, we run experiments with mixed random reads and writes. Figure 6(b) illustrates the performance with mixed random reads and writes on a device that initially was completely written using sequential writes. For each operation in one run of the experiment, we first determine whether a read or write operation should be performed based on the ratio of reads and writes defined as an experimental parameter. Thereafter, the page to be read or written is chosen randomly. The upper curve shows the performance when the garbage collector only has to erase blocks, i.e., there are no valid pages in the blocks selected for garbage collection. The lower curve includes all relocation operations in the steady state. As we perform more and more random writes, the number of relocated pages per garbage-collected block increases until we reach another steady state, i.e., on average, each physical block that is garbage collected contains a similar number of valid pages. As expected, the performance drops as the percentage of write operations increases, especially if relocation operations of valid pages are done during garbage collection.

F. Analysis of Memory Effects due to Workload History

To analyze the memory effects and performance due to the workload history, we ran a sequence of tests in the following order: 1) sequential write on a clean device; 2) sequential read; 3) random read; 4) random write; 5) sequential read; 6) random read; 7) sequential write; 8) sequential read, and 9) random read. Figure 7 illustrates the variations in the performance as a result of different workloads and different workload histories. As we can see from the graph, the performance of similar workloads differ based on the state of the device.

The first batch of sequential writes (#1) is performed on a clean device. This is why the first curve starts at 18 kIOPS, before dropping to 17 kIOPS for the rest of the run because of the garbage-collection process. The sequential reads (#2) that are performed next, read the data in the same sequence as they were written. Therefore, all the channels and dies are utilized in an optimal fashion, and the read performance is 27 kIOPS. This is significantly higher compared to the subsequent random reads (#3) which experience a drop of 11 % and only attain 24 kIOPS. The reason for the lower performance of the random reads in this case is due to the fact that not all channels can be fully utilized as the read operations are no longer distributed evenly among channels and dies.

The second set of writes and reads begins with a random write sequence (#4). The performance is initially high, since the garbage collector has erased a few blocks since the end of the initial sequential write, and these are immediately available in the free block queues of each die. After these blocks have been filled, however, we see a significant drop in performance to as low as 5 kIOPS. The drop stems from the fact that the blocks selected for garbage collection by the cyclic buffer scheme have a high number of valid pages left over from the initial sequential write, and these need to be relocated. As more and more random writes are performed, the number of valid pages per garbage collected block decreases and stabilizes, and we get a steady-state write performance of 8.5 kIOPS. The subsequent sequential reads (#5) achieve a lower performance than when reading the sequentially written data, and this is also due to the fact that the channels and dies are not equally utilized. The performance of the random reads (#6) start off at the same level as for the sequential reads, but towards the end of the experiment the performance starts increasing. The performance improvement in this case is an artifact of the pseudo random

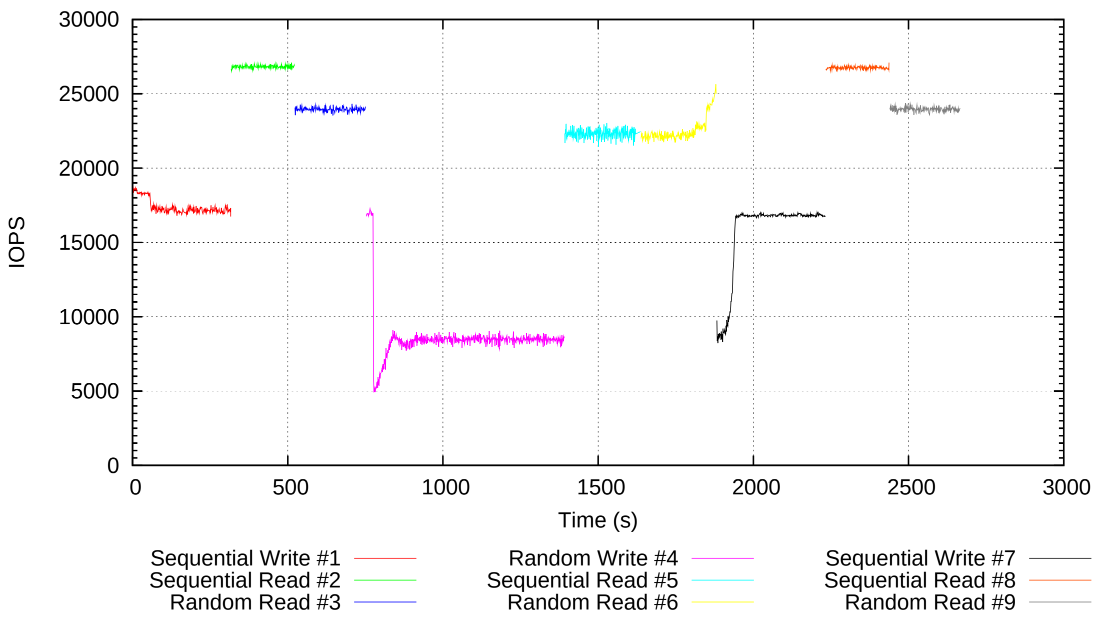


Fig. 7. Analysis of Memory effects from Workload History.

number generator used to calculate the logical block address for reads and writes: At the end of the random read run, the reads target the same pages and in the same order as they were (pseudo-)randomly written. As was the case for sequential reads when reading the sequentially written data, the channels and dies are traversed in the same, optimal round-robin fashion, resulting in abnormally high performance for the last part of random reads.

The sequential write sequence (#7) in the last set of write and read sequences initially has similar performance as the one of the random write sequence. This is due to the number of pages being relocated, which is initially the same as in the random write steady state but then decreases to zero because the sequential pattern is always overwriting the previously written pages. Once the sequential write achieves steady state, the performance improves and reaches the same value as in the first test (#1). The final sequential (#8) and random (#9) reads exhibit similar performance as in the first set of operations.

The results show that the performance of the device depends on the workload history, which has also been found by Stoica et al. [21]. The subsequent performance changes affect both reads and writes. However, often the performance of the device can often be "reset" by writing specific patterns, such as sequential or random patterns, over extended periods of time.

VI. Discussion and Conclusion

Regarding the random write performance of Flash, the number of pages that need to relocated is as important as the asymmetry between read and write operations. The latter is given by the chip characteristics and cannot be changed. Relocation of pages can be reduced by sophisticated garbage-collection schemes. The challenge lies in keeping the complexity and memory requirements low because resources in the controller are limited. The relocation operation itself can be improved by using so-called copy-back operations that keep data in the Flash chip register, i.e., do not transfer data through the Flash channels. This can only be done if page relocation is performed within the same plane in a Flash chip. The random write performance can also be improved by methods that reduce writes by means of compression or de-duplication.

We can confirm that the PPC in OFC is strong enough to execute Flash management algorithms. Preliminary tests using only the controller firmware revealed that up to 16 channels with pipelining can be handled.

One of the basic aspects of garbage collection - independent of the method used to select a block - is the number of blocks and page relocations that are handled in parallel. Too low a parallelism requires write requests from the host to be blocked every once in a while to keep up freeing space. On the other hand, too many parallel requests can add significant delays. We measured that single-block, per-channel garbage collection with 25% of the pages being relocated frees sufficient space to reach the maximum write bandwidth only in the absence of host requests. However, as soon as host requests are interleaved with garbage-collection requests, the latter will be significantly delayed. Hence, besides having simple per-die ring buffers, additional priority rings for garbage collection would improve the garbage-collection rate. The concept could also be used to favor reads over writes, which would be beneficial in a cache application. This however, would exceed the scope of this paper.

Another aspect is latency, which we could only measure using a logic analyzer directly on the Flash chips. The results were in the range the manufacturer has specified. The latency due to controller firmware processing could not be measured because the timer resolution in the Linux kernel is not sufficient. We plan to measure latency directly from the host once the interface has been completed.

To summarize, the OFC is a flexible and scalable Flash controller architecture that allows a rapid evaluation of new Flash management algorithms. We presented performance measurements on real hardware, supporting four Flash channels and two pipelined dies per channel, that quantify the fundamental issues in Flash management. The results can be applied to other schemes as well if the key parameters, namely the expected number of relocated pages and the spare factor, are known.

All the performance-related behavior such as memory effects due to workload history can be fully explained based on the hardware and software architecture of the OFC. To our knowledge, this is the first time this relationship is studied.

References

[1] N. Mielke, T. Marquart, N. Wu, J. Kessenich, H. P. Belgal, E. Schares, F. Trivedi, E. Goodness, and L. R. Nevill, "Bit error rate in NAND Flash memories," in *Proc. 46th Annual International Reliability Physics Symposium IRPS*, pp. 9–19, Apr./May 2008.

[2] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 24–33, Dec. 2009.

[3] Open NAND Flash Interface (ONFI), http://onfi.org/, *Open NAND Flash Interface Specification*, Jan. 2009.

[4] A. Birrell, M. Isard, C. Thacker, and T. Wobber, "A design for high-performance flash disks," *ACM Operating Systems Review*, vol. 41, pp. 88–93, Apr. 2007.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proceedings of the 2008 USENIX Technical Conference*, pp. 57–70, June 2008.

[6] G. Goodson and R. Iyer, "Design tradeoffs in a flash translation layer," in *Proceedings of Workshop on the Use of Emerging Storage and Memory Technologies HPCA WEST 2010*, Jan. 2010.

[7] C. Dirik and B. Jacob, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," *ACM SIGARCH Computer Architecture News*, vol. 37, pp. 279–289, June 2009.

[8] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of the Israeli Experimental Systems Conference (SYSTOR 2009)*, May 2009.

[9] A. Ben-Aroya and S. Toledo, "Competitive analysis of flash-memory algorithms," in *Proceedings of 14th Annual European Symposium on Algorithms (ESA)*, pp. 100–111, Sept. 2006.

[10] P. Desnoyers, "Empirical evaluation of NAND flash memory performance," *ACM SIGOPS Operating Systems Review*, vol. 44, pp. 50–54, Jan. 2010.

[11] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST)*, pp. 115–128, Feb. 2010.

[12] C. Park, P. Talawar, D. Won, M. Jung, J. Im, S. Kim, and Y. Choi, "A high performance controller for NAND flash-based solid state disk (NSSD)," in *Proceedings of IEEE Non-Volatile Semiconductor Memory Workshop NVSMW 2006*, pp. 17–20, Feb. 2006.

[13] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A case for flash memory SSD in enterprise database applications," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1075–1086, June 2008.

[14] J. D. Davis and L. Zhang, "FRP: A nonvolatile memory research platform targetting NAND flash," in *First Workshop on Integrating Solid-state Memory into the Storage Hierarchy WISH '09*, Mar. 2009.

[15] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li, "DFS: A file system for virtualized flash storage," in *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST)*, pp. 85–100, Feb. 2010.

[16] "Texas Memory System RamSan-20." http://www.ramsan.com.

[17] "Xilinx ML507 evaluation platform." http://www.xlinix.com.

[18] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in *Proceedings of ASPLOS '09*, Mar. 2009.

[19] P. Hsieh, "Super fast hash, http://www.azillionmonkeys.com/qed/hash.html," 2004.

[20] J. Menon and L. Stockmeyer, "An age-threshold algorithm for garbage collection in log-structured arrays and file systems," in *High Performance Computing Systems and Applications* (J. Schaeffler, ed.), pp. 119–132, Kluwer Academic Publishers, 1998.

[21] R. Stoica, M. Athanassoulis, R. Johnson, and A. Ailamaki, "Evaluating and repairing write performance on Flash devices," in *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, 2009.