What Systems Researchers Need to Know about NAND Flash

Peter Desnoyers
Northeastern University

Abstract

Flash memory has been an active topic of research in recent years, but hard information about the parameters and behavior of both flash chips and SSDs has been difficult to obtain for those outside of the industry. In this paper several misconceptions found in the literature are addressed, in order to enable future researchers to avoid some of the errors found in prior work.

We examine the following topics: flash device parameters such as page and erase block size, speed, and reliability, as well as flash translation layer (FTL) requirements and behavior under random and sequential I/O. We have endeavored to find public sources for our claims, and provide experimental evidence in several cases. In doing so, we provide previously unpublished results showing the viability of random writes on commodity SSDs when restricted to a sufficiently small portion of the logical address space.

1 Introduction

Non-volatile memory (NVM) has become an active area of research in the systems community, with multiple sessions devoted to it at some conferences. In Figure 1 we see the number of matches in the ACM Digital Library for the search terms "flash memory" by year; although the early matches are almost all accidental, the trajectory of the graph shows the emergence of a new research field within less than a decade.

Some NVM work being done today focuses on emerging technologies such as Phase Change Memory and STT-MRAM, and as such must make speculative assumptions about device characteristics when (or if) they come to market. However much research addresses NAND Flash, a fairly mature technology which is on the market in the form of devices with definite—if sometimes unknown—characteristics. If we assume the goal of much of this research is to develop ideas which may eventually be implemented on real systems, then it is important that assumptions made concerning the underlying technology are not in significant error.

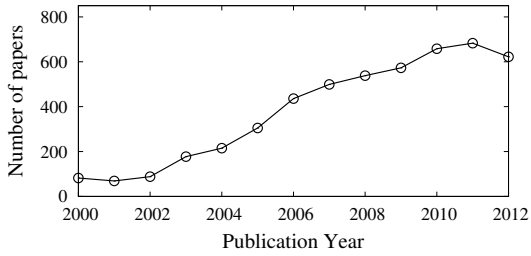

Figure 1: Matches for "flash memory" in ACM Digital Library by year of publication

However flash technology has been progressing at breakneck speed, sometimes documented in venues unfamiliar to the computer system research community, and other times hidden in industry behind non-disclosure agreements and trade secret protection. It is often difficult to obtain the information needed to ensure that research is relevant or even correct, sometimes requiring either experimentation or personal communication to obtain information not publicly disclosed. In recent years the specifications of the devices themselves (i.e. data sheets) have been treated as confidential information by vendors, unobtainable without a non-disclosure agreement and thus difficult to use in work destined for the open literature.

In this environment it is all too easy to base research on out-dated assumptions, and difficult for reviewers to determine when this is the case. In this paper we address a number of topics which have (based on the authors best determination) been mistaken, ignored, or confused in parts of the literature:

- Reliability and wear-out
- Device parameters: erase block and page size
- Read, write, and erase speed, and the curious orga-

nization of MLC pages
- FTL memory limitations
- Random write behavior of various FTLs
- The unique challenges of mobile storage

We examine each of these areas, dividing the topics into those concerning the flash chip and its parameters, FTL operation, and application-visible behavior from above the FTL.

2 The flash device itself

The characteristics of the silicon devices themselves play a large role in shaping algorithms and determining performance. Issues in this domain include:

Page and block size assumptions
For years flash device page sizes were standardized at 512+16 bytes and then 2048+64 bytes [21], with block sizes of 32 and then 64 pages; early work in the field often made assumptions based on these sizes [17]. Recently, however, these numbers have begun to fluctuate widely between devices and generations; the extreme values the authors are aware of to date are 8 KB (+ spare) pages grouped in 256-page erase blocks [22]. Even power-of-two assumptions may be incorrect, as erase blocks in TLC (three-bit cell) devices will be multiples of three pages (e.g. 192); however we are unable to find publicly-available documents to cite. There are strong incentives for vendors to continue to increase both page and block size parameters: long bit lines (i.e. large blocks) amortize the area of program and sense circuitry over more cells, while large pages enable high programming throughput despite long program latency [20].

Erase speeds
Google reports 12,700 hits for the phrase "expensive erase operations". However erase times have remained in the range of 2 ms to 4 ms across many generations of devices, while program times and block sizes have increased greatly. If all pages in a block are programmed before block erasure, the per-page amortized cost of erasure on today's devices is minor; on some TLC devices erasure may even be faster that page programming.

Read and write speeds
Work by the author [6] and Grupp et al. [11] has shown publicly what was no doubt widely known in industry, that the read and program latencies specified in device data sheets are only a somewhat vague indication of actual performance, especially for MLC devices. Of the two works cited, the author's misses a primary source of write latency variation—the existence of "fast" and "slow" pages in MLC devices—which is analyzed in Grupp. Neither work addresses systematic read latency variations, which appear to be far more device-specific. Both works demonstrate that performance varies in predicable ways as a device wears out: write latency decreases significantly for SLC devices (much less so for MLC) while erase latency increases.

These effects can be exploited in real systems. Grupp et al. present an FTL taking advantage of fast and slow pages, and commercial devices have been observed which use fast pages exclusively, or only for the LBA range used for the MSDOS FAT table.

The origin of these effects is due to the nature of the cell write and sensing mechanisms. A value is stored as a change in the threshold voltage V_t of a cell, which is an *input* of the cell, not an output. Sensing is performed in a *read-out* cycle which applies a particular voltage to all cells in the page, and 1s and 0s are sensed for cells with V_t below or above that voltage [9]. Since the relationship between programming voltage and V_t is not direct, *incremental step pulse programming* (ISPP [5]) is used: the programming voltage is increased in steps, each followed by a verification read-out.

In MLC devices, even-numbered pages are formed from the most significant bit on each cell, and odd-numbered pages from the LSBs. The precision required for even-page programming is much less, allowing larger and thus fewer ISPP steps. Read latency for even and odd pages depends on how many levels must be sensed; in one encoding an even page requires two read-outs while an odd page requires only one [5]. The author is not aware of any vendor disclosure of the encoding scheme used by a particular device.

Endurance
Many works assume that flash devices have a fixed lifetime (e.g. of 10^4 and 10^5 cycles for MLC and SLC), or that wear-out detection may be delegated to the device, which returns an error in the case of an unsuccessful program or erasure operation [6]. In reality wear-out is not a binary phenomenom, where cells suddenly "die", but is rather in large the result of the relationship between program/erase cycling and data retention.

Flash cells do not retain data forever; even in new cells V_t will fall over time, resulting in a corresponding increase in raw bit error rate. Program/erase cycling increases the speed at which this loss occurs, thus decreasing the effective retention time. Vendor-specified endurance figures are for 1-year retention; thus an endurance value such as 10^4 means that after 10^4 program/erase cycles, data retained for 1 year will have an uncorrectable bit error rate no greater than ε after ECC capable of correcting k errors, where k is vendor-specified and ε is typically 10^{-13} to 10^{-16} [23].

In this formulation it becomes clear that endurance is a function of retention and RBER requirements: by accepting lower retention times or using stronger ECC one may increase effective endurance [19]. Conversely, longer re-

tention requirements will reduce the effective endurance: for commercial parts, 10-year retention is only guaranteed up to 1/10 the rated endurance.

Like page and block sizes, endurance ratings now vary widely. They may be as low as 2000 or 3000 with 8 bit per 512 byte ECC for MLC parts; conversely "E-MLC" (enterprise MLC) devices are available with rated endurance of between 10K and 30K cycles. (and a rated retention time of 3 months [29]) In addition these values are worst-case limits, and it is known that management of the interval between program/erase cycles can increase lifetime significantly [24].

Since effective wearout occurs when data retention falls below a certain point (e.g. 1 year), it is unlikely to be detected by a program verify step microseconds after programming. It appears that devices will report successful programming long after they cease storing data reliably, calling into question extreme lifetime measurements [6]. (Some low-end devices may in fact use program/erase failure to detect wearout [3], risking massive data loss.) This also calls into question absolute bit error rates reported by Grupp, which were measured shortly after programming; retention testing appears necessary for accurate measurements [23].

To avoid data loss it appears to be necessary to abandon a block long before it fails programming or erasure, to avoid bit error rates higher than can be handled by the implemented ECC. This may be done by strict counting; it also appears possible to do this by measuring errors corrected by ECC at read time. Finally, some proposed (and implemented? [30]) designs incorporate multiple levels of coding [19] and/or decoding [31] depending on retention and bit error rate; among other effects this may cause read or write performance to fall with age.

3 FTL Design Parameters

Here we examine the constraints on FTLs themselves.

RAM requirements

"Flash translation layers must minimize their use of costly SRAM". This phrase may be found in the introduction of many papers proposing new flash translation layers, but in what cases is it true?

For enterprise SSDs, memory usage seems to be at best a secondary consideration. In some cases (e.g. Fusion IO [10]) flash translation layers take advantage of large amounts of host DRAM. Almost all consumer SSDs combine a controller chip with an external DRAM; in this case the cheapest DRAMs today are 128 MB devices [8]; SSDs examined by the author have used DRAMs between 32 MB and 128 MB, with larger sizes more common.

However, SD cards and USB "thumb" drives are vastly different. Here a single-chip controller is equipped with very limited on-board SRAM—an older USB controller, for instance, has 5 KB of DRAM [26]. This would not be enough for a block mapping table for e.g. 8 GB of 512 K blocks (128 pages of 4 KB), but if a virtual block size of 4 MB is used the table (2 K 16-bit entries) will just fit. Published SRAM sizes for newer USB and SD controllers could not be found; however they are unlikely to be more than a few 10s of kilobytes.

SSD overprovisioning

A few works on wear leveling assume free space is solely for replacing blocks that wear out; in fact it is primarily for performance. Like file systems, flash translation layers perform poorly if they have little free space, causing large amounts of data copying [7]. Consumption of significant amounts of this free space by bad blocks would in fact result in substantial performance degradation.

Page spare areas

At least one proposed flash translation layer (Superblock [15]) requires significant space in the page spare area for data structures. As raw bit error rates grow with shrinking geometries, the space needed for ECC bits has grown greatly, leaving little space in page spare areas for FTL metadata. (the extended Superblock paper [14] includes a data area-only variant) In extreme cases (e.g. early Indilinx Barefoot controllers [25]) hardware designers have been known to effectively omit FTL access to the spare area, requiring all metadata to be stored in page data areas.

Scanning on startup

Some page-mapped FTLs in the open literature (e.g. DFTL [12]) recover from unclean shut-down by scanning all page spare areas on start-up. Unfortunately a page read takes a constant time, regardless of how much data is needed. Under very conservative assumptions—a 128 GB SSD with 8 channels, 4 KB pages, and a $50\mu S$ read latency— a scan of all pages would require an infeasible 195 seconds at power-on. A block scan, taking slightly over a second for 128-page blocks, would however be feasible [1]; more sophisticated mechanisms [28] can provide further improvements.

4 Application-level Behavior

Finally we address the application-level issues: which workloads will result in best- or worst-case behavior? And is smartphone storage really as bad as reported [16]?

Write size and alignment

On hybrid block-mapped FTLs (e.g. BAST, FAST, or derivatives) peak performance may be achieved—even on fragmented drives—with long sequential writes. It seems that it should be possible to achieve this performance for more random operations by restricting writes to properly-aligned erase-block sizes, but it's a bit tricky.

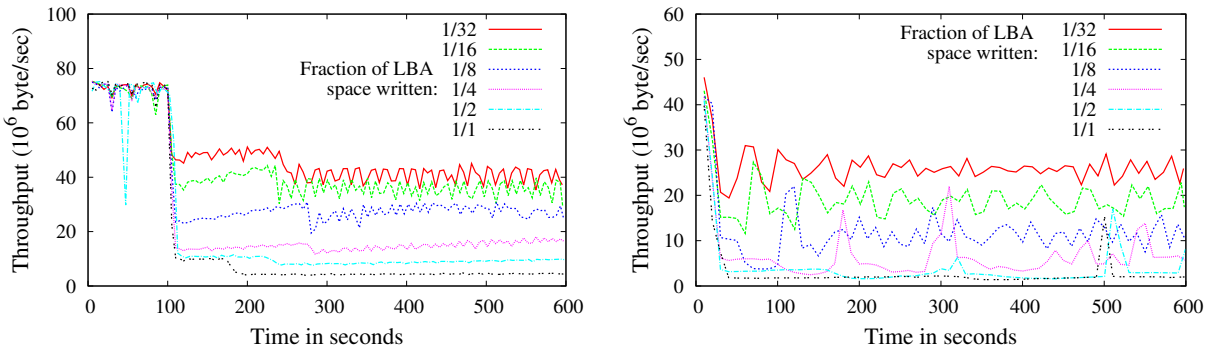

Figure 2: Random write performance for page-mapped (left) and FAST-like SSDs. Random writes were restricted to different fractions of the LBA space, ranging from the entire drive $(1/1)$ to $1/32^{th}$ of the drive.

This is almost impossible on a page-mapped FTL, as there are no constraints on the relationship between logical and physical addresses, preventing the alignment of external writes to physical boundaries. It is possible with block-mapped FTLs, but the problem is in finding the actual erase block size.

The on-chip erase block size is often far different from the *effective* block size - i.e. the amount of consecutively-written data which will be erased as a unit. On small devices this happens when multiple physical blocks are combined into a single virtual block, to minimize memory requirements of per-block tables. On consumer and high-end devices, however, it occurs because writes are striped across multiple channels. Writing one block of data on an 8-channel device[1] will actually fill one-eighth block on each of eight chips; one must fill all eight in order to achieve fragmentation-free operation. One cannot always assume a power-of-two number of channels (e.g. Intel, with 10), or that all pages in a block hold data [1].

Random write speed

Although SSDs are famous for slow random writes, on page-mapped devices or those using FAST [18] derivatives, random write performance may be high if writes are concentrated on a region smaller than the drive free space. With large write working sets the drive may still handle a burst of random writes before slowing. For Hybrid Log Block [17] (BAST), if the area being written fits within the FTL's log blocks, performance will be degraded by a factor of two, but no further—a much more stringent restriction on the write working set.

In Figure 2 we see throughput for two consumer SSDs, one page-mapped and one using a FAST-like translation layer. For each run the drive was prepared by trimming and then sequentially overwriting the entire LBA space, then random 32 K writes were performed. (using Linux 2.6.35, direct I/O, and an LSI SAS1068E adapter)

[1]Ignoring ways per channel...

Each line corresponds to a single 10-minute run, with random writes being targeted to a certain fraction of the LBA space, from 100% down to $\frac{1}{32}$. As the fraction of the LBA space written decreases, the time until performance degradation is fairly constant, but the level to which it falls varies widely.

Commodity SSDs may thus be able to achieve sustained high throughput for random writes, if restricted to a sufficiently small fraction of the LBA space. This argues for a re-examination of free space management and allocation in on-SSD file systems. In particular, an allocation strategy which maximizes re-use of logical blocks (e.g. by tracking free space in LIFO order) may offer increased performance.

At least one prior SSD measurement work (Chen et al. [4]) restricts most random write measurements to a small range of the device, no doubt giving results which do not reflect more highly random workloads.

Mobile device storage

Recent work by Kim et al. [16] has shown that MicroSD cards used as removable storage in smart phones are extremely slow, significantly degrading application performance. Yet their experiments focus almost exclusively on removable devices, leaving some hope that reasonable performance might be achieved by using internal flash.

Unfortunately this does not appear to be the case. Every Android device we have checked (e.g. [27, 13] uses eMMC [2] flash, which is essentially a MicroSD card in a soldered-down package. These eMMC devices have the same problems of extremely poor random write performance and high per-operation latency (for reads as well as writes) as have been documented for SD cards, no doubt due to similar controller limitations.

Experiments were run on a Galaxy Nexus, using a modified Android kernel for direct I/O, and performing sequential reads and writes of varying sizes; results are seen in Figure 3. (random is even worse) Performance

Figure 3: Sequential I/O on Google Nexus 7 (eMMC storage) for varying read and write request sizes.

for small request sizes is extremely low; in the read case this appears to be due to high per-operation latency (over 0.5 ms), while in the write case we hypothesize the use of read-modify-write operations on large block sizes.

5 Conclusions

Flash research has been an area to date with many pitfalls. We illuminate several of them in this paper, present experimental results showing conditions under which random I/O may be performed at high speed on SSDs, and extending results of Kim et al. to demonstrate similar performance of smartphone on-board flash to that of removable flash devices tested in their work.

This work was supported in part by NSF award CNS-1149232.

References

[1] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M., AND PANIGRAHY, R. Design tradeoffs for SSD performance. In *USENIX ATC* (2008), pp. 57–70.

[2] ASSOCIATION, J. S. S. T. Embedded multimediacard(e-mmc) emmc/card product standard, high capacity. Tech. Rep. JESD84-A441, Mar. 2010.

[3] BOBOILA, S., AND DESNOYERS, P. Write endurance in flash drives: Measurements and analysis. In *FAST* (San Jose, California, Feb. 2010), USENIX Association.

[4] CHEN, F., KOUFATY, D. A., AND ZHANG, X. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *SIGMETRICS* (Seattle, WA, USA, 2009), ACM, pp. 181–192.

[5] CRIPPA, L., MICHELONI, R., AND MARELLI, A. MLC storage. In *Inside NAND Flash Memories*. Springer, Aug. 2010.

[6] DESNOYERS, P. Empirical evaluation of NAND flash memory performance. In *HotStorage'09* (Oct. 2009), vol. 44, ACM, pp. 50–54.

[7] DESNOYERS, P. Analytic modeling of SSD write performance. In *SYSTOR* (2012), pp. 1–10.

[8] DRAMEXCHANGE. Dram spot price. `dramexchange.com`, Mar. 2013.

[9] FRIEDERICH, C., MICHELONI, R., CRIPPA, L., AND MARELLI, A. Program and erase of NAND memory arrays. In *Inside NAND Flash Memories*. Springer, Aug. 2010, pp. 55—88.

[10] FUSION-IO INC. Fusion-IO drive datasheet. `www.fusionio.com/data-sheets/iodrive-data-sheet/`.

[11] GRUPP, L. M., CAULFIELD, A. M., COBURN, J., SWANSON, S., YAAKOBI, E., SIEGEL, P. H., AND WOLF, J. K. Characterizing flash memory: anomalies, observations, and applications. In *Int'l Symp. on Microarchitecture* (2009), ACM, pp. 24–33.

[12] GUPTA, A., KIM, Y., AND URGAONKAR, B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS* (2009), pp. 229–240.

[13] IFIXIT. Nexus 4 teardown. `www.ifixit.com/Teardown/Nexus+4+Teardown/11781/1`, Nov. 2012.

[14] JUNG, D., KANG, J.-U., JO, H., KIM, J.-S., AND LEE, J. Superblock FTL: a superblock-based flash translation layer with a hybrid address translation scheme. *ACM Trans. Embed. Comput. Syst. 9*, 4 (2010), 1–41.

[15] KANG, J.-U., JO, H., KIM, J.-S., AND LEE, J. A superblock-based flash translation layer for NAND flash memory. In *IEEE EMSOFT* (2006), pp. 161–170.

[16] KIM, H., AGRAWAL, N., AND UNGUREANU, C. Revisiting storage for smartphones. In *FAST* (2012), p. 17.

[17] KIM, J., KIM, J. M., NOH, S., MIN, S. L., AND CHO, Y. A space-efficient flash translation layer for CompactFlash systems. *IEEE Trans. Consumer Electronics 48*, 2 (2002), 366–375.

[18] LEE, S.-W., PARK, D.-J., CHUNG, T.-S., LEE, D.-H., PARK, S., AND SONG, H.-J. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst. 6*, 3 (2007), 18.

[19] LIU, R.-S., YANG, C.-L., AND WU, W. Optimizing NAND flash-based SSDs via retention relaxation. In *FAST* (2012), p. 1111.

[20] MAFFITT, T. M., DEBROSSE, J. K., GABRIC, J. A., GOW, E. T., LAMOREY, M. C., PARENTEAU, J. S., WILLMOTT, D. R., WOOD, M. A., AND GALLAGHER, W. J. Design considerations for MRAM. *IBM Journal of Research and Development 50*, 1 (2006), 25–39.

[21] TN-29-07: small-block vs. large-block NAND flash devices. Tech. Rep. TN 29-07, Micron Technologies, 2005.

[22] MICRON TECHNOLOGY INC. 128gb, 256gb, 512gb async/sync enterprise nand. Tech. rep., 2010.

[23] MIELKE, N., MARQUART, T., WU, N., KESSENICH, J., BELGAL, H., SCHARES, E., TRIVEDI, F., GOODNESS, E., AND NEVILL, L. Bit error rate in NAND flash memories. In *IEEE Int'l Reliability Physics Symposium* (2008), pp. 9–19.

[24] MOHAN, V., SIDDIQUA, T., GURUMURTHI, S., AND STAN, M. R. How i learned to stop worrying and love flash endurance. In *HotStorage* (June 2010).

[25] Jasmine OpenSSD platform. `www.openssd-project.org/wiki/Jasmine_OpenSSD_Platform`.

[26] PHISON ELECTRONICS CORPORATION. PS2134 USB 2.0 Flash Controller Specification, May 2005.

[27] POWERBOOK MEDIC. Google nexus 10 take apart first look. `www.powerbookmedic.com/wordpress/2012/11/16/google-nexus-10-take-apart-first-look`, Nov. 2012.

[28] ROGERS, A. M., KANSAL, A., AND PATEL, S. C. US Patent: 7818610 - rapid crash recovery for flash storage, Oct. 2010.

[29] EMLC vs. MLC NAND flash. Application Note AN002, SmartStorage Systems, Feb. 2012.

[30] CellCare technology. Tech. rep., STEC Inc., 2011. at `www.storagesearch.com/stec-cellcare-paper.pdf`.

[31] ZHAO, W., DONG, G., SUN, H., ZHENG, N., AND ZHANG, T. Reducing latency overhead caused by using LDPC codes in NAND flash memory. *EURASIP J. Adv. in Sig. Proc. 2012*, 1 (Dec. 2012), 1–9.