The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs

Laura M. Grupp[†], John D. Davis[‡], Steven Swanson[†]
[†]Department of Computer Science and Engineering, University of California, San Diego
[‡]Microsoft Research, Mountain View

Abstract

Recent years have witnessed significant gains in the adoption of flash technology due to increases in bit density, enabling higher capacities and lower prices. Unfortunately, these improvements come at a significant cost to performance with trends pointing toward worst-case flash program latencies on par with disk writes.

We extend a conventional flash translation layer to schedule flash program operations to flash pages based on the operations' performance needs and the pages' performance characteristics. We then develop policies to improve performance in two scenarios: First, we improve peak performance for latency-critical operations of short bursts of intensive activity by 36%. Second, we realize steady-state bandwidth improvements of up to 95% by rate-matching garbage collection performance and external access performance.

1 Introduction

NAND flash memory can provide orders-of-magnitude faster performance than traditional rotating media (HDDs), albeit at the cost of reduced capacity. Pushing flash to higher densities, causes significant decline in other metrics – like performance, endurance, and reliability. Increasing flash's capacity by storing an additional bit per memory cell (1 to 2 bits, or 2 to 3 for example) reduces the chip's lifetime by 5-10%, shrinks throughput by 22% to 98% (55% on average) and increases latency by 1.3× to 4.0× (2.3× on average) [14]. Increasing density via scaling leads to smaller, but still significant declines.

Despite the disturbing trends resulting from increasing the density of the underlying flash technology, flash systems remain very promising. The chip-level trends are driving the development of increasingly sophisticated flash management techniques. For example, sophisticated error coding techniques based on a deep understanding of flash's behavior [12, 5] can bring triple-level cell (TLC) bit error rates and performance in line with multi-level cell (MLC 2-bit/cell) technology [1], and aggressively exploiting parallelism can partially compensate for increasing latencies.

This paper exploits another characteristic of high-density flash devices to improve SSD performance. The dominance of MLC over SLC devices leads to systematic variation in the program latency of different pages. We have developed a flash translation layer (FTL) that schedules programs to pages according to the program operation's purpose (e.g., internal garbage collection vs. storing user data) and the speed of the page (i.e., faster or slower). Our scheduling algorithm improves performance without sacrificing capacity or endurance, providing speed of the hare (high performance) *and* the endurance of the tortoise (increased capacity and reduced write amplification). In particular, we make the following contributions:

- A flexible FTL which is aware of different page types and can direct operations accordingly.
- A *Many Write Point* mechanism for increasing scheduler flexibility and thereby enhancing the effect of scheduling policies.
- A scheduling policy that provides SLC performance on an MLC device for performance-critical operations and bursty workloads.
- An analytical model of steady state SSD performance that guides our access scheduler and suggests some non-intuitive scheduling algorithms.

Our FTL architecture and multi-write point mechanism allow the system to more readily access the array's variability. With this improved access and our policies, our FTL improves burst bandwidth by up to 36% (equal to the performance of an SLC array) with no increase in wear, and improves performance of sustained traffic by up to 95%.

First, we provide some background information on NAND flash and SSDs. Section 3 follows with a description of our baseline architecture, simulation infrastructure and our methodology. Next, Section 4 describes our enhancements to the FTL which efficiently leverage page latency variation. We follow this with our evaluation in Section 5, suggestions for applying the mechanisms in Section 6, related work in Section 7, and conclusions in Section 8.

2 Background

NAND flash memory is the driving force behind the ongoing success of solid-state drives (SSDs). This section describes the basics of flash chip operation and the source, magnitude and patterns of page latency variation.

2.1 Flash memory

The packages composing the flash array in an SSD each contain one or more flash dies. Within a flash die, multiple (typically two) "planes" each contain several thousand 128 kB to 3 MB blocks that, in turn, contain 64 to 384 2-8 kB pages. The chips perform reads and writes on pages. However, before the chip can program (write) new data to a page, it must first erase the parent block. Further complicating writes, FTLs must write pages in order within each block. The FTL may skip over a page, but after doing so cannot write to it until after erasing it.

To represent the data, each memory element uses charge stored on a floating gate between the control gate and channel of a transistor. Varying amounts of charge on the floating gate determine the effective threshold voltage (V_{TH}) of the transistor, creating an analog range which the chip interprets as two regions for a single bit. Physically, a block comprises an array of "flash chains" that each contain 32-128 floating gate transistors in series with each other. To a first order, the n^{th} page in the block comprises the n^{th} bit in each of the block's chains (we discuss this more detail in Section 2.2).

Multi-level cell (MLC) flash stores multiple bits per floating gate (usually 2 bits) to improve density by interpreting the range of possible V_{TH} as 4 regions. This improved density (i.e., lower cost) makes MLC the dominant type of flash. Single-level cell (SLC) devices are less-dense, faster, and more expensive. TLC is in production systems and Macronix recently demonstrated 6-bit-per-cell technology [16]. We focus on the performance of the write operation in MLC devices in this study, and we discuss it in more detail in the next section.

Flash memories exhibit a well-known wear-out behavior which causes their data retention time to degrade with increasing program-erase (PE) cycle counts. Manufacturers rate current MLC devices for between 5,000 and 10,000 PE cycles, after which the data may become unrecoverable without very aggressive ECC protection. While wear-out remains a first-class concern, large over-provisioned flash arrays, common wear management techniques and recent advances in chip-level technology [11] help.

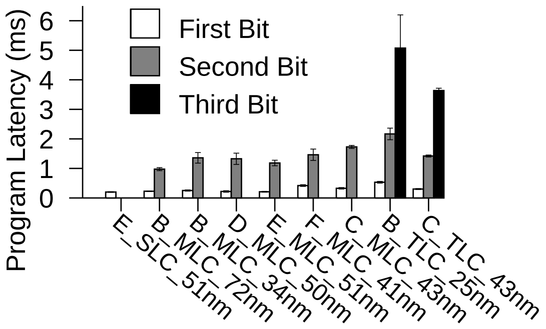

Figure 1: **Chip Program Latency** Multi-bit flash chips retain single-bit performance in their fast pages. The increase in latency is confined to the chips' added capacity.

2.2 Flash Chip Performance Variability

The techniques we propose exploit systematic page-level variation in write performance. This section describes the source of this variation, magnitude of variation we have measured in flash chips, the architectural lay-out of fast and slow pages within each flash chip, and how the FTL can non-destructively detect this pattern. Each of the 30 chip models (from 6 manufacturers) we have characterized show distinct groups of latencies in proportion with the number of bits stored in each memory element.

The variation arises because, although MLC devices store multiple bits on a single floating gate, those bits map into different pages. As a result, the programming operation for the first *fast* bit stored on the gate is much faster than the programming operation for the second *slow* bit, and so on for all additional bits stored in the cell. We refer to individual pages as fast or slow depending on which kind of bits they contain.

Figure 1 shows the latency of a representative sample of SLC, MLC and TLC chips. For each chip we measure the time to write random data to each page in 16 blocks. We divide these measurements into fast, slow and (for TLC) medium page latencies. Slow pages from the average MLC chip are 4.8× slower than fast pages, with D_MLC_50nm exhibiting the largest gap (6×) and F_MLC_41nm the smallest at 3.5×. Our data show that fast page program latency is comparable to SLC program latency in devices from similar technology generations [13].

Our previous work reveals two common organizations for fast and slow pages within an MLC block. We now extend those observations to TLC parts as well. With the exception of one manufacturer, the chips exhibit the organization in Figure 2A. In MLC devices, the first four pages are fast, the last four are slow and every pair of pages mid-block alternate between fast and slow. TLC devices cycle through the three latencies with pairs of

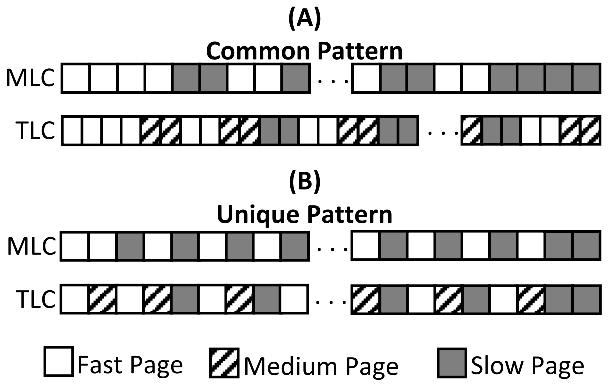

Figure 2: **Latency Pattern** Pages' read and write latencies follow the same pattern within each block of a given chip.

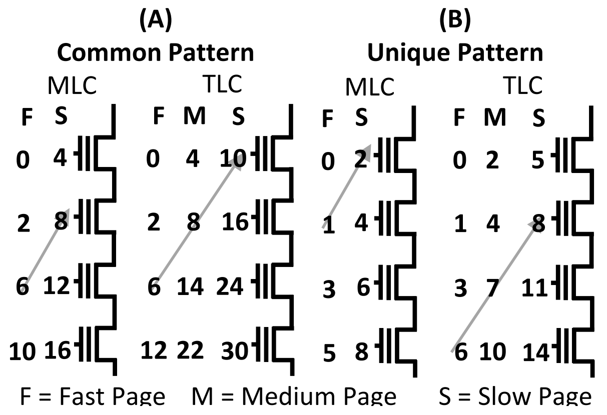

Figure 3: **Memory Cell Anatomy** Fast pages consist of each memory element's first-written bit. In-order programming causes the final bit of a memory cell to be written after most programs to the surrounding cells.

pages as well. The unique manufacturer follows the single-page alternating patterns in Figure 2B.

Figure 3 shows how a single bit from each page maps to the chain of flash memory cells. The numbers correspond to the page's location within the block and are in columns corresponding to the time required to program the bit. Figure 3A shows the even-numbered NAND chains from MLC and TLC parts made by most manufacturers (the corresponding odd chain is similar), and Figure 3B shows the pattern used by the manufacturer with a unique pattern.

Because of the in-order programming constraint, the final program of a cell occurs after most of the program operations to adjacent cells are complete. This reduces the program disturb that is a major hindrance to enabling multi-bit technology [21]. The blocks of most manufacturers alternate between page speeds in pairs because they separate pages into even and odd chains, while the unique manufacturer uses only one chain. Also, we observe most of the variation in the latency of slow pages (indicated by the wide error bars in Figure 1) comes from the even chain being slower than the odd chain, though we are unfamiliar with the cause.

The techniques we develop in the following sections depend on the FTL knowing the layout of fast and slow pages within a block. Since the layout is consistent for a given part number and does not vary over time, it is sufficient for the manufacturer to detect this pattern using a single block and configure the FTL accordingly. An FTL could perform the measurement at initialization time by monitoring the programming time of pages in a block, reducing the cost of moving to a new type of flash chip in an existing SSD design. There is also a non-destructive technique for determining page type. Page read latencies exhibit the same variation pattern. Furthermore, differentiating between the small number of possible patterns (either mentioned in the datasheets or derived empirically) requires only a few page reads.

Overall, as shown in Figure 1, the dramatic differences in page program latency provide a better opportunity to exploit diversity to improve SSD performance. In Section 4, we describe our extensions to the baseline FTL (from Section 3) which leverage these variations in program latency.

3 Baseline FTL

SSDs contain both an array of flash and a controller to manage wear leveling and access requirements while presenting a block interface. The following sections describe the basic algorithms needed in all FTLs, how we structure the algorithms to isolate important policy decisions, and our simulation infrastructure and array parameters.

3.1 FTL Basics

SSDs maintain a mapping between the logical block addresses (LBA) that the host system uses and the physical block addresses (PBA) that identify particular pages within the flash array. The FTL maintains this map with the goal of minimizing wear and maximizing performance. FTLs fall into three broad categories based on the granularity of this map – block-based, page-based and hybrids of the two. Improving the FTL is the object of intense work both in industry and academia (see Section 7).

In this work, we study variability-aware enhancements to a page-based FTL, but the concepts extend to other designs as well. We begin with the parallelized FTL architecture described in [7]. It uses log-structured write operations, filling up one block before moving on to another. To improve bandwidth, the FTL maintains one log for each chip in the array. We refer to the head of each log as a *write point*.

As the FTL writes new data at a write point, the old version of the data for that LBA becomes invalid but remains in the array. The effects of this *copy-on-write* procedure requires that we provide functionality to (1) recover the physical-to-logical address mapping after unexpected power failure and (2) convert pages containing stale data to erased flash through garbage collection (GC).

First, for the FTL to recover from unexpected power failure it must track each page's logical address (LBA) as well as which copy of data for a given LBA is most-recent. With a single-write point array, a block sequence number suffices. However, when the system contains more than one write point, the FTL must use a page sequence number to maintain strict ordering. (See [6] and [7] for more details.)

Second, the FTL must remove the stale copies and create room for new data by performing GC. GC algorithms copy valid data from partially-invalid blocks to write points on or off chip, and erase the now fully-invalid blocks to make them ready for new write operations.

GC must constantly maintain a pool of erased blocks on each chip. When a write point reaches the end of a block, the block is full and the FTL must locate a new, erased block for that write point to continue writing. When a chip starts to run short on erased blocks, GC begins to consolidate valid data to create additional erased blocks for that chip. In the best case, garbage collection makes use of idle periods to hide its impact on performance. However, GC latencies are a significant source of performance variability in SSDs.

Our FTL uses two thresholds as parameters for the GC routines. The FTL maintains these thresholds on a per-chip basis, so in the worst case, any single chip can free up resources by taking itself off-line for cleaning. The first threshold is the *background (BG)* threshold. When the FTL finds any chip in the array idle, it performs GC operations on that chip up to the BG threshold. If the number of erased blocks on any chip drops below the second, *emergency* threshold, GC becomes the FTL's top priority for that chip and it will divert all incoming traffic to other chips or block entirely while GC proceeds. In normal operation, the FTL should very rarely enter this "emergency mode."

3.2 Design for Flexible Policy Choices

Figure 4 shows the high-level structure of the FTL's operation scheduler. The FTL maintains three queues. The queues hold write, erase, read and *cleanup* operations waiting to execute. External accesses to the SSD enter the *external* queue, background GC operations reside in the *background* queue, and the *emergency* queue holds emergency GC operations. Emergency mode is a rare occurrence.

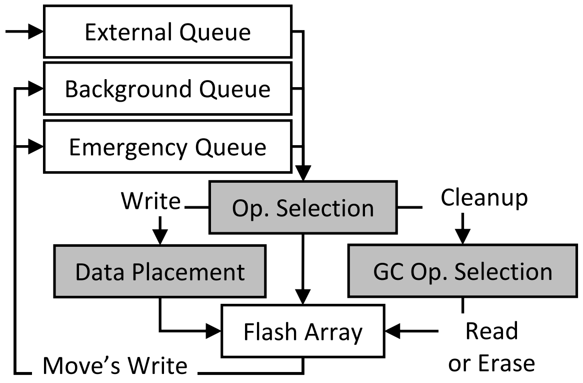

Figure 4: **Operation Flow** Operations move through the FTL's queues and a series of policy decisions (the gray boxes) before executing on a flash chip.

Operations pass from the queues to the flash array via three distinct policies, marked by the gray boxes in Figure 4:

Operation Selection Policy First, the FTL chooses which operation to execute next. Operations in the emergency queue have the highest priority. If the emergency queue is empty, or contains operations that cannot yet execute (for example, they must access a busy chip or wait for data being read), then an operation is taken from the external queue. Finally, operations are taken from the background queue when the system is idle.

Data Placement Policy The second policy in the FTL determines where to schedule writes. Because the physical address of an LBA changes with each write, the FTL has the freedom to choose, for example, the fastest page available. In our baseline design, the FTL follows a round robin approach which avoids busy chips and seeks to maintain a uniform number of valid LBAs on each chip.

GC Operation Selection Policy

The third policy is critical to efficient and flexible operation of GC. Rather than enqueue a list of move operations followed by one erase, we enqueue *cleanup* operations that represent one step in cleaning a block. The "Cleanup Operation Selection" policy in Figure 4 determines whether to start a read, write or erase operation. Delaying the choice of which page to move allows GC to adapt as pages become invalid due to external writes.

With GC policy reduced to the decision of executing one flash operation at a time, the particular algorithm is simple. Erasing fully invalidated blocks is the best option. When no such blocks are available, we move a page from a block with the least number of valid pages. A move begins with a read operation which, once complete

Parameter	Value
Channels	4 or 8
Dies per channel	2 or 16
Blocks per chip	2048
Pages per block	64 or 128
Bytes per page	4096
Fast Page Read Latency	27 μs
Slow Page Read Latency	40 μs
Fast Page Write Latency	253 μs
Slow Page Write Latency	1359 μs
Erase Latency	2871 μs

Table 1: **SSD Configuration** Architectural dimensions of the flash array and operation latencies to the flash chips.

pushes the paired write operation to the front of the queue from where the cleanup operation originated.

We will use this platform to demonstrate how to more effectively harness the variable performance available in high density flash. Many of these concepts and algorithms will transfer to the more memory-efficient hybrid FTL designs.

3.3 Simulation Setup

To evaluate these alternative organizations, we have developed a detailed trace-driven flash storage system simulator. It supports parallel operations between flash devices, models the flash buses and implements our FTL.

Table 1 details the array's dimensions. We model two moderately-sized SSDs – one to quickly simulate results for our microbenchmarks and a larger configuration to run the workloads. We also simulate an *All Fast* configuration, which models a half-capacity SLC-speed array by (1) reducing block size from 128 to 64 pages and (2) using only the fast read and write latencies.

Our SSD manages the array of flash chips and presents a block-based interface. The controller in the SSD coordinates 4 or 8 channels that each connect 2 chips to the controller via a 400 MB/s bus. Larger SSD configurations are possible, but the configurations we choose provide similar performance trends with much shorter simulation times.

To ensure steady state behavior, we arrange all of the LBAs randomly throughout the chips in the SSD before starting the simulations. We add enough invalidated pages to fill all blocks to the background threshold. The write points begin on a random page in the write point's assigned block.

4 Leveraging Variability

In this section, we describe our mechanisms for scheduling flash operations based on flash page performance variation. We demonstrate how careful, variation-aware scheduling can improve performance under both bursty and sustained workloads. With both mechanisms, we show how increasing the number of write points on each chip increases the FTL's ability to leverage the variability in its flash array.

4.1 Many Write Points for More Flexibility

Making good scheduling decisions requires the scheduler to have multiple options available, and without multiple options, no scheduling policy can have much impact on performance. Since each write point is associated with a single block, and the FTL must write to pages in the block in order, a single write point offers limited options: The FTL can either write to the next page (which may not be the type of page it would prefer) or it can skip the page, writing to the page of its choice, but wasting space.

Our baseline FTL maintains one write point per chip, which can only provide multiple options under light load (and some chips are idle). Under heavy load the FTL's only choice is to schedule an access to the most recently idled chip. Even under light load, a large burst of write traffic will use up the fast pages available on each write point. Both of these scenarios force the FTL to choose between the two undesirable options described above.

To provide flexibility, we extend the baseline FTL with multiple write points per chip, ensuring that the FTL will have choices and can make wise scheduling decisions. In the following subsections, we demonstrate how increasing the number of write points in the system and on each chip increases the policies' ability to access its desired page type.

While additional write points provide the flexibility to access fast and slow pages on demand, their number and use constitute a trade-off with over-provisioned capacity and data placement policies the FTL designer wishes to incorporate. Because each write point requires an open block, when the FTL maintains too many write points the over-provisioned space becomes too fractured across open blocks. In particular, the number of blocks between the background and emergency thresholds (for the GC routines described in section 3) provide a hard limit for the possible number of write points in our design. The FTL designer will also have to carefully weigh the value of placing data to potentially improve the efficiency of future GC with the effects of using a high or low latency page.

4.2 Handling Bursty Workloads

In this section, we present a policy called *Return to Fast* (RTF) that allows the FTL to service bursts of performance-critical operations exclusively with fast pages. The algorithm seamlessly provides nearly the speed of SLC while using all of the MLC pages.

We can apply the RTF policy in a number of situations. With an interface that passes information about the criticality of writes to the device, the system could schedule critical operations to fast pages. Such an interface could, for example, enable fast distributed locking protocols that require persistent writes for ordering via a log.

Even without changes to the interface, we can significantly enhance the performance of bursty workloads by treating user accesses as performance critical and GC operations as non-critical. In this case, we use fast pages exclusively until we run out, and then return to our baseline policy. We focus on this application in this paper.

RTF aims to service as many external writes as possible with fast pages. One approach is to skip over slow pages in order to move write points to the fast pages, but that would waste those skipped pages – reducing SSD capacity, invoking GC sooner, and increasing wear and potentially decreasing performance.

RTF avoids skipping pages by returning all write points to fast pages during the idle periods through GC writes. The FTL saves up a reserve of fast pages which it can spend on performance-critical operations. The number of write points in the system controls the size of reserve of fast pages.

The most common pattern of fast and slow pages provides up to two fast pages per write point. The FTL can fully exploit both pages in *Strongly RTF*, which ensures the write points reach the first of the pair of fast pages. The FTL can store an average of 1.5 writes per write point in *Weakly RTF*, which returns the write points to any fast page. Strongly RTF will give us the largest number of fast pages available after a large enough idle period.

We can further enhance the FTL with *preemptive GC*. During idle periods, the FTL continues to GC until each write point points to a fast page. This runs the risk of increased wear, when external writes or trims invalidate the pre-emptively moved data. However, simulation results show this is not a problem.

Increasing the number of write points in a system increases the performance of the bursts, even when the workload is a complex mix of reads, writes and potentially short idle times. In order for the FTL to direct an external write to a fast page, (1) there must be a write point already pointing at a fast page and (2) this write point must point to a chip which is not busy with another operation. Under a complex workload, the number of write points in the system is directly related to the likelihood of both of these conditions. The more write points there are, the more write points there will be pointing to fast pages. So, even with very little idle time we have increased the number of fast pages for the next burst.

A similar argument holds when you consider the contention over access to chips in the system. Imagine all

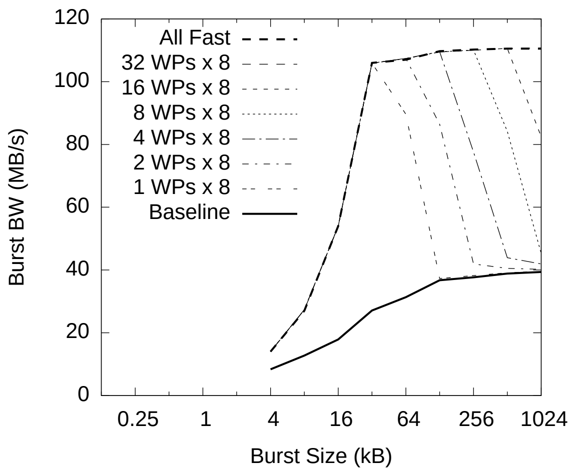

Figure 5: **Performance of Weakly RTF** The weakly RTF policy maintains performance comparable to using only fast pages for burst sizes up to the number of write points before dropping to the performance of using all page speeds.

but one of the chips in the array are blocked with operations. The single available chip is more likely to have a fast page available if there are more write points (and more possible pages available).

4.2.1 Evaluating RTF

We explore the potential of the RTF policy by studying its behavior under a synthetic workload of page-sized accesses to uniformly distributed LBAs, grouped into bursts. The gap between bursts is sufficient to complete all necessary GC and return all the write points to fast pages, when applicable. Each trace uses a different burst size from 4 kB to 4 MB (1 to 1024 pages) and writes a total of 16 MB of data.

Figure 5 shows the performance of the Weakly RTF policy for 1-32 write points per chip on an 8 chip array (x8). For burst sizes less than 32 kB, the array is underused, but as the burst size reaches between one and two pages per chip the performance increases significantly for RTF and the All-Fast configuration. The baseline remains low with a maximum performance of 39.4 MB/s because it uses both fast and slow pages.

At burst sizes greater than 32kB, we observe the positive effect of additional write points in enabling RTF. With one write point, the FTL can manage only short bursts at high speed. Increasing the number of write points per chip provides a larger reserve of fast pages from which to draw and lets the scheduler make better decisions. For weakly (strongly) RTF, the maximum burst size serviced at high speed is equal to (2$\times$) the number of write points in the system times the page size.

4.3 Sustained Write-Intensive Workloads

RTF provides an effective tool for selective performance enhancement. However, under sustained write traffic, external operations must compete for resources with GC, which eclipses the performance benefits of RTF.

In this section, we develop a rate matching technique that allocates fast and slow SSD resources among GC and external operations for the best performance during long periods of sustained load. We begin with a variability-informed analytical model of an FTL, its page scheduling policy, and its GC. The model shows that in most cases the intuitive choice for page variability will lead to performance losses while the counter-intuitive choice improves performance. Finally, we study the potential of the FTL operating with these parameters.

4.3.1 Analyzing FTL Behavior Under Load

In order to maintain the erased block pool during periods of sustained, heavy load, the FTL must match the rate at which it erases pages with its external write rate. The per-chip bandwidths for these two operations remains constant, so the FTL matches these rates by establishing the correct number of chips performing each of the two sets of operations. Equations 1 and 2 describe the two per-chip bandwidths. For Equation 2, we assume 20% over-provisioning and include a parameter (*pgsMvd*) for the number of page moves GC must perform on the average block (which is determined by the workload's locality).

$$ExternalWriteBW = \frac{pageSize}{wLat} \tag{1}$$

$$GC\_BW = \frac{0.2 * blockSize}{pgsMvd * (mvLat) + eLat} \tag{2}$$

With respect to write latency variability, we consider two choices. The FTL could use slow pages to service GC writes and fast pages to service user writes (SGC), or vice versa (FGC).

Figure 6 plots the SSD's bandwidth for these policies and a baseline, latency agnostic configuration over a range of workload localities. Our model assumes the FTL always has access to the preferred page speed without skipping pages. For the FGC configuration, for example, we determine the per-chip user write BW and the cleaning BW using slow page write latency for Equation 1 and fast page write latency for Equation 2, respectively. The ratio of the two yields the correct ratio of chips to use for each operation. The chip counts are averaged over time, so they do not need to be integers. Ultimately these values yield the user-visible write bandwidth.

Without the analytical model, our initial choice was to accelerate external operations, corresponding to the SGC

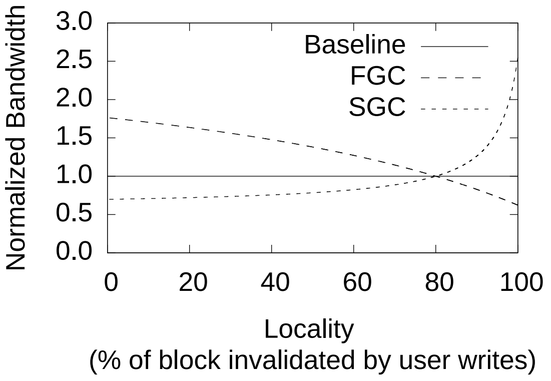

Figure 6: **Design Space for Rate Matching** Which configuration to use under heavy load depends on the workload's locality. If locality is low (less than 80% on this graph), GC must move lots of data and prioritize those writes to fast pages to improve overall performance.

configuration. However, as Figure 6 shows, the highest performance configuration allocates fast pages to online GC instead (FGC).

Scenarios with average to low page locality will do best under FGC, because GC reclaims relatively few erased pages for many moves. SGC experiences a disadvantage because fast user writes and slow GC writes exacerbate the inherent slowness of GC. FGC, on the other hand, uses the speed of fast pages to help GC to keep pace with the user accesses. Because block erase is necessary, and such a heavy weight process, the FTL does best by completing it quickly.

The crossover point falls exactly at 80% locality because of the particular amount of over-provisioning in our array (20%). The analytical model frees 20% of the pages in a block for the average whole-block GC sequence. With 80% locality, the number of pages erased per block GC equals the number of pages moved, and so external write BW is the same as GC write bandwidth for all configurations. As locality decreases from this crossover point, GC requires more moves and higher-performing writes (FGC).

In order to study FGC and SGC, we make two changes to the baseline FTL. The first does not include knowledge of page variability and is simply to maintain the pool under sustained write traffic. To do this, we modify the operation selection policy. We calculate the ratio of per-chip GC bandwidth to per-chip external write bandwidth, called the *target ratio*. The FTL maintains a *chip use ratio* by monitoring the ratio of time spent on GC and external write operations for the recent history. The FTL then chooses the next operation by attempting to match the chip use ratio to the target ratio.

The second policy change accounts for page variability in the data placement policy by directing pages to match

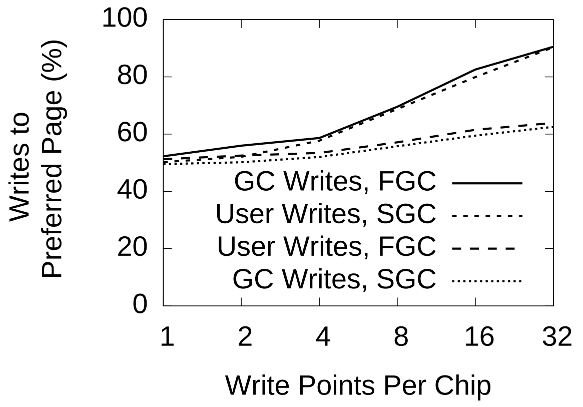

Figure 7: **Page Preference Improvement** Increasing the number of write points per chip increases the availability of the preferred page type when the SSD is under heavy load.

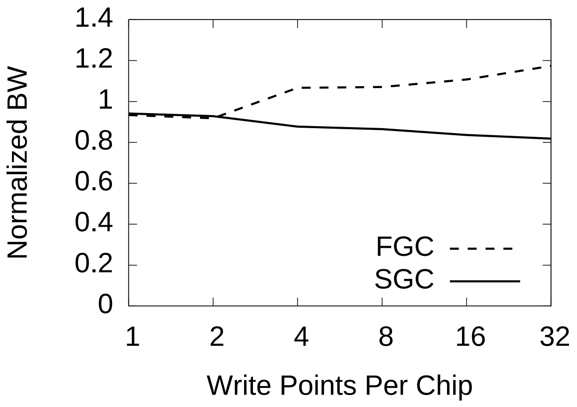

Figure 8: **Sustained Performance** Adding write points allows fast online GC to improve SSD performance by 20%.

either the SGC or FGC configurations. We implement a page preference policy whereby given the choice between several locations to write, the FTL prefers to direct the previously chosen operation according to the SGC or FGC configuration.

The baseline for studying the FTL under sustained load includes the changes to the operation choice policy, but retains the original round robin baseline for the write point choice policy.

4.3.2 Evaluating FGC and SGC

To study rate matching with page preference under the complex constraints imposed by a real FTL, we apply a write-intensive synthetic load to our simulator. The workload consists of 5 s pulses of infinite load followed by 4 s of idleness. This cycles repeats 80 times, and the load consists purely of writes with evenly distributed LBAs.

Under such a load, all operations reach the Data Placement policy with only one idle chip in the flash array. Because each chip only has one write point, the page preference has no effect, and all operations have an equal probability of being written to fast or slow pages. Skipping pages is not a good option because its negative effect on performance overwhelms any advantage gained from using fast pages, due to the added GC.

Write points again provide the flexibility needed for the FTL to leverage the fast pages in the FTL. With multiple write points on each chip, when the operation arrives with only one idle chip from which to select, it still has multiple options for where it can write.

Figure 7 shows how, as the number of write points increases, the FTL can run operations on the desired pages type more frequently. With one write point, both SGC and FGC direct their operations to the two page speeds with equal probability. As the number of write points increases, a larger percentage of operations are scheduled to their preferred page speed. This is especially true when that preferred page speed is fast.

Figure 8 shows the performance resulting from the FTL accessing its preferred pages more often, normalized to the baseline of no page preference. As more write points allow the FTL to select its preferences, the performance of FGC improves while the performance of SGC declines.

These results verify that the optimal choice for page preference under heavy write load is to save fast pages for servicing online garbage-collecting moves (FGC), and that increasing the number of write points on each chip better enables the FTL to tap into that supply of fast pages.

5 Results

In this section we evaluate the effectiveness of our variability aware FTL policies – RTF, FGC and SGC – on a set of five benchmarks.

5.1 Workloads

Table 2 describes the five trace files we use to explore our proposed FTL enhancements. Their burst sizes span a range as do the idle times between each burst.

The conventional method of replaying traces does not accurately retain fixed computation time (seen by the SSD as idle time). This runs the risk of mixing the idle and active parts of the workload which could both (1) eat into the idle time needed for RTF and (2) lessen the load FGC and SGC are intended to accommodate.

We pre-process our trace files to alleviate these problems. Instead of each trace line indicating what time it arrives at the SSD, it indicates how much later than the previous trace line it arrives. Then, if the delta is below a

Trace Name	Min. Δ (Thresh.)	Avg. Burst Size (pgs)	Avg. Idle Time (s)	Description
Build	0.087 s	3.56	1.74	Compilation of the Linux 2.6 kernel.
Financial	18 ms	0.140	0.0620	Live OLTP trace for financial transactions.
WebIndex	48 μs	212	0.000564	Indexing of webpages using Hadoop.
Swap	150 ms	0.0645	0.0218	Virtual memory trace for desktop applications.
DeskDev	0.7 s	4.48	3.82	24 hour trace of a software development work station.

Table 2: **Workload Statistics** Characteristics of the burstiness of our tracefiles and the idle times between the bursts.

particular per-trace threshold, we group that access in the same burst with the previous access by setting the delta to zero. In this way, we ensure the SSD experiences the full brunt of the burst without added idle time.

We assume that a large enough idle period (i.e. that greater than the threshold) indicates the program is executing calculations using the previous burst's data. We also assume that the amount of time before issuing its next burst will remain constant for a given processor architecture. We then enforce the delta time between each burst by issuing the first access of a given burst *delta* seconds after the previous burst completes (i.e. after the completion of the last access).

We set the delta threshold to be the average time between each trace line for a given file. Table 2 details the delta threshold for each trace file as well as the average size of the bursts and average amount of idle time between them.

Measuring the performance of an SSD running a trace file that includes idle time requires some care. To factor out the effect of idle time in the trace file, we divide the amount of data written in a given burst by the time it takes to complete that burst (this is the burst's write bandwidth). We then report the average of these bandwidths for each policy normalized to the baseline.

5.2 Return To Fast

Figure 9 shows the performance of the delta traces running under the Strongly RTF (sRTF) and weakly RTF (wRTF) policies with 1, 8 and 32 write points per chip. The All-Fast configuration shows a potential for 19% to 62% increase in write performance (34% on average) over the baseline and all traces realize at least a portion of these gains. On average, traces realize a 9% performance increase going from 1 to 32 write points per chip and no increase in performance for using strongly RTF rather than weakly RTF.

Financial (*Fin.* in the figures) works well with RTF – it contains a significant amount of idle time between bursts for recovery, and has very few reads which could block and stall the burst. Financial also has very few writes in each burst, so the SSD is able to realize the full potential of the fast pages with very few write points. For other workloads, added performance comes with more write points because a larger pool of fast pages increases the options for where to write, getting around the effect of blocking reads. All workloads on both strongly and weakly RTF achieve more than 24% of the All Fast configuration's gains and most see more than 64%.

While RTF consistently improves the write performance, it has negligible effect on the read performance. On average the RTF configurations gain less than 0.1% in read bandwidth.

Figure 10 shows the wear out experienced by our SSD under the different workloads and RTF configurations. Trying to achieve high performance by using only the fast pages significantly increases the wear – up to 2.0×, and 1.7× on average. However, if we instead fill the slow areas with garbage collected data we were planning on moving anyway, our wear increases by 5% relative to the baseline on average, and never more than 34%.

5.3 Rate Matching with FGC and SGC

Figure 11 shows the performance of the traces running on the FGC and SGC rate matching policies using 1, 8 and 32 write points per chip. The All-Fast configuration is able to realize much larger gains over the baseline, because the FTL makes use of all of the pages during external activity. Even so, the FGC configuration on most workloads achieves a significant portion of these gains while the more intuitive SGC configuration remains at baseline levels. DeskDev reaches the highest performance at 95% above baseline, and the average of all the traces except for WebIndex reaches 65% over baseline.

The spacial locality in the WebIndex's writes set this workload apart – in this case the intuitive choice of directing external operations to fast pages (SGC) provides better performance. WebIndex exhibits an average of 31% fewer moves per erase, placing it in the right-most region of Figure 6. The advantage of saving fast pages for online operations in FGC is a result of completing GC as fast as possible to match the rate of external writes. However, when the access stream exhibits good spacial locality, the act of writing external operations invalidates pages on a small set of blocks, accelerating GC.

Increasing the number of write points on each chip al-

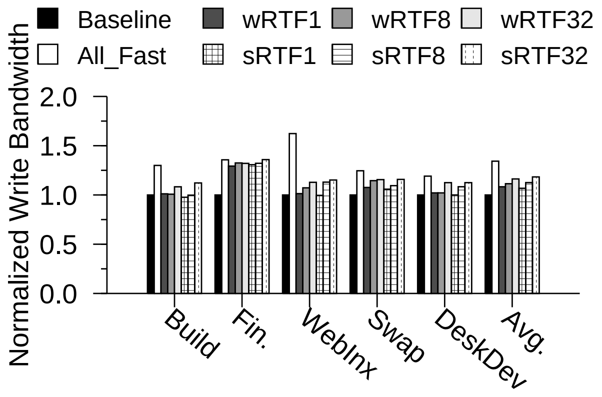

Figure 9: **Performance of RTF** More write points in the flash array increases the reserve of fast pages the FTL can build during idle periods, allowing the FTL to absorb larger burst with only fast pages.

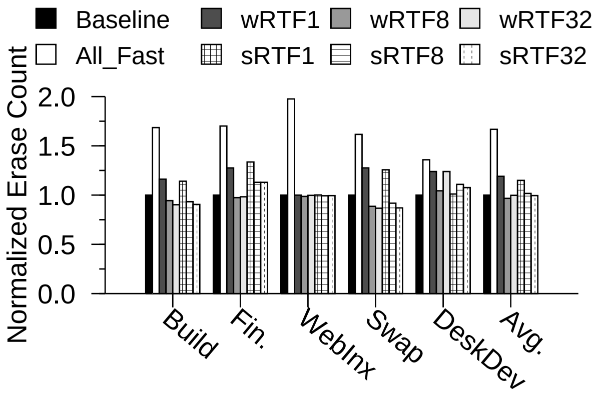

Figure 10: **Wear of RTF** While RTF improves performance, on average its wear is nearly that of the baseline.

lows each configuration to approach the predicted behavior. SGC almost always performs on par or worse than the baseline, often declining from baseline as the number of write points decreases. The opposite trend holds for FGC, frequently beginning with a performance better than baseline and increasing as the number of write points increases. This makes sense because increasing the number of write points increases the impact of each policy. Since SGC hurts performance, adding write points makes performance worse.

While FGC and SGC produce performance gains and losses, respectively, in most cases they both perform a number of erases on-par with the baseline (Figure 12). Excluding WebIndex, the erase count declines by as much as 32% for the SGC-32 configuration on DeskDev, and increases by no more than 2% (Excluding Financial). On average, FGC and SGC experience a 3% decline in wear while the All-Fast configuration is 56% more wear compared to the baseline.

6 Application

Although we propose distinct mechanisms for bursts and heavy load, we discuss their coordination with other policies in the system to address a variety of workloads with mixed access patterns. This section describes how this could be done either through coordination with the operating system or by further enhancing the FTL.

OS Support Coordination with the operating system constitutes one avenue of leveraging the Harey Tortoise techniques. The OS could provide hints with the accesses made to the SSD. For example, the FTL could use RTF to service latency-critical accesses (marked as high priority), providing the functionality of the variability aware FTL in [13] without the added wear. Alternately, the OS could signal a course-grained switch between workload style when, for example, a server transitions between workloads or activities that change between peak and off-peak periods. An enhanced interface, such as NVMe [2], would facilitate these implementations.

Dynamic FTL Without hints from the OS, the FTL could combine the Harey Tortoise's policies to accommodate mixed workloads. It would adjust as a burst of accesses of unknown length progresses – employing RTF early in the burst before transitioning to RM techniques as the "burst" lengthens to a sustained load. This technique would result in RTF accommodating small bursts while the FTL treats long bursts with RM techniques.

For long bursts and sustained load, the FTL would step through several phases combining our techniques proposed in this work. For such a policy, GC during idle period should employ RTF to return as many write points as possible to fast pages. Then, when accesses arrive, the FTL would achieve maximum possible performance from using only fast pages under RTF, before gradually transitioning to RM policies.

During the transition period the FTL would (1) adjust the preference for fast or slow pages of the external and GC writes and (2) tailor the use and cleaning rates to use up the over-provisioned space and create a graceful degradation of performance. The latter could be achieved by relating the target and chip time ratios by some factor which dynamically adjusts to one.

Finally, when the pool of erased blocks reaches a sustainable minimum, the FTL would work exclusively with the RM policies until an idle period allows for additional GC. In this way, the FTL would provide high performance to small bursts and gradually ramp down to a maximum, sustainable performance.

The inversion of preference (for RM) with good write locality suggests another dimension for exploring how to detect and adapt to the correct choice of page preference.

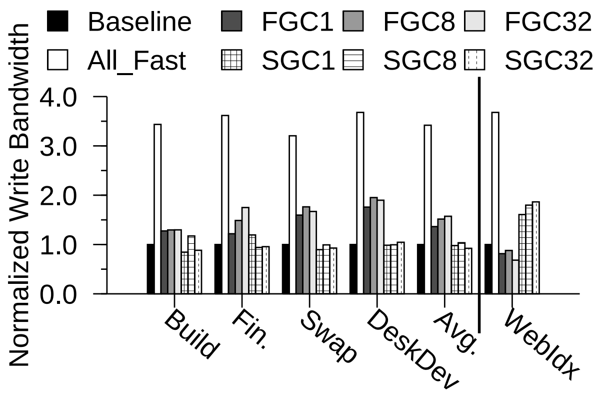

Figure 11: **Performance of FGC and SGC** The counter intuitive choice of servicing online operations with fast pages (FGC) improves the performance, when spacial locality is low.

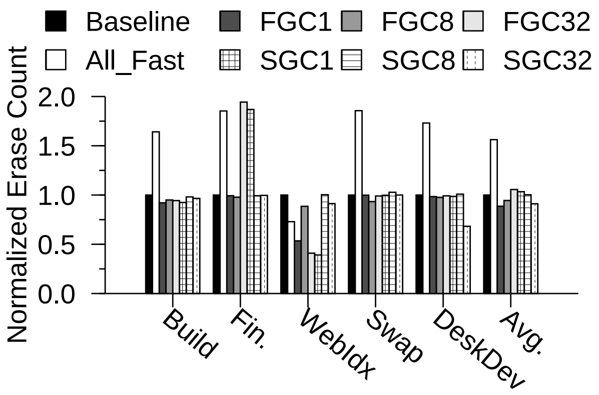

Figure 12: **Wear of FGC and SGC** Leveraging page variability during heavy load does not effect device wear out in most cases.

7 Related Work

There is a large body of flash-based storage research spurred on by the promise of high performance, low energy, and the limitations imposed by its idiosyncrasies. The research most closely related to our work falls in four categories: Mode-switching Flash, FTL algorithms, SSD interleaving, and write buffers. All of these topics try to improve the performance, endurance and/or reliability of the SSD, but do not leverage or address the variability inherent in MLC flash. The final section of related work discusses the emerging research that embraces flash page variability.

Mode-Switching Flash: Changing the cell bit density has been proposed in research [18] and implemented by SSD vendors [24, 20] to improve reliability, endurance, and performance. Switching between MLC mode and SLC mode does have the drawback of sacrificing half of the system capacity. In our work, by leveraging write latency asymmetry across the pages, we are able to approximate the performance of SLC without sacrificing device capacity, the best of both worlds. Furthermore, because we use all the pages in the block by not throwing away the slow pages, we reduce the number of erase cycles, improving overall system endurance and reliability.

FTL Algorithms: There is a large body of work focused on FTL optimizations to improve SSD performance, endurance and reduce memory overhead based on access pattern or application behavior. By using an adaptive page- and block-level addressing mapping scheme, KAST [17], ROSE [10] and WAFTL [27] are able to improve performance, reduce garbage collection overhead and reduce FTL address mapping table size. DFTL [15] goes one step further by caching a portion of the page-level address mapping table for reduced size and fast translation. MNFTL [23] reduces the number of valid page copies for garbage collection, explicitly targeting MLC flash. Finally, CAFTL [9], removes unnecessary duplicate writes and increases the lifespan of the SSD. While some of these FTLs address workload variability, none address the variability in the underlying MLC flash.

SSD Interleaving: Intra-SSD parallelism has been explored by many groups [3, 7, 22, 28, 8, 25, 4]. By not only issuing operations in parallel at the package-, die-, and plane-level, others have also shown that rescheduling operations can improve performance [28]. Our work dives deeper into parallel data placement by providing multiple write points for fast pages within the plane that can adsorb burst and sustain high write performance, on par with SLC devices.

Write Buffers: Historically, buffers have been used in HDD to improve read and write performance. Likewise, write buffers have been shown to improve random write performance in SSDs [19]. These write buffers are also sufficient for handling small burst sizes. More recently, research has shown that per package queues and operation reordering provide more opportunities for parallel operations and further improve performance over LRU-based write buffer mechanisms [25]. Write points can be used in conjunction with write buffers, providing the FTL with more flexibility in data placement, in light of the performance asymmetries that exist in MLC flash.

Variability: The quest for higher density flash has provided opportunities to exploit the variability in flash page latency. Previous work [13] has exposed these asymmetries and predicted their impact on future SSDs [14]. Other work has exploited the differences in the flash to improve error correction [12] or guarantee other properties, like secure erasure [26]. We demonstrate that the FTL can take advantage of flash variability to improve performance while not sacrificing endurance or capacity.

8 Conclusion

In this paper, we developed an FTL that leverages systematic variability in flash memory to provide the speed of the hare (SLC) with the capacity of the tortoise (MLC). We propose increasing the number of write points on each chip to increase the flexibility of the FTL to schedule accesses to pages with a variety of latencies, and we demonstrate how to use this flexibility to achieve up to 100% of the performance an SLC array (or an average of 89%) by using MLC flash without additional wear. Further, we show that the counterintuitive approach of scheduling garbage collection operations on fast pages improves performance by an average of 65% and as much as 95% in workloads with little spacial locality.

Acknowledgements

We would like to thank the reviewers and shepherd of this paper for their valuable input. This work was supported by the NSF Variability Expedition under award number 1029783.

References

[1] Densbits technologies. memory modem: Technology overview. April 2012.

[2] Nvm express. http://www.nvmexpress.org/. 2013.

[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, 2008.

[4] S. Bai and X.-L. Liao. A parallel flash translation layer based on page group-block hybrid-mapping method. *Consumer Electronics, IEEE Transactions on*, may 2012.

[5] A. Berman and Y. Birk. Constrained Flash memory programming. In *IEEE International Symposium on Information Theory*, pages 2128–2132, 2011.

[6] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A design for high-performance flash disks. Technical Report MSR-TR-2005-176, Microsoft Research, December 2005.

[7] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Architectural Support for Programming Languages and Operating Systems*, pages 217–228, 2009.

[8] F. Chen, R. Lee, and X. Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, 2011.

[9] F. Chen, T. Luo, and X. Zhang. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory based Solid State Drives. In *USENIX Conference on File and Storage Technologies*, pages 77–90, 2011.

[10] M.-L. Chiao and D.-W. Chang. ROSE: A Novel Flash Translation Layer for NAND Flash Memory Based on Hybrid Address Translation. *IEEE Transactions on Computers*, 60:753–766, 2011.

[11] H.-T. L. et. al. Radically extending the cycling endurance of flash memory (to ¿100m cycles) by using built-in thermal annealing to self-heal the stress-induced damage. 2012.

[12] R. Gabrys, E. Yaakobi, L. M. Grupp, S. Swanson, and L. Dolecek. Tackling intracell variability in tlc flash through tensor product codes. In *International Symposium on Information Theory*, ISIT, 2012.

[13] 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 *International Symposium on Microarchitecture*, pages 24–33, 2009.

[14] L. M. Grupp, J. D. Davis, and S. Swanson. The Bleak Future of NAND Flash Memory. In *USENIX Conference on File and Storage Technologies*, 2012.

[15] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Architectural Support for Programming Languages and Operating Systems*, pages 229–240, 2009.

[16] K.-C. Ho, P.-C. Fang, H.-P. Li, C.-Y. Wang, and H.-C. Chang. A 45nm 6b/cell Charge-Trapping Flash Memory Using LDPC-Based ECC and Drift-Immune Soft-Sensing Engine. In *Solid-State Circuits IEEE International Conference*, 2013.

[17] H. jin Cho, D. Shin, and Y. I. Eom. KAST: K-associative sector translation for NAND flash memory in real-time systems. In *Design, Automation, and Test in Europe*, pages 507–512, 2009.

[18] T. Kgil, D. Roberts, and T. Mudge. Improving nand flash based disk caches. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, june 2008.

[19] H. Kim and S. Ahn. Bplru: a buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, 2008.

[20] G. e. a. Marotta. A 3bit/cell 32gb nand flash memory at 34nm with 6mb/s program throughput and with dynamic 2b/cell blocks configuration mode for a program throughput increase up to 13mb/s. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 444 –445, feb. 2010.

[21] K.-T. Park, M. Kang, D. Kim, S.-W. Hwang, B. Y. Choi, Y.-T. Lee, C. Kim, and K. Kim. A Zeroing Cell-to-Cell Interference Page Architecture With Temporary LSB Storing and Parallel MSB Program Scheme for MLC NAND Flash Memories. *IEEE Journal of Solid-state Circuits*, 43:919–928, 2008.

[22] S.-H. Park, S.-H. Ha, K. Bang, and E.-Y. Chung. Design and analysis of flash translation layers for multi-channel nand flash-based storage devices. *Consumer Electronics, IEEE Transactions on*, august 2009.

[23] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan. Mnftl: An efficient flash translation layer for mlc nand flash memory storage systems. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, 2011.

[24] D. Raffo. Fusionio builds ssd bridge between slc,mlc, july 2009.

[25] X. Ruan, Z. Zong, M. I. Alghamdi, Y. Tian, X. Jiang, and X. Qin. Improving write performance by enhancing internal parallelism of solid state drives. In *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International*, dec. 2012.

[26] M. Wei, L. M. Grupp, F. E. Spada, and S. Swanson. Reliably erasing data from flash-based solid state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, 2011.

[27] Q. Wei, B. Gong, S. Pathak, B. Veeravalli, L. Zeng, and K. Okada. WAFTL: A workload adaptive flash translation layer with data partition. In *Symposium on Mass Storage Systems*, pages 1–12, 2011.

[28] S. yeong Park, E. Seo, J.-Y. Shin, S. Maeng, and J. Lee. Exploiting internal parallelism of flash-based ssds. *Computer Architecture Letters*, jan. 2010.