Improving Write Performance by Enhancing Internal Parallelism of Solid State Drives

Xiaojun Ruan
Department of Computer Science
West Chester University of Pennsylvania
West Chester, PA 19383
Email: xruan@wcupa.edu

Mohammed I. Alghamdi
Department of Computer Science
Al-Baha University, Al-Baha City,
Kingdom of Saudi Arabia
Email: mialmushilah@bu.edu.sa

Xunfei Jiang
Computer Science & Software Engineering
Auburn University, Auburn,
Alabama 36849-5347
Email: tianyun@eng.auburn.edu

Ziliang Zong
Department of Computer Science
Texas State University, San Marcos
TX 78666-4616
Email: zz11@txstate.edu

Yun Tian
Computer Science & Software Engineering
Auburn University, Auburn,
AL 36849-5347
Email: tianyun@eng.auburn.edu

Xiao Qin†
Computer Science & Software Engineering
Auburn University, Auburn,
AL 36849-5347
Email: xqin@auburn.edu

***Abstract*—Most researches of Solid State Drives (SSDs) architectures rely on Flash Translation Layer (FTL) algorithms and wear-leveling; however, internal parallelism in Solid State Drives has not been well explored. In this research, we proposed a new strategy to improve SSD write performance by enhancing internal parallelism inside SSDs. A SDRAM buffer is added in the design for buffering and scheduling write requests. Because the same logical block numbers may be translated to different physical numbers at different times in FTL, the on-board SDRAM buffer is used to buffer requests at the lower level of FTL. When the buffer is full, same amount of data will be assigned to each storage package in SSDs to enhance internal parallelism. To accurately evaluate performance, we use both synthetic workloads and real-world applications in experiments. We compare the enhanced internal parallelism scheme with the traditional LRU strategy since it is unfair to compare an SSD having buffer with an SSD without a buffer. The simulation results demonstrate that the writing performance of our design is significantly improved compared with the LRU-cache strategy with the same amount of buffer sizes.**

I. Introduction

Hard drives have been the most widely deployed storage media for many years. Compared with hard drives, solid state drives average cost per MB is currently much higher [22]. Although SSDs are still not as popular as hard drives now, they will certainly be the next preferred storage media because of the high performance and low power cost compared with those of HDDs. Two major challenges of SSDs have been addressed in previous research: random write [5] and reliability. Each block on flash-based storage media such as SSDs and flash drives has limited erasure times. Furthermore, each block has to be erased before it can be rewritten. Erasure operations reduce both performance and SSDs' life times of SSDs [4].

People used to believe that access patterns were not correlated to SSDs performance, but Chen, Koufaty, and Zhang indicated that different access patterns may have negative and positive impacts on internel parallelism [9]. A good FTL algorithm is able to reduce the impact of access patterns by balancing workloads at the block level.

Inter-disk parallelism on hard drives has been well studied for decades. Data Striping, for example, is a typical solution to enhance inter-disk parallelism. Such parallelism is storage-media independent, meaning it is effective for hard drive disks, SSDs, and tapes. Unlike inter-disk parallelism, intra-disk parallelism is closely related to storage media, therefore, intra-SSD parallelism needs to be studied despite the fact that intra-HDD parallelism has already been well explored [28]. Interestingly, because flash-based storage has a unique mechanism, parallelism can be applied in multiple levels, namely, package-level, die-level, and plane-level [10].

Unlike Hard Drives, SSDs have Flash Translation Layers (FTL) performing virtual-to-physical address translations. FTL evenly spreads the erasure workload in flash-based storage. Reliability and performance are the two major research areas of solid state drives. Most current research attempts to design new Flash Translation Layer algorithms to improve reliability and enhance performance. Kim and Ahn proposed a buffer management scheme BPLRU for improving random write performance in flash-based storage. BPLRU buffers improve the performance of random writes [20]. Soundararajan and Prabhakaran presented Griffin, a hybrid storage device, to buffer large sequential writes on Hard Drives [29]. Park and Jung also presented write buffer-aware address mapping for flash memory devices [24].

Previous approaches are similar in that they use another type of storage media as a buffer or cache. In this case, the performance enhancement comes from faster buffers. In aforementioned research, incoming writes are also buffered. But the performance improvement is mainly from intra-disk parallelism caused by interleaving. Hence, it would be unfair to compare the enhanced-internal-parallelism write buffer SSDs with the SSDs that have no buffer. In this paper, all SSD configurations are exactly the same to avoid the performance difference from faster buffers or caches.

978-1-4673-4883-6/12/$31.00 ©2012 IEEE

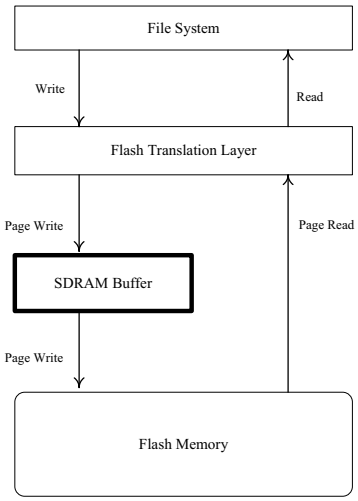


Fig. 1. Architecutre

We collected synthetic workloads, benchmark traces, real-world application traces, and file-writing process traces to test the performance of SSDs. Such traces represent the access patterns of extreme cases, I/O intensive applications, and write intensive file-backup processes. The buffer is designed below the FTL to guarantee data consistency and correctness. To enhance package-level parallelism, there are multiple lists maintained in the buffer. Each list only serves one package. When the buffer is full, the same amount of data (e.g., the same number of pages) is issued to each package to enhance parallelism. Performance evaluation demonstrates that enhancing internal parallelism buffers can significantly improve write performance without increasing buffer size. In other words, our solution uses buffers more efficiently than traditional LRU.

The rest of this paper is organized as follows: Section II describes the design and the algorithm of an enhanced internel parallelism write buffer; Section III presents the methodology used in this paper; Section IV evaluates the system performance with both synthetic workloads and real world applications; Section V presents recent related research; Section VI presents recent related works; Section VII concludes this paper.

II. Design and Algorithm

Unlike hard drive disks, flash-based storage has a Flash Translation Layer (FTL) to map Logic Block Number (LBN) to Physical Block Number (PBN). A flash memory block has very a limited number of erasure times (usually 10,000). Hence, the erasure operations have to be evenly distributed among all blocks. A remapping algorithm is designed in FTL for wear-leveling and load balancing; thereby, spreading the erasure workload and postponing wearing out. Consequently, a block number in the file system level is not fixed mapped to a block number in the flash memory level. Based on different remapping algorithms, one logic block number could be translated to different physical block numbers at different time for wear leveling purposes. Hence, a buffer must be designed at the lower level of FTL to keep data consistency and data correctness. If the buffer is designed in the upper FTL, the buffered requests may be assigned to different page numbers because of different current wear-leveling information. Therefore, pages that are supposed to be updated still contain old information while that pages are not supposed to be updated get overwritten incorrectly.

Fig. 1 presents our architecture for flash memory. We chose Synchronous Dynamic Random Access Memory (SDRAM) as the buffer because of its high performance. Since flash memory performance suffers random writes, the buffer is specifically designed to buffer writes. When a write request arrives,it is first put in the buffer. This data is moved to flash memory chips when package-level parallelism can be enhanced. Read requests access flash memory directly without being buffered; however, when the latest data is from buffered write requests, the buffer also can serve read requests.

Fig. 2 presents the organization structure in a SDRAM Write buffer. Traditionally, all requests in the buffer are logically organized together and pages in the buffer are replaced based on LRU (Least Recently Used) or other cache replacement algorithms. Thus, when data is moved from the buffer to flash memory chips, parallelism can only be triggered accidentally. In a traditional buffer, LRU maintains a queue of pages sorted by time of updating. So, LRU only moves the oldest pages to flash memory chips from the queue. Full package-level parallelism can be triggered only when the package number of continuously buffered pages are all from different packages. Since access patterns, especially those below the FTL, are hard to predict, we will use a simple example to explain the probability of full package-level parallelism. We assume the pages from each package appears at the same rate. Under this assumption, the probability that eight consecutive pages are from eight different packages is 1/8 because there are eight packages in the SSD. The probability of continuous 8 pages are from all different 8 packages is $(1/8)^8$, which is approximately 0.00000596%. This probability is also the rate of full package-level parallelism. Partial package-level parallelism may occur more often because if any two consecutive pages are from two different packages, it is partial package-level parallelism. With a enhanced internal parallelism write buffer, full package-level parallelism can be triggered in almost every data movement between the buffer and the packages. Enhanced full package-level parallelism leads to much higher performance.

In the presented write buffer, multiple lists are maintained for buffering writes for packages. Each list only serves one corresponding package and only buffers write requests for its dedicated package. The lists are dynamic in that they may have different lengths. Since the buffer is built below FTL, the granularity in buffer is page (i.e., 8KB in the design). 1MB buffer can hold 128 page requests and all requests in the SDRAM buffer are writes according to the design. Once the buffer is full, it will issue the same number of pages to each package at same time to enhance write performance because

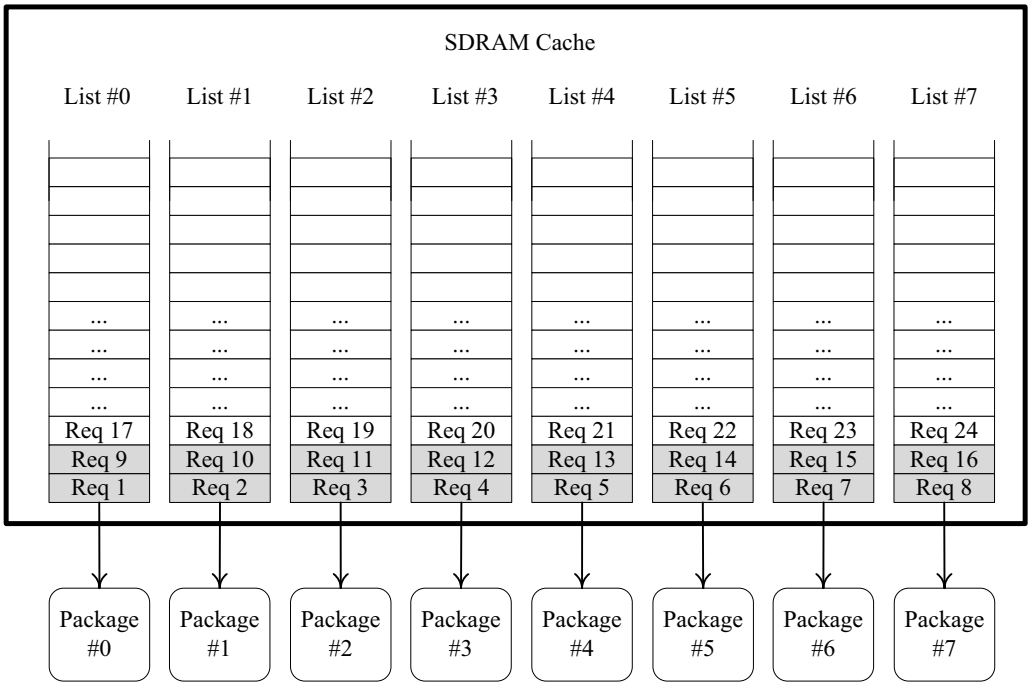


Fig. 2. Design of Internal Parallelism in an SSD

all packages are able to work in parallel.

Algorithm 1 Enhancing Package-Level Parallelism Algorithm. r represents one request. L_i represents the List stores the request in the buffer for the ith package. P_n represents the nth package.

```
if r_current is a write request then
  find its corresponding package P_i
  for r_j ← all requests in L_i do
    if r_j = r_current then
      remove r_j
    end if
    add r_current to L_i^tail
  end for
  if buffer is full then
    for m ← number of parallelism pages do
      for n ← number of packages do
        issue L_n^head to P_n
      end for
    end for
  end if
end if
```

Algorithm 1 is used to enhance parallelism by buffering writes. Upon it's arrival, a write request goes to the corresponding list in the buffer. If the list contains a request having the same page number, the request in the buffer will be replaced by the arrival request which will be put at the end of the list. This actually reduces I/O workload on flash memory chips. The performance improvement of the replaced requests is completely from the SDRAM buffer. When the buffer is full, the same number of requests will be moved from each list to each package on flash storage to enhance package-level parallelism. Even if there are nested loops in the pseudo code, the number of parallelism pages and the number of packages are fixed and relatively small. In our experiments, the number of packages was 8 and the number of parallelism pages was from 8 to 512; therefore the time complexity is still $O(n)$.

III. Methodology

Although SSD manufactures are capable of modifying SSD firmware designs, they are not able to physically implement or modify FTL algorithms on an SSD at the firmware level. This lack of implementation means benchmark applications cannot be used to directly evaluate system performance. Fortunately, there are some well recognized SSD simulators available. After comparing several different existing SSD simulators, we write buffer to implement enhanced internal parallelism in DiskSim 4.0 with a SSD extension.

A. Simulators

DiskSim was originally designed and developed for hard drive disk research by the Parallel Data Lab, at Carnegie Mellon University. It is an efficient, accurate, and highly-configurable simulator. Microsoft Research (MSR) has made an SSD extension for DiskSim 4.0. MSR is not the only one who developed an SSD extension patch for DiskSim. The Computer Systems Lab in the Pennsylvania State University also developed an Object-Oriented Flash Based SSD extension for DiskSim 3.0. Comparing the two simulators, we chose to use the combination of DiskSim 4.0 and MSR's SSD Extension since MSR's SSD Extension provides internal parallelism support.

DiskSim is able to use both external traces and internally-generated synthetic workloads. It supports multiple trace formats including several ASCII trace formats and binary trace formats. New trace formats can also be easily incorporated. We used traces collected from a variety of system platforms and applications to evaluate system performance.

B. Internal-Generated Synthetic Workloads and External Traces

In order to accurately evaluate system performance on DiskSim 4.0 and the MSR SSD extension, We evaluated the I/O performance of our design using different synthetic workloads and external traces. First, we used collected I/O traces representing diverse workloads in addition to synthetic workloads. we collected and tested three types of external traces: simple I/O benchmarks, real data-intensive applications, and files writing operations.

• **Synthetic Workloads:** DiskSim 4.0 contains a synthetic workload generator. Synthetic workloads cannot accurately represent real-world cases. However, synthetic traces can still be used to test system performance in extreme cases. This is possible because workloads and access patterns can easily be manipulated by changing request sizes, block numbers, and inter-arrival time. I/O workloads are categorized as sequential reads, sequential writes, random reads, and random writes with a variety of data sizes.

• **Simple Benchmarks:** MSR's SSD Extension includes traces collected from two I/O intensive benchmarks, namely, IOzone and Postmark. IOzone is adequate for file system analysis and Network File System (NFS) testing [6]. PostMark simulates heavy small-file system loads and provides complete reproducibility. Both IOzone and PostMark traces are made possible by MSR.

• **Real World Applications:** Neither synthetic workloads nor I/O intensive benchmarks are real world applications. Synthetic workloads are generated by an internal workload generator. Although I/O benchmarks can be considered as applications, their workloads are still synthetic. In order to evaluate system performance of real world I/O intensive applications, we collected traces from two real-world applications: the MapReduce Phoenix from Stanford University [26] and the Linux Kernel 2.6.36.4 [1] Compilation. We collected traces from when the Phoenix was running the Word Count application on a 1GB text file. Phoenix intensively accessed an individual swap partition monitored by trace collector programs. Compiling Linux Kernel is also an I/O intensive process. The compiler needs to read source code files and write the output binary files on the same SSD. The kernel source code is stored on a separated partition to avoid unexpected I/O accesses.

• **File Writing Process:** We copied files from one hard drive (source disk) to a separated partition then collected traces only from the separated partition. The workload of the separated partition is mainly writes, so we intended to test the improvement at write performance. We followed three steps to collect real-world traces representing file backup processes. First, we generated ten thousand 100KB files on the source disk; we copied these 10,000 files to a separated partition. Second, we wrote 5.7GB MP3 files (file size varies from 2MB to 10MB) to the separated partition. Third, we generate three 1.9GB files and write them in the separated partition. These operations represent the I/O workloads of small size files, medium size files, and large size files, respectively.

C. Trace Collection

To collect I/O traces of real-world applications and file writing process, there are two trace-collection tools used in this study: DiskMon in Windows [27] and blktrace [3] in Linux. While DiskMon collects ASCII traces, blktrace collects binary traces, which can be further interpreted as ASCII traces. These ASCII traces cannot be processed by DiskSim 4.0 due to the different trace formats, so, we reformatted the traces using a scripting language (i.e., Python).

DiskMon and blktrace collect all I/O operations from a specified partition. To avoid regular Operating System disk operations and I/O of daemon applications, we set up real-world applications on a separated partition for trace collection. Blktrace collects device-level operations thus, we removed device-level operations that were not used by our simulators in traces. After preprocessed the traces, DiskSim could handle the traces and run even-driven simulations based on the traces.

D. SSD Configuration

TABLE I
SPECIFICATION DATA OF THE SSD

Feature	Value
Capacity	64GB
Page Size	8KB
Pages per Block	64
Flash Chip Packages	8
Blocks per Package	16384
Page Read Latency	0.025ms
Page Write Latency	0.200ms
Page Erase Latency	1.5ms
Write Buffer Size	1MB to 64MB

Table I presents the specification data of the SSD. Note that the parameters represent an ideal SSD design. The parameters can be manipulated to tune the SSD performance. There are 8 packages can be accessed in parallel. Write buffer size varies

from 1MB to 64MB. Different configurations have different numbers of pages per block. Flash blocks have to be erased for updating. In other words, if even one page needs to be updated, the entire block (i.e., 64 pages, see the configuration in Table I) must be erased in order to be rewritten. A large number of pages per block leads to long page-erase latency. Page-erase latency is 1.5ms, which is much more expensive than read latency and write latency. The representive SSD has 8 flash chip packages. All eight packages are able to be accessed in parallel. In this research, parallelism is enhanced among those 8 flash chip packages.

IV. Performance Evaluation

A. Experimental Setup

TABLE II
INTERNAL-GENERATED SYNTHETIC WORKLOADS

Name	Feature
sw250k	250,000 synthetic sequential writes in 250KB
rw250k	250,000 synthetic random writes in 250KB
sw5m	250,000 synthetic sequential writes in 5MB
rw5m	250,000 synthetic random writes in 5MB

We tested 13 different traces and benchmarks (see Table II and Table III) on traditional Least Recently Used (LRU) cache algorithm and write buffer algorithm for enhancing internal parallelism. DiskSim has an internal synthetic workload generator, which generates sequential writes at 250KB per request (sw250k), random writes at 250KB per request, sequential writes at 5MB per request (sw5m), and random writes at 5MB per request (rw5m). Each workload contains 250,000 write requests. Each request writes same amount of data. Both IOZone and PostMark are widely used I/O system benchmarks and have many erasure operations [2]. In order to evaluate the performance of real world applications, we collected I/O traces of 5 real world I/O intensive applications: Linux Kernel Compilation, MapReduce Phoenix, small-file writes, MP3 files writes, and large-file writes. Both Linux Kernel Compilation and MapReduce Phoenix contain very intensive reads and writes. The three traces concerning file write operations test the I/O performance of small file access, medium size file access, and large file access, respectively. There are much less erasures than in IOZone because all data is only written once during copy operations.

Table II outlines the features of synthetic workloads. All four synthetic workloads have 250,000 write requests where the tested data sizes are 250KB for small-size requests and 5MB for medium-size requests. Write requests are also categorized into two groups: sequential writes and random writes.

Table III summarizes the detail features of real world applications. Since more than half of the requests of kernel compilation and most requests of MapReduce Phoenix are writes, we not only tested the original Kernel Compilation and MapRedcue traces, but also replaced all read requests with write requests to make two new traces - Kernel Compilation AW (All Writes) and Phoenix AW (All Writes). The three file-write traces 10thou100KB, MP3, and 1.9GB*3, compose 90 to 99 percent of all write requests.

B. Results and Evaluation

Fig. 3(a) shows the performance of the Linux kernel compilation. Although our buffer only enhances write performance and most of the requests in the Linux kernel compilation are reads, our scheme still achieves a significant performance improvement. When buffer size is small, our internal parallelism mechanism outperforms LRU. This is because when buffer size is small, LRU cannot benefit very much from a high performance buffer; consequently, internal parallelism dominates performance improvements. When buffer size is large, most performance improvement is contributed by large buffer size; therefore, performance improvements diminishes.

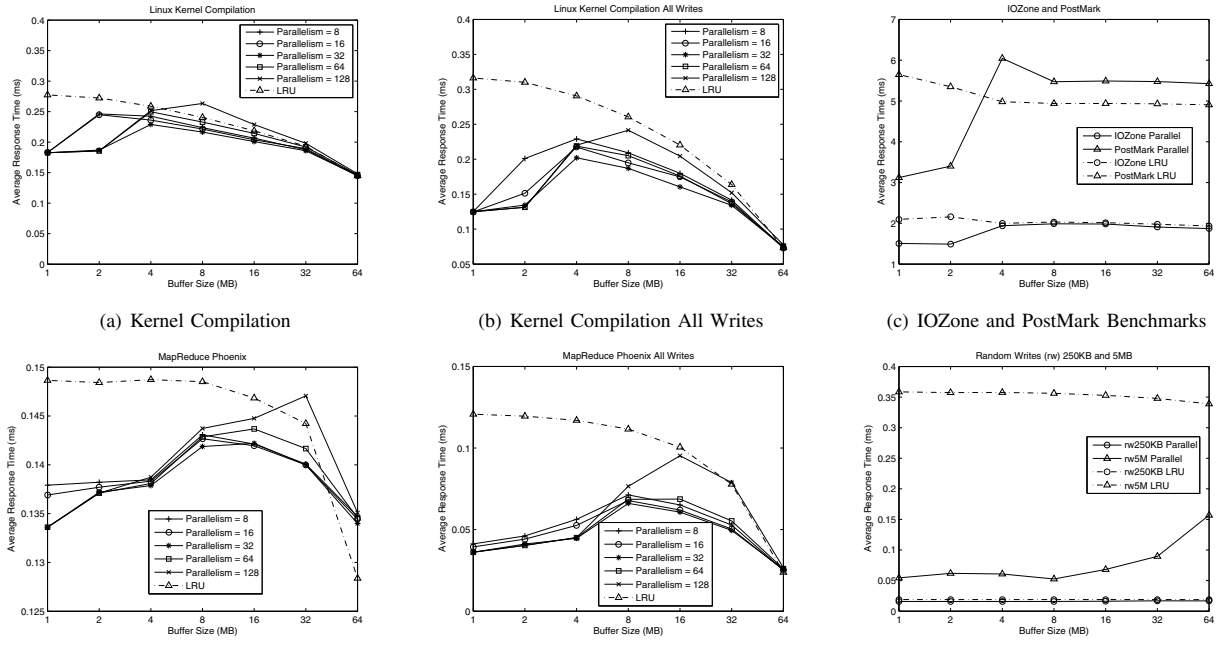
Because most requests are writes, we replaced all the read requests by write requests in Linux kernel compilation trace to focus on system write performance. In Fig. 3(b), the trends demonstrate that when buffer size is small, an enhanced internel parallelism write buffer can more significantly improve system performance than LRU can. When buffer size is increasing, it has pronounced cost-effective impacts on system performance. Hence, LRU and our write buffer scheme have similar average response times. In Fig. 3(a), the write workload is much more intensive than that in Fig. 3(b). Therefore, performance improvement is better in Fig. 3(b) because the parallelism is only enhanced for writes.

Fig. 3(c) indicates that LRU has better performance when buffer size is large in PostMark, meaning heavy small-file reads can hurt the performance of the write buffer. When buffer size is large, internal Parallelism is unable to outperform traditional LRU. The reason is that the parallel writes are issued at inappropriate times (during I/O burstness for example), so the SSD cannot serve a large number of incoming requests immediately. Compared with PostMark, the IOZone trace provided in the SSD Extension only has write requests and we were able to obtain better performance than with PostMark.

Fig. 3(d) and Fig. 3(e) demonstrates similar performance trends as those plotted in Fig. 3(a) and Fig. 3(b). These figures indicate that when buffer size is small, performance improvement achieved by internal parallelism is significant. But large buffer size dominates the performance, making both internal parallelism and LRU achieve similar performance in such cases. Most on-board cache sizes in SSDs are small due to the cost of SDRAM; using a large SDRAM cache as a write buffer is not cost-effective.

Fig. 3(f) reveals the performance in extreme cases by using synthetic workloads. The rw5m provides extremely heavy random write workload, which allow our scheme to achieve the best performance improvement in all experiments.

Fig. 4 shows the performance of 3 file-write traces. Fig. 4(a) shows that when we issued a larger number of pages for parallelism, the performance was worse than that of issuing a small number of pages. This is because the average response



(a) Kernel Compilation (b) Kernel Compilation All Writes (c) IOZone and PostMark Benchmarks

(d) MapReduce Phoenix (e) MapReduce Phoenix All Writes (f) Random Writes 250KB and 5M

Fig. 3. Performance Evaluationn of Synthetic Workloads, Benchmarks, and Real World Applications

TABLE III
TRACES FEATURES

Trace Name	# request	# read	# write	% read	% write	# seq read	# seq write	% seq read	% seq write
iozone	99999	0	99999	0	100	0	50554	0	50.55
postmark	62257	51780	10477	83.17	16.82	27409	1067	44.02	1.71
Kernel Compilation	155097	84849	70248	54.70	45.29	12917	4548	8.32	2.93
Kernel Compilation AW	155097	0	155097	0	100	0	17483	0	11.27
Phoenix	978329	822914	155415	84.11	15.88	330401	14886	33.77	1.52
Phoenix AW	978329	0	978329	0	100	0	345293	0	35.29
10thou100KB	20084	1982	18102	9.86	90.13	773	15233	3.84	75.84
MP3	94473	51	94422	0.05	99.94	46	93353	0.04	98.81
1.9GB*3	95352	191	95161	0.20	99.79	1	89605	0.001	93.97

time was already very small (e.g. smaller than 0.73ms). So, even moving data from the buffer to flash memory chips only introduced little overhead, though it still showed in the performance overhead.

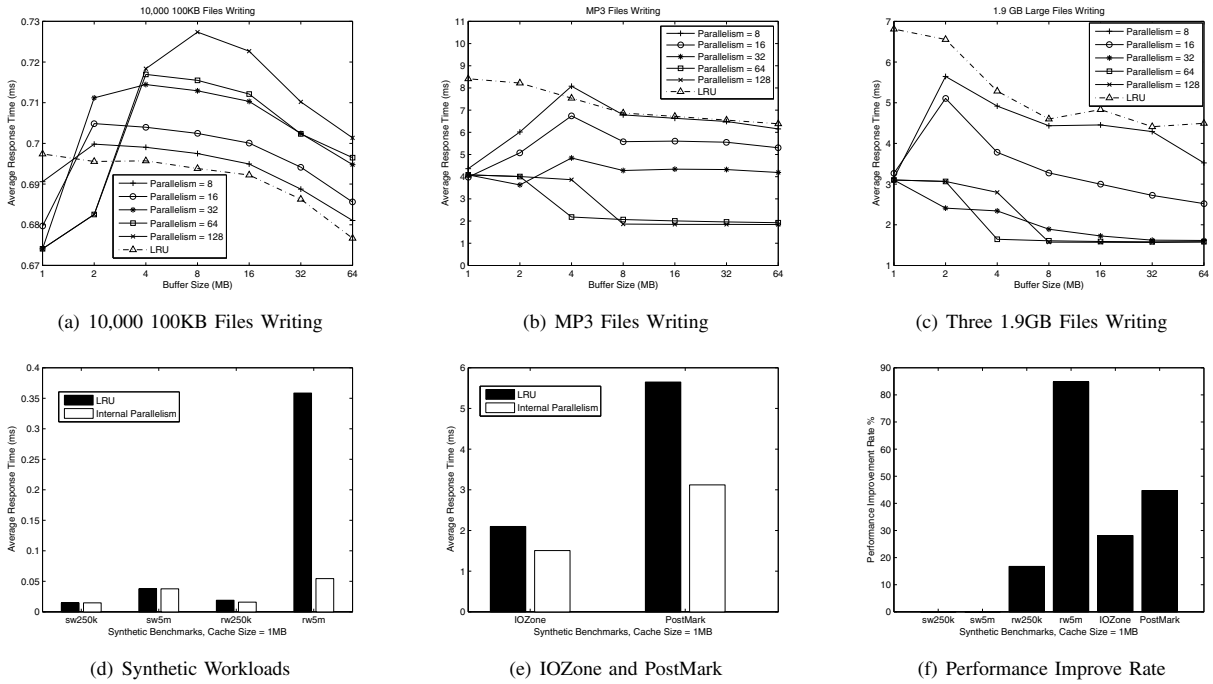
Fig. 4(b) plots the results when file sizes are large. In this case, the overhead of moving data from buffer to flash memory chips is not obvious. Fig. 4(c) shows response time of the simulated SSD when file size is very large (e.g. 1.9GB). There are only 3 files as large as 1.9GB. The overhead of moving data is not significant and the internal parallelism scheme performed best when we moved 128 pages from each list for its corresponding package.

Fig. 4(d) illustrates the results when buffer size is set to 1MB. When we use the sw250k and sw5m traces to drive the simulator, we observed similar average response time. The performance of these two traces were close, because there is not much room to leverage internal parallelism in regular sequential writes. rw250k and rw5m achieve as much as 84% performance improvement as shown by Fig. 4(f). We attribute the improvement to the fact that random writes can benefit parallelism writes. There are 8 packages in the configuration of the SSD used in this experiments. Ideally, if parallelism can be triggered in all the cases, as much as 87.5% (7/8) performance improvements can be achieved. Obviously, 84% is very close to the ideal upper bound.

Fig. 4(f) shows the performance of PostMark and IOZone benchmarks when buffer size is 1MB. The results in Fig. 4(f) confirm that average response time can be reduced by 27% for IOZone and about 45% for PostMark.

V. RELATED WORK

Research of flash-based storage has been active due to its high performance and low energy cost. Most flash-based memory studies focus on performance [14], FTL algorithms [11][16], energy consumption [30][7][8], file systems [17][25][12], wear-leveling [31][19], and controllers [23].



(a) 10,000 100KB Files Writing (b) MP3 Files Writing (c) Three 1.9GB Files Writing

(d) Synthetic Workloads (e) IOZone and PostMark (f) Performance Improve Rate

Fig. 4. Performance Evaluationn of Files Writing

Because of hardware limitations, simulations are widely used by researchers to investigate SSDs.

• **SSD Simulators:** Flash Translation Layer algorithms are built in firmware on SSDs. Only manufacturers are capable of modifying firmwares. The Parallel Data Laboratory of Carnegie Mellon University developed a validated simulator - DiskSim - to accurately simulate hard drives activities and performance [15]. DiskSim reads both binary and ASCII traces and provides detailed performance results. Based on DiskSim 4.0, Microsoft Research (MSR) has developed a SSD extension for DiskSim [2]. This extension is validated by both MSR and the University of Wisconsin - Madison. Computer System Lab of the Pennsylvania State University developed a FTL simulator based on DiskSim 3.0 to preliminarily verify FTL algorithms [16][21]. DiskSim 4.0 and the MSR SSD extension were chosen in this research, because the PSU FTL simulator does not support internal parallelism.

• **FTL Algorithms:** FTL algorithms have a far reaching impact on SSD performance and reliability. An ill-designed FTL algorithm not only reduces the SSDs performance, but also rapidly wears out SSDs storage units. Chen and Zhang proposed CAFTL - a content-aware flash translation layer enhancing the lifespan of flash memory based SSDs [11]. CAFTL removes unnecessary duplicate writes to reduce write traffic to flash memory. Gupta and Urgaonkar proposed a Demand-based Flash Translation Layer (DFTL) which selectively caches page-level address mappings [16]. A journal Remapping Algorithm JFTL was presented by Choi and Park [13]. JFTL writes all the data to a new region in a out-of-place update process by using an address mapping method [13]. Enhancing the internal parallelism buffer is FTL algorithm independent because the buffer is built below FTL and all page numbers are physical page numbers already remapped from logical page numbers.

• **Flash Based File Systems:** Josephson and Li designed the Directed File System (DFS) for virtualized flash storage. DFS performs consistently better than ext3 on both direct access and buffered access [17]. Prabhakaran and Zhou proposed Transactional flash, which exports a transactional interface to higher-level software so that the file systems on SSDs could be built much easier [25]. Chen proposed FlashLogging to exploit multiple flash drives for synchronous logging. The performance of Flashlogging is up to 5.7 times of magnetic-disk-based logging [12].

• **SSD Interleaving:** The inter-disk parallelism issue has been well explored for decades. Data striping, like RAID 0, represents the basic idea of inter-disk parallelism. However, even for hard drives, intra-disk parallelism has just started to be considered recently [28]. Since there is no mechanical movements in SSDs, there is a strong likelihood to improve internal parallelism. Park indicated that intra-SSD parallelism is possible on die-level, package-level, and plane-level. Furthermore, parallelism-aware request processing is an effective solution to enhance intra-SSD parallelism [32]. Chen and Zhang analyzed the essential roles of exploiting internal parallelism SSDs in high-speed data processing and proposed that the write performance is largely independent of access patterns [10]. Our work is concerned with SSD interleaving, which focuses on package-level interleaving. Die-level and plane-level interleaving also improves I/O performance. How-

ever, because they are in the level even below package-level, it is hard to enhance parallelism by interleaving at those two level.

• **Write Buffer:** Like hard drives, SSDs have a small amount of available cache space cache built on-board. Cache can significantly improve random read performance of hard drives. However, SSDs provide high-performance for random reads. So, many research projects have been conducted to use cache as write buffers. Kang applied Non-Volatile RAM (RAM) as write buffer for SSDs to improve overall performance [18]. Kim and Ahn demonstrated that random write performance could be greatly enhanced by a certain amount of write buffer in SSDs [20]. However, both of the works were done for non-volatile RAM whose performance is not as good as that of SDRAM.

VI. Future Work

There are a few open issues related to internal I/O parallelism in SSDs. Thus, we will address these challenging issues by conducting the following tasks.

• **Parallelism for Reads:** We explored an enhanced internal parallelism scheme for writes to reduce erasure penalties. Reads can benefit from internal parallelism. Because SSDs have very good performance on random reads, the overall performance improvement may not be as significant as that for writes.

• **Reducing System Impact:** Our results indicate that, when request size is relatively small, SSD system performance is affected by data movement between the buffer and the packages. Now, we are in process to identify good time slots to move data without compromising system performance.

• **Reliability:** Reliability is a critical issue for all flash based storage. In this study, we showed that data can be updated in buffer. Updating data in the write buffer of an SSD can reduce the number of erasures, thereby improving the reliability of the SSD. We plan to quantitatively evaluate the reliability impact of our write buffers on SSDs.

VII. Conclusion

In this paper, we presented an approach to improve write performance by enhancing internal parallelism for solid state drives. To maintain data consistency and correctness, we built a SDRAM based buffer below Flash Translation Layer (FTL). We proposed a different logic structure in write buffers. The number of lists in the buffer structure is the same as the number of flash memory chips (package). When the buffer is full, the same number of pages will be issued from all lists to their corresponding packages to enhance internal parallelism. To quantitatively evaluate system performance improvement, we collected traces from I/O intensive real-world applications: Linux Kernel Compilation and MapReduce Phoenix. File writing traces were also collected from three different file writing process. The performance evaluation demonstrated that enhancing internal parallelism can achieve a better performance in most cases compared with the existing caching algorithm Least Recently Used (LRU).

Acknowledgment

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as by Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation.

References

[1] Linux kernel 2.6.36.4, http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.36.4.tar.bz2. 2011.

[2] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[3] Jens Axboe and Alan D. Brunelle. blktrace user guide. 2007.

[4] Simona Boboila and Peter Desnoyers. Write endurance in flash drives: measurements and analysis. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.

[5] Luc Bouganim and Philippe Bonnet. uflip: Understanding flash io patterns. In *CIDR 2009, FOURTH BIENNIAL CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH*. CIDR, 2009.

[6] Don Capps and Tom McNeal. Analyzing nfs client performance with iozone.

[7] Adrian M. Caulfield and Steven Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[8] Feng Chen, Song Jiang, and Xiaodong Zhang. Smartsaver: turning flash drive into a disk energy saver for mobile computers. In *Proceedings of the 2006 international symposium on Low power electronics and design*, ISLPED '06, pages 412–417, New York, NY, USA, 2006. ACM.

[9] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 181–192, New York, NY, USA, 2009. ACM.

[10] Feng Chen, Rubao Lee, and Xiaodong Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *Proceedings of 17th International Symposium on High Performance Computer Architecture*, ISCA '11. IEEE Computer Society, 2011.

[11] Feng Chen, Tian Luo, and Xiaodong Zhang. Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies*, FAST'11, pages 77–90. USENIX Association, 2011.

[12] Shimin Chen. Flashlogging: exploiting flash devices for synchronous logging performance. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 73–86, New York, NY, USA, 2009. ACM.

[13] Hyun Jin Choi, Seung-Ho Lim, and Kyu Ho Park. Jftl: A flash translation layer based on a journal remapping for flash memory. *Trans. Storage*, 4:14:1–14:22, February 2009.

[14] Cagdas Dirik and Bruce Jacob. The performance of pc solid-state disks (ssds) as a function of bandwidth, concurrency, device architecture, and system organization. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 279–289, New York, NY, USA, 2009. ACM.

[15] Gregory R. Ganger, Gregory R. Ganger, Bruce L. Worthington, Bruce L. Worthington, Yale N. Patt, and Yale N. Patt. The disksim simulation environment - version 4.0 reference manual. Technical report, Carnegie Mellon University, 2008.

[16] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, pages 229–240, New York, NY, USA, 2009. ACM.

[17] William K. Josephson, Lars A. Bongo, David Flynn, and Kai Li. Dfs: a file system for virtualized flash storage. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 7–7, Berkeley, CA, USA, 2010. USENIX Association.

[18] Sooyong Kang, Sungmin Park, Hoyoung Jung, Hyoki Shim, and Jaehyuk Cha. Performance trade-offs in using nvram write buffer for flash memory-based storage devices. *IEEE Trans. Comput.*, 58:744–758, June 2009.

[19] Taeho Kgil, David Roberts, and Trevor Mudge. Improving nand flash based disk caches. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 327–338, Washington, DC, USA, 2008. IEEE Computer Society.

[20] Hyojun Kim and Seongjun 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, pages 16:1–16:14, Berkeley, CA, USA, 2008. USENIX Association.

[21] Youngjae Kim, B. Tauras, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *Advances in System Simulation, 2009. SIMUL '09. First International Conference on*, pages 125 –131, 2009.

[22] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 145–158, New York, NY, USA, 2009. ACM.

[23] Chanik Park, P. Talawar, Daesik Won, MyungJin Jung, JungBeen Im, Suksan Kim, and Youngjoon Choi. A high performance controller for nand flash-based solid state disk (nssd). In *Non-Volatile Semiconductor Memory Workshop, 2006. IEEE NVSMW 2006. 21st*, pages 17 –20, feb. 2006.

[24] Sungmin Park, Hoyoung Jung, Hyoki Shim, Sooyong Kang, and Jaehyuk Cha. Write buffer-aware address mapping for nand flash memory devices. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1 –2, 2008.

[25] Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Transactional flash. In *In Proc. Symposium on Operating Systems Design and Implementation (OSDI)*, 2008.

[26] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society.

[27] Mark Russinovich. Diskmon for windows v2.01, http://technet.microsoft.com/en-us/sysinternals/bb896646. 2006.

[28] Sriram Sankar, Sudhanva Gurumurthi, and Mircea R. Stan. Intra-disk parallelism: An idea whose time has come. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 303–314, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.

[30] Guangyu Sun, Yongsoo Joo, Yibo Chen, Dimin Niu, Yuan Xie, Yiran Chen, and Hai Li. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *HPCA*, pages 1–12, 2010.

[31] Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably erasing data from flash-based solid state drives. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 8–8, Berkeley, CA, USA, 2011. USENIX Association.

[32] Seon yeong Park, Euiseong Seo, Ji-Yong Shin, Seungryoul Maeng, and Joonwon Lee. Exploiting internal parallelism of flash-based ssds. *Computer Architecture Letters*, 9(1):9 –12, 2010.