Reducing SSD Read Latency via NAND Flash Program and Erase Suspension

Guanying Wu and Xubin He
Department of Electrical and Computer Engineering
Virginia Commonwealth University, Richmond, VA 23284

Abstract

In NAND flash memory, once a page program or block erase (P/E) command is issued to a NAND flash chip, the subsequent read requests have to wait until the time-consuming P/E operation to complete. Preliminary results show that the lengthy P/E operations may increase the read latency by 2x on average. As NAND flash-based SSDs enter the enterprise server storage, this increased read latency caused by the contention may significantly degrade the overall system performance. Inspired by the internal mechanism of NAND flash P/E algorithms, we propose in this paper a low-overhead P/E suspension scheme, which suspends the on-going P/E to service pending reads and resumes the suspended P/E afterwards. In our experiments, we simulate a realistic SSD model that adopts multi-chip/channel and evaluate both SLC and MLC NAND flash as storage materials of diverse performance. Our experimental results show that the proposed technique achieves a near-optimal performance gain on servicing read requests. Specifically, the read latency is reduced on average by 50.5% compared to RPS and 75.4% compared to FIFO at cost of less than 4% overhead on write requests.

1 Introduction

NAND flash-based SSDs have better random access performance over hard drives and have potential in high performance computing system market. However, NAND flash has performance and cost problems which limit its application [11]. The problem addressed in this paper is the read vs. program/erase (P/E) contention. Due to slow P/E speed of NAND flash, once P/E is committed to the flash chip, pending or subsequent read requests suffer from the prolonged service latency caused by the waiting time. As disk read requests are resulted from upper level cache misses, the compromised read latency of the disk causes degraded application performance. To reduce read latency, on-disk write buffers may avoid or postpone the write commitments to the flash [9, 6, 7]. Executing the garbage collection processes during the idle time of the drive may also alleviate the contention between read and P/E [1, 10]. Furthermore, the read requests can be prioritized in a pending list to reduce the queuing time caused by the P/E. However, none of these approaches preempt the committed P/E for read requests.

To address this read vs. P/E contention problem, we propose a *P/E Suspension* scheme for NAND flash that allows the execution of the P/E operations to be suspended so as to service the pending reads and then the suspended P/E is resumed. The internal process of the program operation is done in a "step-by-step" fashion (Incremental Step Pulse Programming, or ISPP [2]), and thus the program can be suspended at the interval of two consecutive steps, or the on-going step could be canceled and re-executed upon resumption. The erase process requires the duration of erase-voltage pulse to be satisfied, and thus the erase can also be suspended and resumed as long as we ensure the required timing.

The implementation of P/E suspension for NAND flash involves minimal modifications to the flash interface, i.e., merely the *"program suspend/resume"* and *"erase suspend/resume"* commands need to be added in the command set of the flash interface [12]. To support P/E suspension, the *control logic inside the flash chip* is required to determine the appropriate time to suspend the P/E (suspension point) and to maintain or retrieve the previous state of the suspended P/E so as to resume it. Noting that the implementation feasibility of the proposed schemes is based on the fundamental/typical circuitry of flash memories [3].

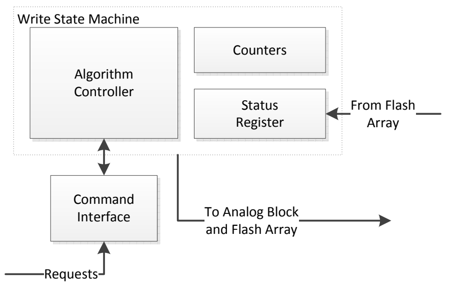
This paper makes the following contributions. First, we analyze the impact of the long P/E latency on read performance, showing that even with the read prioritization scheduling, the read latency is still severely compromised. Second, by exploiting the internal mechanism of the P/E algorithms in NAND flash memory, we propose a low-overhead P/E suspension scheme which suspends the on-going P/E operations for servicing the pending read requests. In particular, two strategies for suspending the program operation, *Inter Phase Suspension (IPS)* and *Intra Phase Cancelation(IPC)* are proposed. Third, based on simulation experiments under various workloads, we demonstrate that compared to FIFO, the proposed design can significantly reduce the SSD read latency for both SLC and MLC NAND flash.

The rest of this paper is organized as follows: In Section 2 we give an overview of the internal mechanism for P/E on NAND flash and briefly discuss related work. In Section 3, we conduct simulations to show how the read latency is increased by chip contention. We describe our detailed P/E suspension scheme in Section 4 and evaluate our approach via simulation experiments in Section 5. Finally we conclude our paper in Section 6.

2 Background and Related Work

2.1 NAND Flash Program/Erase Algorithm

Incremental Step Pulse Programming (ISPP) is typically used for precisely programming or erasing the NAND flash [3]. It is made of a series of program and verify iterations. The execution of ISPP and the erase process is implemented in the flash chip with an analog block and a control logic block. The analog block is responsible for regulating and pumping the voltage for program or erase operations. The control logic block is responsible for interpreting the interface commands, generating the control signals for the flash cell array and the analog block, and executing the program and erase algorithms. As shown in the following diagram [3], the write state machine consists of three components: an *algorithm controller* to execute the algorithms for the two types of operations, several *counters* to keep track of the number of ISPP iterations, and a *status register* to record the results from the verify operation.



2.2 Related Work

The idea of preempting low priority operations for high priority ones via breaking down an operation to small phases has been embodied in [4], [13], etc. Dimitrijevic et al. proposed *Semi-preemptible IO* [4] to divide HDD I/O requests to small disk commands to enable preemption for high priority requests. Similar to NAND flash, *Phase Change Memory* (PCM) has much larger write latency than read latency. Qureshi *et al.* proposed in [13] a few techniques to preempt the on-going writes of PCM for reads: *write cancelation* and a threshold-based overhead control method to reduce the overhead are proposed to cancel entire write operations; PCM, like NAND flash, adopts the *iterative-write* algorithm. Our work differs from [13] as follows: PCM has the in-place update capability, while NAND flash requires erase before program. In our work, the suspension of erase operation is proposed. *Write Cancelation* for the entire write process of NAND flash is not viable. NAND flash's iterative write process differs from PCM in that, each iteration has two phases (program and verify). Thus, for each iteration, we may have two suspension points. Furthermore, we propose the *shadow buffer* to overcome the overhead of re-transferring the write data upon resumption, which is not discussed in [13].

3 Motivation

In this section, we demonstrate how the read vs. P/E contention increases the read latency under various workloads. We have modified *MS-add-on* simulator [1] based on Disksim 4.0. Specifically, under the workloads of a variety of popular disk traces, we compare the read latency of two scheduling policies, FIFO and read priority scheduling (RPS), to show the limitation of RPS. Furthermore, with RPS, we set the latency of program and erase operation to be equal to that of read and *zero* to justify the impact of P/E on the read latency.

3.1 Configurations and Workloads

The simulated SSD is configured as follows: there are 16 flash chips, each of which owns a dedicated channel to the flash controller. Each chip has four planes that are organized in a RAID-0 fashion; the size of one plane is 512 MB or 1 GB assuming the flash is used as SLC or 2-bit MLC, respectively (the page size is 2 KB for SLC or 4 KB for MLC). To maximize the concurrency, each individual plane has its own allocation pool [1]. The garbage collection processes are executed in the background so as to minimize the interference with the foreground requests. In addition, the percentage of flash space over-provisioning is set as 30%, which doubles the value suggested in [1]. Considering the limited working-set size of the workloads used in this paper (described in next subsection), 30% over-provisioning is believed to be sufficient to avoid frequent execution of garbage collection processes. The write buffer size is 64 MB. The SSD is connected to the host via a PCI-E of 2.0 GB/s. The physical operating parameters of the flash memory is summarized in Table 1.

We choose 6 disk I/O traces for our experiments: *Financial 1 and 2* (F1, F2) [14]; *Display Ads Platform and payload servers* (DAP) and *MSN storage metadata* (MSN) traces [8]; *Cello99* [5] traces (C3, C8). Noting that those traces were originally collected on HDDs, to produce more stressful workloads for SSDs, we compress all these traces so that the system idle time is reduced from 98% to around 70% for each workload.

Symbols	Description	Value	
		SLC	MLC
T_{bus}	The bus latency of transferring one page from/to the chip	20 μs	40 μs
T_{r_phy}	The latency of sensing/reading data from the flash	10 μs	25 μs
T_{w_total}	The total latency of ISPP in flash page program	140 μs	660 μs
N_{w_cycle}	The number of ISPP iterations	5	15
T_{w_cycle}	The time of one ISPP iteration (T_{w_total}/N_{w_cycle})	28 μs	44 μs
$T_{w\_program}$	The duration of program phase of one ISPP iteration	20 μs	20 μs
T_{verify}	The duration of the verify phase	8 μs	24 μs
T_{erase}	The duration of erase pulse	1.5 ms	3.3 ms
$T_{voltage\_reset}$	The time to reset operating voltages of on-going operations	4 μs	
T_{buffer}	The time taken to load the page buffer with data	3 μs	

Table 1: Flash Parameters

Trace	SLC		MLC	
	Read	Write	Read	Write
F1	0.37	0.87	0.44	1.58
F2	0.24	0.57	0.27	1.03
DAP	1.92	6.85	5.74	11.74
MSN	4.13	4.58	8.47	25.21
C3	0.25	2.85	0.52	6.30
C8	0.44	2.33	0.56	4.54

Table 2: Numerical Latency Values of FIFO (in *ms*)

3.2 Experimental Results

In this subsection, we compare the read latency performance under four scenarios: FIFO; RPS; PER (the latency of program and erase is set equal to that of read); and PE0 (the latency of program and erase is set to zero). Note that both PER and PE0 are applied upon RPS in order to study the chip contention and the limitation of RPS. Due to the large range of the numerical values of the experimental results, we normalize them to the corresponding results of FIFO, which are listed in Table 2 for reference. The normalized results are plotted in Fig. 1, where the left part shows the results of SLC and the right part is for MLC. Compared to FIFO, RPS achieves impressive performance gain, e.g., the gain maximizes at an effective read latency ("effective" refers to the actual latency taking the queuing delay into account) reduction of 44.6% (SLC) and 48.3% (MLC) on average. However, if the latency of P/E is the same as read latency or zero, i.e., in the case of *PER* and *PE0*, the effective read latency can be further reduced. For example, with PE0, the read latency reduction is 71.7% (SLC) and 75.6% (MLC) on average. Thus, even with RPS policy, the chip contention still increases the read latency by about 2x on average.

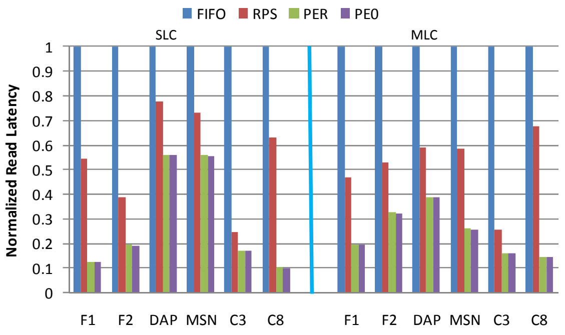


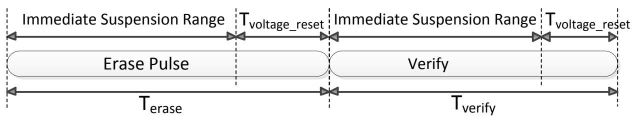
Figure 1: Read Latency Performance Comparison: FIFO, RPS, PER, and PE0. Results normalized to FIFO.

4 Design of P/E Suspension Scheme

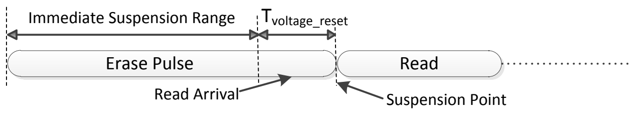
4.1 Erase Suspension and Resumption

In NAND flash, the erase process consists of two phases: first, an erase pulse lasting for T_{erase} is applied on the target block; second, a verify operation that takes T_{verify} is performed to check if the preceding erase pulse has successfully erased all bits in the block. Otherwise, the above process is repeated until success, or if the number of iterations reaches the predefined limit, an operation failure is reported. Typically, for NAND flash, since the *over-erasure* is not a concern [3], the erase operation can be done with a single erase pulse.

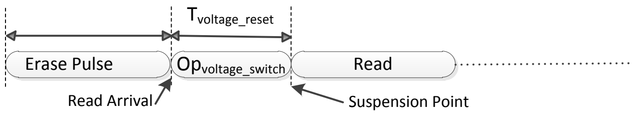
How to suspend an erase operation: suspending either the erase pulse or verify operation requires resetting the status of the corresponding wires that connect the flash cells with the analog block. Specifically, due to the fact that the flash memory works at different voltage bias for different operations, the current voltage bias applied on the wires (and thus on the cell) needs to be reset for the pending read request. This process ($Op_{voltage\_reset}$ for short) takes a period of $T_{voltage\_reset}$. Noting that either the erase pulse or verify operation always has to conduct $Op_{voltage\_reset}$ at the end (as shown in the following diagram of erase operation timeline).



Thus, if the suspension command arrives during $Op_{voltage\_reset}$, the suspension will succeed once $Op_{voltage\_reset}$ is finished (as illustrated in the following diagram of erase suspension timeline).



Otherwise, an $Op_{voltage\_reset}$ is executed immediately and then the read request is serviced by the chip (as illustrated in the following diagram).

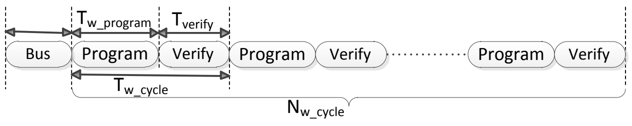


How to resume an erase operation: the resumption means the control logic of NAND flash resumes the suspended erase operation. Therefore, the control logic should keep track of the progress, i.e., whether the suspension happens during the verify phase or the erase pulse. For the first scenario, the verify operation has to be re-done all over again. For the second scenario, the erase pulse time left (T_{erase} minus the progress), for example, 1 ms will be done in the resumption if no more suspension happens. Actually, the task of progress tracking can be easily supported by the existing facilities in the control logic of NAND flash: the pulse width generator is implemented using a counter-like logic [3], which keeps track of the progress of the current pulse.

The overhead on the effective erase latency: resuming the erase pulse requires extra time to set the wires to the corresponding voltage bias, which takes approximately the same amount of time as $T_{voltage\_reset}$. Suspending during the verify phase causes a re-do in the resumption, and thus the overhead is the time of the suspended/cancelled verify operation. In addition, the read service time is included in the effective erase latency.

4.2 Program Suspension and Resumption

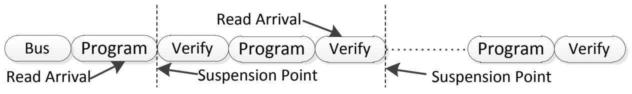
The process of servicing a program request is: first, the data to be written is transferred through the controller-chip bus and loaded in the page buffer; then the ISPP is executed, in which a total number of N_{w_cycle} iterations consisting of a *program* phase followed by a *verify* phase are conducted on the target flash page. In each ISPP iteration, the program phase is responsible for applying the required program voltage bias on the cells so as to charge them. In the verify phase, the content of the cells is read to verify if the desired amount of charge is stored in each cell: if so, the cell is considered *program-completion*; otherwise, one more ISPP iteration will be conducted on the cell. Due to the fact that all cells in the target flash page are programmed simultaneously, the overall time taken to program the page is actually determined by the cell that needs the most number of ISPP iterations. A major factor that determines the number of ISPP iterations needed is the amount of charge to be stored in the cell, which is in turn determined by the data to be written. For example, for the 2-bit MLC flash, programming a “0” in a cell needs the most number of ISPP iterations, while for “3” (the erased state), no ISPP iteration is needed. Since all flash cells in the page are programmed simultaneously, N_{w_cycle} is determined by the smallest data (2-bit) to be written; nonetheless, we make a rational assumption in our simulation experiments that N_{w_cycle} is constant and equal to the maximum value. The program process is illustrated in the following diagram.



How to retain the page buffer content: before we move on to suspension, this critical problem has to be solved. For program, the page buffer contains the data to be written. For read, it contains the retrieved data to be transferred to the flash controller. If a write is preempted by a read, the content of the page buffer is certainly replaced. Thus, the resumption of the write demands the page buffer re-stored. Intuitively, the flash controller that is responsible for issuing the suspension and resumption commands may keep a copy of the write page data until the program is finished and upon resumption, the controller re-sends the data to the chip through the controller-chip bus. However, the page transfer consumes a significant amount of time: unlike the NOR flash which does *byte programming*, NAND flash does *page programming*, and the page size is of a few kilobytes. For instance, assuming a 100 MHz bus and 4 KB page size, the bus time T_{bus} is about 40 μs.

To overcome this overhead, we propose a *Shadow Buffer* in the flash. The shadow buffer serves like a replica of the page buffer and it automatically loads itself with the content of the page buffer upon the arrival of the write request and re-stores the page buffer while resumption. The load and store operation takes the time T_{buffer}. The shadow buffer has parallel connection with the page buffer, and thus the data transfer between them can be done on the fly. T_{buffer} is normally smaller than T_{bus} by one order of magnitude.

How to suspend a program operation: compared to the long width of the erase pulse (T_{erase}), the program and verify phase of the program process is normally two orders of magnitude shorter. Intuitively, the program process can be suspended at the end of the program phase of any ISPP iteration as well as the end of the verify phase. We refer to this strategy as “Inter Phase Suspension” (**IPS**). IPS has in total $N_{w\_cycle} * 2$ potential suspension points as illustrated in the following diagram.



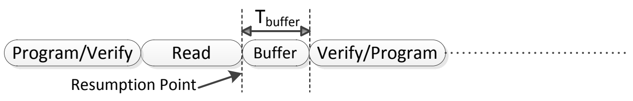
Due to the fact that at the end of the program or verify phase, the status of the wires has already reset ($Op_{voltage\_reset}$), IPS does not introduce any extra overhead, except for the service time of the read or reads that preempt the program. However, the effective read latency should include the time from the arrival of read to the end of the corresponding phase. For simplicity, assuming the arrival time of reads follows the uniform distribution, the probability of encountering the program phase and the verify phase is $T_{w\_program}/(T_{verify}+T_{w\_program})$ and $T_{verify}/(T_{verify}+T_{w\_program})$, respectively. Thus, the average extra latency for the read can be calculated as:

$$T_{read\_extra} = \frac{T_{w\_program}}{(T_{verify}+T_{w\_program})} * \frac{T_{w\_program}}{2} + \frac{T_{verify}}{(T_{verify}+T_{w\_program})} * \frac{T_{verify}}{2} \quad (1)$$

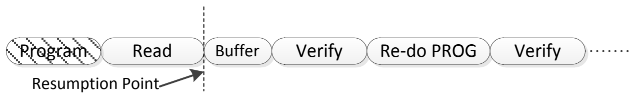
Substituting the numerical values in Table 1, we get 8.29 μs (SLC) and 11.09 μs (MLC) for T_{read_extra}, which is comparable to the physical access time of the read (T_{r_phy}). To further improve the effective read latency, we propose "Intra Phase Cancelation" (**IPC**). Similar to canceling the verify phase for the erase suspension, IPC cancels an on-going program or verify phase upon suspension. The reason of canceling instead of pausing the program phase is that the duration of the program phase, $T_{w\_program}$, is short and normally considered atomic (cancelable but not pause-able).

Again, for IPC, if the read arrives when the program or verify phase is conducting $Op_{voltage\_reset}$, the suspension happens actually at the end of the phase, which is the same as IPS; otherwise, $Op_{voltage\_reset}$ is started immediately and the read is then serviced. Thus, IPC achieves a T_{read_extra} no larger than $T_{voltage\_reset}$.

How to resume from IPS: first of all, the page buffer is re-loaded with the content of the shadow buffer. Then, the control logic examines the last ISPP iteration number and the previous phase. If IPS happens at the end of the verify phase, which implies that the information of the status of cells has already been obtained, we may continue with the next ISPP if needed; on the other hand, if the last phase is the program phase, naturally we need to finish the verify operation before moving on to the next ISPP iteration. The resumption process is illustrated in the following diagram.



How to resume from IPC: compared to IPS, the resumption from IPC is more complex. Different from the verify operation, which does not change the charge status of the cell, the program operation puts charge in the cell and thus changes the threshold voltage (V_{th}) of the cell. Therefore, we need to determine whether the canceled program phase has already achieved the desired V_{th} (i.e., whether the data could be considered written in the cell), by a verify operation. If so, no more ISPP iteration is needed on this cell; otherwise, the previous program operation is executed on the cell again. The later case is illustrated in the following diagram.



Re-doing the program operation would have some affect on the tightness of V_{th}, but with the aid of ECC and a fine-grained ISPP, i.e., small incremental voltage ΔV_{pp}, the IPC has little impact on the data reliability of the NAND flash. The relationship between ΔV_{pp} and the tightness of V_{th} is modeled in [15].

The overhead on the effective write latency: IPS requires re-loading the page buffer, which takes T_{buffer}. For IPC, if the verify phase is canceled, the overhead is the time elapsed of the canceled verify phase plus the read service time and T_{buffer}. In case of program phase, there are two scenarios: if the verify operation reports that the desired V_{th} is achieved, the overhead is the read service time plus T_{buffer}; otherwise, the overhead is the time elapsed of the canceled program phase plus an extra verify phase, in addition to the overhead of the above scenario. Clearly, IPS achieves smaller overhead on the write than IPC but relatively lower read performance.

5 Performance Evaluation

In this section, we evaluate our proposed design under different workloads described in Section 3.1.

5.1 Read Performance Gain

First, we compare the average read latency of P/E suspension with RPS, PER and PE0 in Fig. 2, where the results are normalized to that of RPS. For P/E suspension, the IPC (Intra Phase Cancelation), denoted as "PES_IPC", is adopted in Fig. 2. PE0, with which the physical latency values of program and erase are set to zero, serves as an optimistic situation where the contention between reads and P/E's is completely eliminated. Fig. 2 demonstrates that, compared to RPS, the proposed P/E suspension achieves a significant read performance gain, which is almost equivalent to the optimal case, PE0 (with less than 1% difference). Specifically, on the average of the 6 traces, PES_IPC reduces the read latency by 48.9% for SLC and 50.5% for MLC compared to RPS, and 71.6% for SLC and 75.4% for MLC compared to FIFO. For conciseness, the results of SLC and (then) MLC are listed without explicit specification in the following text.

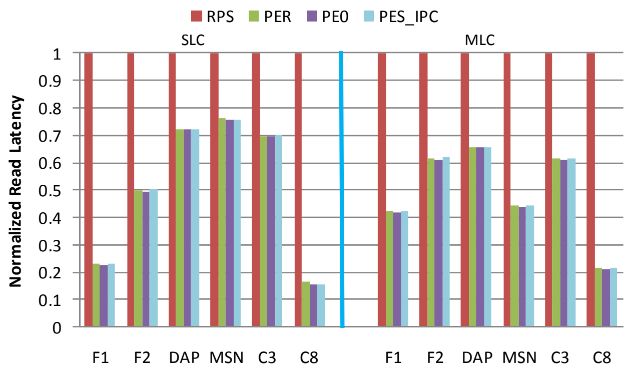


Figure 2: Read Latency Performance Comparison: RPS, PER, PE0, and PES_IPC (P/E Suspension using IPC). Normalized to RPS.

As stated in Section 4, IPC can achieve better read performance but cause higher write overhead compared to IPS. We compare the read performance of IPC and IPS in Fig. 3. The read latency of IPS is 8.0% and 2.7% on average and at-most 13.2% and 6.7% (under F1) higher than that of IPC. The difference is resulted from the fact that IPS has extra read latency, which is mostly the time between read request arrivals and the suspension points at the end of the program or verify phase. We notice that the latency performance of IPS using SLC is poorer than MLC under all traces, which is because of the higher sensitivity of SLC's read latency to the overhead caused by the extra latency.

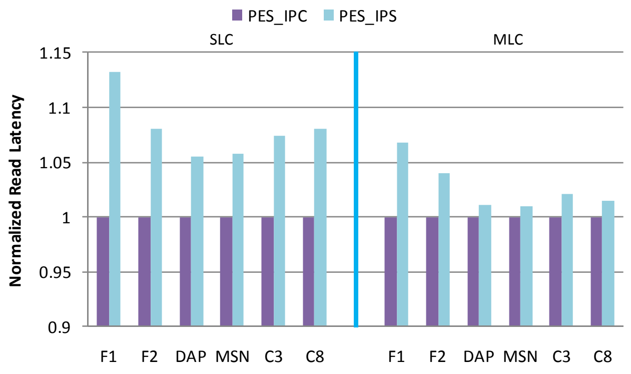


Figure 3: Read Latency Performance Comparison: PES_IPC vs. PES_IPS. Normalized to PES_IPC.

5.2 Write Overhead

Since both RPS and P/E suspension introduce minimal extra chip bandwidth usage, the write throughput is barely compromised. We use the latency as a metric for the overhead evaluation. First, we compare the average write latency of FIFO, RPS, PES_IPS, and PES_IPC in Fig. 4. Obviously, the write overhead in terms of latency is trivial compared to the read performance gain we achieve with P/E suspension. Specifically, RPS increases the write latency by 2.3% and 1.2% on average and at-most 6.7% (SLC, MSN) and 3.8% (MLC, DAP), compared to FIFO. PES_IPC increases write latency by 3.6% and 1.9% on average and at-most 6.9% (SLC, MSN) and 4.3%(MLC, DAP), respectively. PES_IPC increases the write latency by 3.6% and 2.0% on average and at-most 6.9% (SLC, MSN) and 4.3%(MLC, DAP).

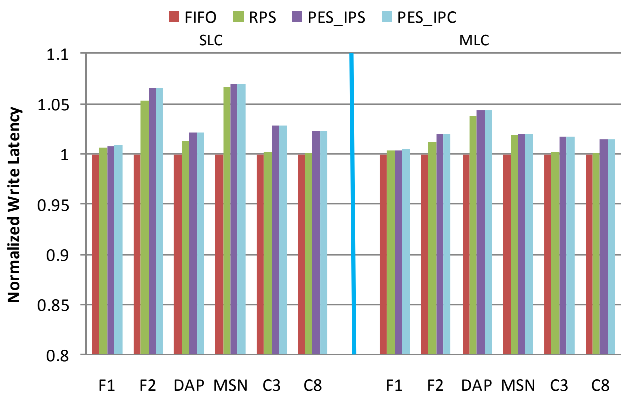


Figure 4: Write Latency Performance Comparison: FIFO, RPS, PES_IPC, PES_IPS. Normalized to FIFO.

Two major factors determine the write latency overhead: increased latency of each suspended P/E operation; the percentage of P/E that are suspended. We compare the original P/E latency reported by the device with latency after suspension in Fig. 5. The average overhead of suspended P/E is about 10.2% (SLC) and 7.8% (MLC). The percentage of suspended P/E is presented in Fig. 6. There is 4.9% (SLC) and 7.4% (MLC) of P/E's that are suspended on average. These two sets of results explain the low write overhead our design achieves.

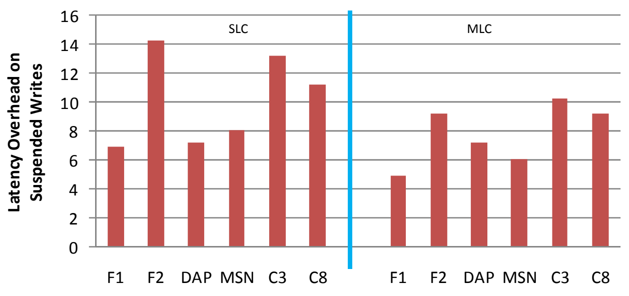


Figure 5: Compare the original write latency with the effective write latency resulted from P/E Suspension. Y axis represents the percentage of increased latency caused by P/E suspension.

5.3 Sensitivity Study on Write Queue Size

Finally, we study the sensitivity of write overhead to the write queue size. In order to obtain an amplified write overhead, we select F2, which has the highest percentage of read requests, and compress the simulation time

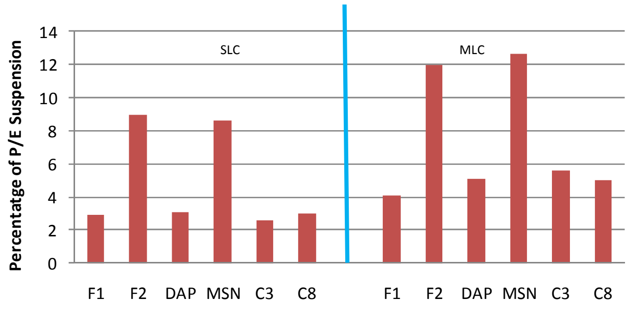


Figure 6: Percentage of suspended writes.

of F2 by 7 times to intensify the workload. In Figure 7 we present the write latency results of RPS and PES_IPC (normalized to that of FIFO) by varying the maximum write queue size from 16 to 512. Clearly, the write overhead of both RPS and PES_IPC is sensitive to the maximum write queue size, which suggests that the flash controller should limit the write queue size to control the write overhead. Noting that, relative to RPS, the PES_IPC has a near-constant increase on the write latency, which implies that the major contributor of overhead is RPS when the queue size varies.

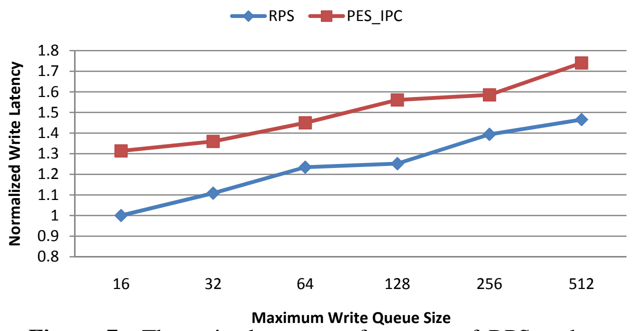


Figure 7: The write latency performance of RPS and PES_IPC while the maximum write queue size varies. Normalized to FIFO.

6 Conclusion and Future Work

One performance problem of NAND flash is that its program and erase latency is much higher than the read latency. This problem causes the chip contention between reads and P/Es due to the fact that with current NAND flash interface, the on-going P/E cannot be suspended and resumed. To alleviate the impact of the chip contention on the read performance, in this paper we propose a light-overhead *P/E suspension* scheme by exploiting the internal mechanism of P/E algorithm in NAND flash. The design is simulated/evaluated with precise timing and realistic SSD modeling of multi-chip/channel. Experimental results show that the proposed P/E suspension significantly reduces the read latency with trivial overhead on write performance.

Our future work will apply the idea of P/E suspension to further improve the performance of foreground processes via suspending the background operations (e.g., the garbage collection operations) in SSDs.

Acknowledgments

We thank Tong Zhang for the preliminary discussion on some physical parameters of NAND flash. We also thank our shepherd Dushyanth Narayanan and anonymous reviewers for their feedback. This research is sponsored in part by the U.S. National Science Foundation (NSF) under grants CCF-1102605 and CCF-1102624.

References

[1] AGRAWAL, N., PRABHAKARAN, V., AND ET AL. Design Tradeoffs for SSD Performance. In *USENIX ATC* (Boston, Massachusetts, USA, 2008).

[2] ARASE, K. Semiconductor NAND Type Flash Memory with Incremental Step Pulse Programming, Sept. 22 1998. U.S. Patent 5,812,457.

[3] BREWER, J., AND GILL, M. Nonvolatile Memory Technologies with Emphasis on Flash. *IEEE Whiley-Interscience, Berlin* (2007).

[4] DIMITRIJEVIC, Z., RANGASWAMI, R., AND CHANG, E. Design and Implementation of Semi-preemptible IO. In *FAST* (2003), USENIX, pp. 145–158.

[5] HEWLETT-PACKARD LABORATORIES. Cello99 Traces. `http://tesla.hpl.hp.com/opensource/`.

[6] JO, H., KANG, J.-U., AND ET AL. FAB: Flash-Aware Buffer Management Policy for Portable Media Players. *IEEE Transactions on Consumer Electronics 52*, 2 (2006), 485–493.

[7] KANG, S., AND ET AL. Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices. *IEEE Transactions on Computers 58*, 6 (2009), 744–758.

[8] KAVALANEKAR, S., AND ET AL. Characterization of Storage Workload Traces from Production Windows Servers. In *IISWC* (2008).

[9] KIM, H., AND AHN, S. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage Abstract. In *FAST* (2008).

[10] KIM, Y., ORAL, S., SHIPMAN, G., LEE, J., DILLOW, D., AND WANG, F. Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-State Drives. In *MSST* (2011), IEEE, pp. 1–12.

[11] NARAYANAN, D., THERESKA, E., DONNELLY, A., ELNIKETY, S., AND ROWSTRON, A. Migrating server storage to SSDs: Analysis of tradeoffs. In *EuroSys* (2009).

[12] ONFI WORKING GROUP. The Open NAND Flash Interface, 2011. `http://onfi.org/`.

[13] QURESHI, M., AND ET AL. Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. In *HPCA* (2010), IEEE, pp. 1–11.

[14] STORAGE PERFORMANCE COUNCIL. SPC Trace File Format Specification. `http://traces.cs.umass.edu/index.php/Storage/Storage`.

[15] WU, G., HE, X., XIE, N., AND ZHANG, T. DiffECC: Improving SSD Read Performance Using Differentiated Error Correction Coding Schemes. *MASCOTS* (2010), 57–66.