# A Read Latency Variation Aware Independent Read Scheme for QLC SSDs

Dong Huang, Dan Feng*, Qiankun Liu, Bo Ding, Wei Zhao, Xueliang Wei, Wei Tong*
*Key Laboratory of Information Storage System, Engineering Research Center for Data System and Technology, Ministry of Education*
*Wuhan National Laborotory of Optoelectronics, Huazhong University of Science and Technology*
*haltz,dfeng,lqk,boding,weiz,xueliang_wei,tongwei@hust.edu.cn, *Corresponding Author*

***Abstract*—QLC flash-based SSDs has attracted growing interest and is expected to fit in read-intensive scenarios owing to its higher cost-effectiveness and shorter write endurance compared with the Triple-Level Cell (TLC) SSDs. Recently, new commands supporting independent reads such as Single Operation Multiple Locations (SOML) and Independent multi-plane (IMP) read are proposed to improve read performance. Unfortuantely, while independent read exhibits a significant performance improvement, we show in this paper that the exisiting approach fails to fully exploit its potential due to the larger read latency variation and more planes per die for current SSD architecture. Through a set of experiments, we demonstrate that the lack of a read latency variation aware machanism leads to low performance of independent read among a wide variety of workloads. To alleviate this issue, we propose LITA, which key idea is to combine read transactions with similar latency into one command. LITA includes (1) LIT, a latency variation aware transaction combination, (2) and TASAP, a latency variation aware transaction completion service. The experimental results show LITA can reduce read latency by 20.4% and 9.6% on average for 4-planes QLC SSDs under IMP and SOML, respectively.**

***Index Terms*—QLC, Flash-based SSDs, Independent Read, Latency Variation**

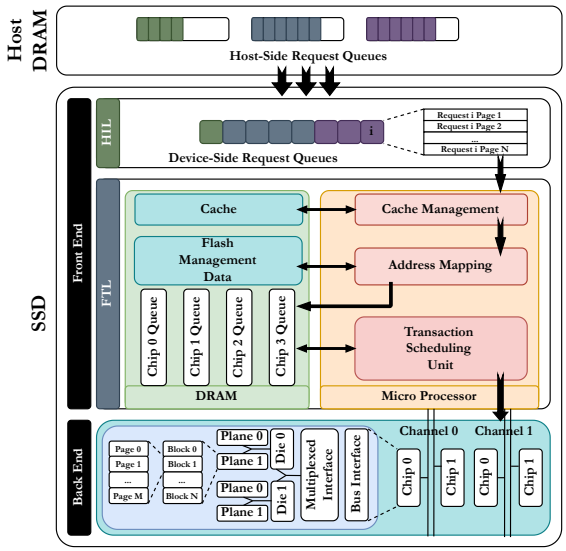



Fig. 1. ***SSD Architecture Overview.***

## I. Introduction

Quad-Level Cell (QLC) SSDs offer a cost-effective alternative to conventional Triple-Level Cell (TLC) SSDs, with a greater storage capacity and approximately 1/3 higher density [1]. Nonetheless, the aforementioned economic benefit is accompanied by reduced read/program efficiency attributed to extended operation latency. In order to enhance the efficiency of read operations, two novel types of read commands have been proposed. Single Operation Multiple Locations (SOML) [2] is a command that enables the controller to access multiple partial pages on a single plane with different page addresses. Another command, known as Independent Multi-Plane (IMP) read [3], has been developed to alleviate the limitations of traditional multi-plane read command [4]. Conventional multi-plane read command needs the target addresses to have equal block and page offset. In contrast, IMP read permits the target address for each plane to possess different block and page offsets, thereby relaxing the constraints of conventional multi-plane read. To unify the name, we call them Independent Read. Independent read can enhance performance by executing multiple read transactions concurrently instead of executing them sequentially [2].

Unfortunately, we discover that the present architecture of QLC SSDs is inadequate in harnessing the complete advantages of independent read. First, the utilization of gray-code programming in QLC flash cells is associated with a higher degree of read latency variation in contrast to TLC [5], [6]. In an independent read command, the short-latency reads must wait for the long-latency reads, which results in idle time and decreases independent read performance. Second, NAND flash manufacturers have augmented the quantity of planes incorporated in a single die to increase density [7]–[9], further intensifying the effects of latency variation. We analyze the impact of augmented number of planes and increased latency variation on the independent read efficiency therotically and experimentally. The results suggest that the current architecture of QLC SSDs presents a challenge in effectively utilizing independent read.

In this paper, we propose a lightweight approach called LITA to address this challenge. LITA comprises of two techniques called **L**east **I**dle **T**ime (LIT) and **T**ransfer **A**s **S**oon **A**s **P**ossible (TASAP). LIT enables the formation of independent read commands while taking into account variations in read latency. On the other hand, TASAP transfers read data immediately

979-8-3503-4859-0/DATE24/© 2024 EDAA

after a read transaction is completed, thereby minimizing the wait time associated with read latency variations. In conlusion, we mainly make the following contributions:

- We observe that with 4-planes QLC flash chips, the independent read performance degradation due to latency variation cannot be ignored. We theroitically and experimentally evaluate the effects of the present QLC SSD architecture, which motivates us to explore a potential approach to address this challenge.
- We propose LITA as the combination of LIT and TASAP, which forms independent read commands through combining transactions with similar access latency and transfers read data to finalize the transaction. LITA is a firmware-level approach and requires minimal hardware overhead.
- We implement LITA in a state-of-the-art simulator MQSim [10] and rigorously evaluate the implementation. The experimental results demonstrate that LITA has the potential to significantly decrease average read latency.

## II. Background

### A. SSD Architecture

We offer a brief overview of the internals of an SSD. An SSD typically comprises of an SSD controller and an array of flash chips, as shown in Figure 1. PCIe-based NVMe protocol connects the SSD to the host system. The SSD's front end contains the control and management units, and the back end contains the memory chips.

***Back End.*** The backend is composed of organized hierarchically flash channels and chips to maximize I/O concurrency [11]. The back end contains multiple channels for front-to-back data transfer. Each channel is connected to multiple flash chips and each flash chip is divided into multiple dies. Each die can independently execute memory commands. The number of planes that comprise a die is determined by the NAND flash manufacturer. Each plane consists of numerous blocks, and each block consists of numerous pages, which is the smallest read and program unit.

***Front End.*** The front end comprises mainly of two components: host interface logic (HIL) and flash translation layer (FTL). The HIL handles receiving/responding I/O requests from the host system. The FTL runs on a microprocessor in the SSD, processing I/O requests and flash management operations.

### B. Conventional Multi-Plane Read and Independent Read

Conventional multiplane read can execute multiple read transactions within a die if these transactions have the same type, block and page offsets [11]. Despite the significant performance advantage, it is difficult to use multi-plane for random reads because random read transactions usually have different page offsets [2]. To address this challenge, independent read is proposed.

***Single Operation Multiple Locations (SOML).*** Liu et al. introduce SOML, leveraging intra-chip parallelism through the simultaneous reading of multiple partial pages from different blocks in a single command [2]. This command comprises smaller sub-commands, each composed of an 8-bit start code, a 24-bit address, and an 8-bit end code. Notably, to indicate the conclusion of the SOML operation, the start code for the final read sub-command differs from the start codes of the preceding sub-commands.

***Independent Multi-Plane Read (IMP).*** Independent Multi-Plane Read (IMP) is introduced as specified in the Open-NAND-Flash-Interface (ONFI) specification [3], [4]. IMP read allows distinct block and page offsets in a multi-plane command sequence, employing multiple read commands, each with a start code, address, and end code. Notably, IMP employs a unique last end code to indicate the conclusion of the command sequence.

***SOML vs. IMP***: (1) IMP reads multiple pages for each die at 16KiB, while SOML reads multiple partial-pages for each plane at 2KiB-8KiB. (2) SOML requires hardware modifications and prohibits concurrently accessed partial pages from using the same block decoder, while IMP does not.

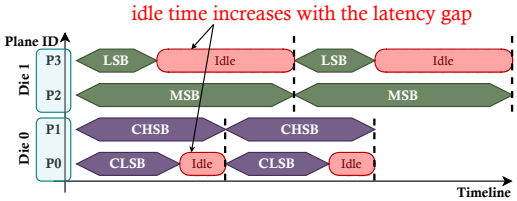
### C. Read Latency Variation and The Existing Approach

The widespread use of unbalanced gray code introduces read latency variations across different page types in QLC flash memory [6]. In QLC cells, comprising four page types (LSB, CLSB, CMSB, and MSB), the unbalanced gray coding scheme requires varying reference voltages for sensing to read these pages [5]. As a result, accessing an MSB page incurs significantly higher read latency compared to an LSB page.

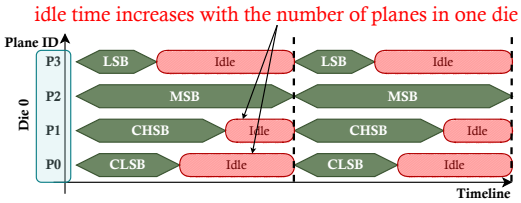
The existing approach to use independent read does not consider the read latency variation. Liu et al. [2] proposed an approach that involves iterating over the chip transaction queue to combine compliant read transactions into a SOML command. We also use it for IMP. Liu's approach includes allocating a read transaction to a plane that has not yet been assigned one, until there are no remaining transactions or idle planes. Through iterating the chip queue, an independent read command comprising of multiple transactions is formed and sent to the corresponding chip. The controller transfers read data until all transactions of the command are completed. This approach does not particularly consider latency variation since it uses a 2-planes TLC SSD, in which the read latency variation imposes a limited impact as revealed in the following section.

## III. Inefficiency of Independent Read

***Problem Analysis.*** An independent read command sequence comprises multiple page read transactions with various execution latencies. Consequently, the short-latency transaction is required to wait until the long-latency transaction is completed, leading to idle time. For simplicity, we use IMP for analysis. However, it can be easily extended to the case within the context of SOML since their command sequences are similar as mentioned in Section II. Assuming $r_i$ represents the $i_{th}$ read transaction targeting plane 0 in the chip queue. Let $IR_j$ denote the independent read command that executes $r_j$, $T_{exec}(IR_j)$ denote the execution latency of $IR_j$. The turnaround latency of $r_i$ can be calculated as $T_{exec} + T_{wait} =$




(a) *Larger latency variation increases more idle time.*




(b) *More planes increases average read command execution time.*

Fig. 2. ***Read Inefficiency for QLC 4-planes SSD.***

$T_{exec}(IR_i) + \sum_{j=0}^{i-1} T_{exec}(IR_j)$. This equation can be further extended as follows.

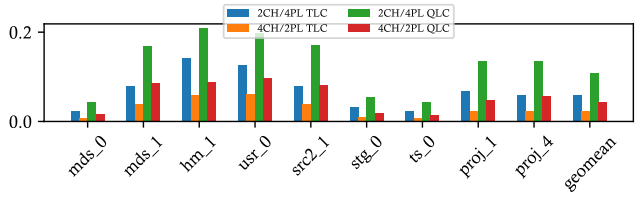
$$T_{turnaround}(r_i) = \sum_{j=0}^{i} T_{exec}(IR_j) \quad (1)$$

The execution latency $T_{exec}(IR_j)$ can be calculated as $T_{exec}(r_j) + T_{idle}(r_j)$, where $T_{idle}(r_j)$ represents the length of the interval during which $r_j$ has finished but $IR_j$ has not yet finished (i.e., $T_{idle}(r_j) = T_{exec}(IR_j) - T_{exec}(r_j)$). Using this information, we can derive the following extended equation. Since $T_{exec}(r_j)$ is a fixed value that depends on the page type, the turnaround latency of $r_i$ is determined entirely by the sum of the $T_{idle}(r_j)$ values.

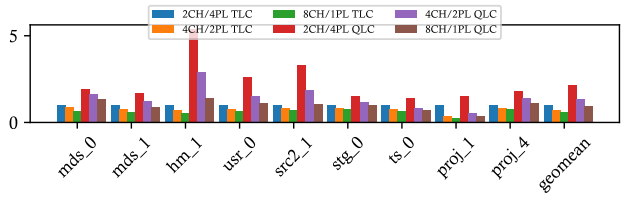
$$T_{turnaround}(r_i) = \sum_{j=0}^{i} T_{exec}(r_j) + \sum_{j=0}^{i} T_{idle}(r_j) \quad (2)$$

*1) Large Latency Variation:* The increased latency variation directly increases the idle time. This relationship is illustrated in Figure 2(a), where the idle time corresponds to the latency gap between different types of pages. Specifically, when NAND flash transitions to the QLC era, the read latency gap between the LSB page and MSB page expands significantly. Consequently, the idle time increases accordingly, leading to performance degradation.

*2) Increased Number of Planes in A Single Die:* NAND flash manufacturers aim to increase die density and reduce circuit area overhead by incorporating multiple planes within a single die [7]–[9]. An example, as illustrated in Figure 2, demonstrates its impact on the idle time. When reading four different pages from a 2-plane die (Figure 2(a)), Plane 1 (P1) and P2 do not experience idle time because the read transactions on these planes are the long-latency commands of die 0 and die 1, respectively. The idle time of P0 is determined by the long-latency command of die 0, calculated as $T_{CHSB} - T_{CLSB}$. Likewise, the idle time of P3 is given by $T_{MSB} - T_{LSB}$. However, if there are four planes in a die (Figure 2(b)), only




(a) ***Percentage of Idle Time across Various SSDs.*** *The idle time is calculated as in Section III. For 8CH/1PL, the idle time is 0 since read command execution time is equal to transaction execution time with a single plane.*




(b) ***Normalized Average Read Response Time across Various SSDs.*** *The results are normalized to 2CH/4PL TLC.*

Fig. 3. ***The Impact of Latency Variation across Various SSDs.*** *CH means chip and PL means plane.*

P2 does not experience idle time. P0, P1, and P3 have idle time equal to $T_{MSB} - T_{CLSB}$, $T_{MSB} - T_{CHSB}$, and $T_{MSB} - T_{LSB}$, respectively. Evidently, the idle time increases with the number of planes.

To confirm the analysis, as shown in Fig 3, we evaluate the percentage of idle time and the average execution time of independent read commands in SSDs with 1/2/4-planes and TLC/QLC using the setup in Section V. The read latency parameters are from previous study [6]. The experimental results show a 147% increase in the percentage of idle time from 4CH/2PL to 2CH/4PL for QLC. Besides, we can observe that compared to TLC, the average read response time of QLC exhibits a greater increase from 8CH/1PL to 2CH/4PL. These experimental results suggest that the 4-planes QLC SSD architecture severely slows down the independent read command execution, motivating us to explore a potential approach to address this challenge.

## IV. Design of LITA

### A. Overview

Section III shows that the idle time that hampers read performance stems from the read latency variation. Based on this insight, we propose a novel independent read scheme enabled with latency variation awareness called LITA. LITA is divided into three stages, namely transaction combination, command execution, and transaction completion service, as illustrated in Figure 4. In Stage 1, the Least Idle Time (LIT) transaction combination merges transactions with similar access latency to form an independent read command. Stage 2 is command execution, which is unchanged from the original SSD controller. Stage 3, referred to as the Transfer As Soon As Possible (TASAP) transaction service, focuses on promptly notifying the controller of completed fast transactions within the command sequence. TASAP achieves this by actively polling the status register of each plane. Once a transaction is completed, TASAP promptly transfers the read data from

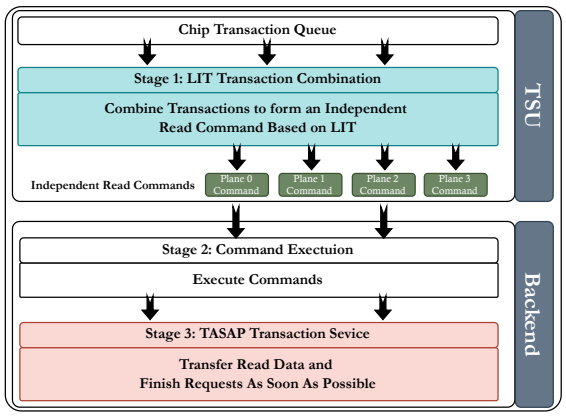



Fig. 4. ***Design Overview.*** *The design contains two components: LIT takes effect in stage 1 (transaction combination) and TASAP takes effect in stage 3 (handlement of transaction completion).*

the data register to the controller. Detailed descriptions of LIT and TASAP are provided in the subsequent subsections.

### *B. Least Idle Time (LIT)*

We introduce the Least Idle Time (LIT) principle, which focuses on minimizing idle time through combining read transactions with similar access latency. However, identifying the optimal transaction combination with the least idle time involves an exhaustive search through all transaction sequences for each plane, resulting in a computationally demanding task known to be NP-hard (expotential time-complexity). To overcome this challenge, we propose a heuristic design that scans the transaction queue only once to form only one independent read command each time (linear time complexity), effectively reducing the computational overhead.

LIT begins by reserving an empty slot for each plane (i.e., a slot array). Then, LIT scans the chip queue sequentially, examining each transaction. The candidate slot for the currently examined transaction is determined based on its target plane address. If the candidate slot is unoccupied, the transaction is directly placed into that slot. Otherwise, LIT estimates the idle time as in Section III between the existing transaction in the slot and the currently examined transaction is made. Specifically, the independent read execution time is the largest transaction execution time; idle time is the sum of the intervals between independent read execution time and transaction execution time for all transactions in the slots. The candidate slot will be allocated to the transaction with the lesser idle time. This process continues until all transactions in the chip queue have been scanned. Algorithm 1 outlines the key steps involved in implementing the LIT principle under IMP. For SOML, the only needed modification is that the SOML constraint should be enforced: in case of a conflict between the currently examined transaction and the existing transactions in other slots, the former one is skipped.

### *C. TASAP*

We introduce TASAP (Transfer data As Soon as Possible) to further reduce latency. TASAP permits the prompt transfer of read data once a transaction has been completed, thereby accelerating the completion of requests. Specifically, conventional

**Algorithm 1** ***LIT principle-based transaction combination.***

**Input:** *Transactions[N]:queued read transactions*
**Data:** *Slots[NumPlane],iterTransaction,candidateSlot*

```
1:  candidateSlot ← 0;
2:  Slots[NumPlane] ← [null, null, ..., null];
3:  for iterTransaction in Transactions do
4:      candidateSlot ← iterTransaction.plane;
5:      if IsNull(Slots[candidateSlot]) then
6:          nullIdleTime ← EstimateIdleTime(Slots);
7:          Slots[candidateSlot] ← iterTransaction;
8:          newIdleTime ← EstimateIdleTime(Slots);
9:          if newIdleTime ≥ nullIdleTime then
10:             Slots[candidateSlot] ← null;
11:         end if
12:     else
13:         oldTransaction ← Slots[candidateSlot];
14:         oldIdleTime ← EstimateIdleTime(Slots);
15:         Slots[candidateSlot] ← iterTransaction;
16:         newIdleTime ← EstimateIdleTime(Slots);
17:         if newIdleTime ≥ oldIdleTime then
18:             Slots[candidateSlot] ← oldTransaction;
19:         end if
20:     end if
21:     if EstimateIdleTime(Slots) = 0 then
22:         break                    ▷ Reach minimum idle time.
23:     end if
24:     iter ← iter + 1;
25: end for
```

SSDs unaware of latency variation do not transfer data until the entire command sequence has been executed. Consequently, fast transactions experience delays as they await the completion of the slowest transaction. TASAP, however, transfers the read data as soon as a transaction on any plane is completed, finishes the corresponding request if possible. TASAP requires that each plane data register at page unit (subpage unit for SOML) has an independent completion status. The channels poll the data registers to get the status and transfer data if possible. This implementation bears resemblance to cache read/program and introduces minimal overhead if any [4].

Figure 5 provides a concrete example of how TASAP works. In this example, let us assume each request consists of a single transaction; therefore, the request is completed when the transaction is completed. There are one CLSB page read transaction and one MSB page read transaction on Plane-3 (P3). P1 and P2 each have two page read CLSB transactions. P0 has a single CLSB page read transaction. These transactions can be combined into two independent read command sequences (IR1, IR2). Based on Liu's approach, read data transfer for CLSB transactions cannot take place until the MSB transaction has completed. However, with TASAP, read data transfer of CLSB transactions occurs immediately upon completion of each CLSB transaction even the MSB transaction is ongoing. This strategy shortens the delay for these CLSB transactions and completes the corresponding requests earlier. By implementing

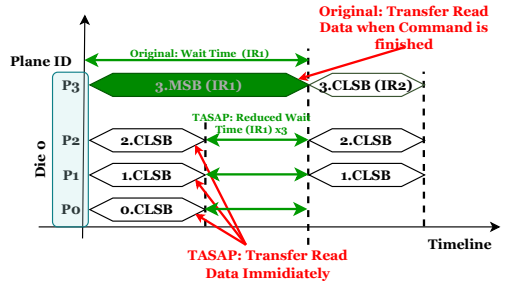



Fig. 5. ***An Example of TASAP.*** *The red text/lines means the conventional scheme and TASAP. The green text/lines means the original wait time and wait time reductions. The blue line/text shows the TASAP's benefit for improving independent read command parallelism. 0.CLSB means it is a read transaction which needs to read a CLSB page in plane 0 (others are similar).*

TASAP, the transactions collectively experience a reduction in waiting time equal to $3 * (T_{MSB} - T_{CLSB})$.

## V. Evaluation

### A. Experimental Setup

In our model-based evaluations, we employ MQSim [10], which is a state-of-the-art simulator for SSDs. The SSD setup is shown in Table I. We use the out-of-order scheduler and static page-level mapping. We select various traces from the MSR-Cambridge dataset [12]. The workload characteristics are omitted due to limited pages. Before replaying the traces, the SSD is pre-filled with 95% of its available logical capacity. Following methodologies similar to prior studies [5], [13], we intensify these traces (25 times more intense in our study) to emulate modern SSD workloads. Given our primary focus on reducing read latency, we implement a simplified SOML by reducing page size, increasing the number of planes, and enforcing the SOML read constraint. The SOML subpage size is set at 8KiB. Liu's approach serves as the baseline. LIT/TASAP denotes the application of only LIT/TASAP. LITA signifies the combined application of both LIT and TASAP. The geomean represents the geometric mean value of the results.

TABLE I
Configuration of the simulated SSD.

| | |
|---|---|
| *# of channels/chips/dies/planes* | *4/2/1/4* |
| *# of blocks in a plane* | *1024* |
| *# of pages in a block* | *1024* |
| *page size (KiB)* | *16* |
| *read latency (µs)* | *(48, 76, 134, 228)* |
| *program latency (µs)* | *1500* |
| *block erase latency (µs)* | *3800* |
| *percentage of over-provisoned space* | *27%* |

### B. Experimental Results

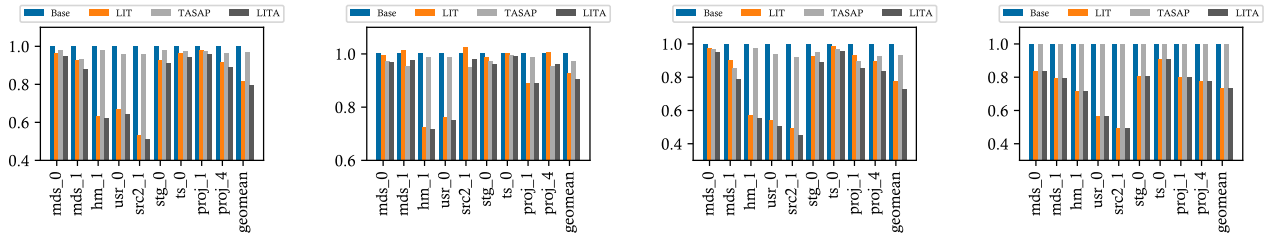
***#Experiment.1 I/O Latency.*** Figures 6(a) and 6(b) depict the normalized read latency for IMP and SOML. Write latency evaluation, with negligible improvement, is omitted. Compared to the IMP baseline, LIT reduces average read latency by 18.2%, TASAP reduces it by 3%, and LITA achieves a 20.4% reduction, with a maximum of up to 48.7%. For SOML, LITA exhibits less performance improvement (9.6% reduction on average) due to unique constraints on transaction combinations and its different address allocation, and mapping caused by smaller page granularity, influencing the potential benefits of LITA. Interestingly, for some workloads, LIT marginally increases latency, attributed to its inherent greedy nature that may prioritize local optimizations, potentially overlooking globally optimal solutions.

***#Experiment.2 Idle Time and Transaction Wait Time.*** We evaluate idle time and read transaction wait time, as in Figure 6(d) and Figure 6(c). Notice that we merely present results under IMP, as the outcomes for SOML exhibit a similar trend. We observe that LIT yields an average reduction in idle time of 26.9%. Along with Section III, we can observe that the effectiveness of LITA increases as the idle time percentage, the effect of idle time and the reduction in idle time increase. LIT, TASAP and LITA contribute to reductions in transaction wait time (notice it is not request wait time), with figures of 22.4%, 6.7% and 27.2%, respectively. This result further explains the performance advantage offered by LITA.

***#Experiment.3 Impact of Intensiveness.*** We evaluate read latency when re-running the traces at intensities ranging from 10 to 25 times higher than the initial workload to investigate the impact of workload intensiveness on the effectiveness of LITA. The results as in Figure 7 demonstrate that LITA's advantages become more pronounced as workloads intensify. In instances of workloads surpassing 25 times in intensiveness, LITA accomplishes additional read latency reduction, the specific results of which are omitted due to page limits. This phenomenon can be attributed to the fact that more intensive workloads tend to result in a greater number of transactions in the chip queues, thereby creating more opportunities for LIT transaction combinations. The effectiveness of TASAP in improving read latency is diminished, as it primarily focuses on reducing transaction wait time by addressing the execution latency gap between different pages (up to 180$\mu$s based on our flash parameters in Table I, representing the latency gap between the slowest and fastest page reads). The diminishing benefit becomes more apparent under heightened workload intensiveness, leading to a noteworthy increase in overall read latency. Consequently, the wait time caused by latency gap constitutes a smaller portion of the total read latency.

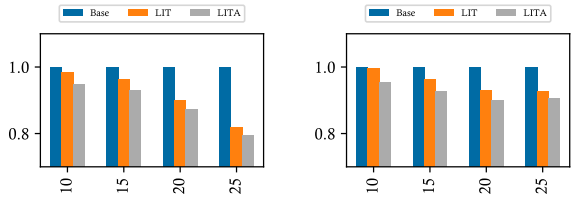
## VI. Related Works

Previous studies have endeavored to investigate latency variation as a means of enhancing I/O performance. Cui et al. proposed some scheduling techniques based on latency variation [14]. Lv et al. proposed a latency variation aware data placement scheme [15]. Chen et al. designed a page type aware cache [6]. These studies did not consider independent read and are orthogonal to our work. LITA could be enhanced by integrating their optimizations. Some other works used independent read to reduce/accelerate read-retries. For example, Cui et al. [16] proposed 3D NAND layer Raw Bit Error Rate (RBER) aware read-retry strategies based on SOML to reduce read-retries. Nie et al. [17] also focuses on the layer RBER variation of 3D NAND flash and proposed a layer RBER aware I/O scheduling scheme based on SOML. These studies mainly concentrate on SOML on 2-planes TLC SSDs and emphasize the optimization of read-retries stemming




(a) ***Average read lat with IMP.*** (b) ***Average read lat with SOML.*** (c) ***Average transaction wait time.*** (d) ***Idle time.***

Fig. 6. ***Read Latency, Idle Time and Transaction Wait Time.*** *The results are normalized to Baseline*




(a) ***IMP.*** (b) ***SOML.***

Fig. 7. ***Normalized Geometric Mean Read Latency of Traces across Various Intensiveness.*** *Y-axis is the average read latency normalized to baseline. X-axis is the re-rated times.*

from RBER. Comparatively, our work elucidates that page read latency variation is an pivotal contributing factor to performance degradation in the context of 4-planes QLC SSDs. Consequently, we proposes a straightforward yet effective strategy to address the challenge.

Some flash memory vendors have implemented asynchronous IMP, wherein each plane can execute read commands akin to a die or chip, thereby enhancing IMP. In such cases, LITA maybe not effective. However, asynchronous IMP possibly introduces additional hardware modifications to address issues incurred by asynchronous operations [18]. LITA, as a firmware-level proposal, alleviates the problem with advantage of requiring minimal hardware modifications. Besides, SOML and current ONFI can not support asynchronous operations so far [4]. Hence, LITA temporarily focuses on the SOML and synchronous IMP for now. In the near future, LITA will be extended within the context of asynchronous IMP.

## VII. Acknowledgements

We greatly thank the anonymous reviewers for their helpful and insightful feedbacks. This work was sponsored in part by the National Natural Science Foundation of China under Grant No.61821003 and Grant No.62172178.

## VIII. Conclusion

The integration of additional planes within a single die, coupled with increased read latency variation across different page types, exacerbates the negative impact of the current SSD architecture on independent read performance. To address this issue, we propose LITA, which reduces idle time through a greedy transaction combination and reduces wait time by promptly transferring data upon completion. The experimental results demonstrate that LITA significantly reduces read latency across various workloads. In the future, we plan to extend LITA to incorporate more advanced independent read commands, such as asynchronous IMP.

## References

[1] A. Goda *et al.*, "Recent Progress on 3D NAND Flash Technologies," *Electronics*, vol. 10, no. 24, p. 3156, 2021.

[2] C. Liu *et al.*, "SOML Read: Rethinking the Read Operation Granularity of 3D NAND SSDs," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 955–969.

[3] N. Shibata *et al.*, "A 1.33Tb 4-bit/Cell 3D-Flash Memory on a 96-Word-Line-Layer Technology," in *2019 IEEE International Solid State Circuits Conference (ISSCC)*, 2019, pp. 210–212.

[4] "ONFI Specification August 11, 2022." [Online]. Available: https://www.onfi.org/

[5] Y. Lv *et al.*, "MGC: Multiple-Gray-Code for 3D NAND Flash based High-Density SSDs," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 122–136.

[6] Q. Chen *et al.*, "PACA: A Page Type Aware Read Cache Scheme in QLC Flash-based SSDs," in *IEEE 40th International Conference on Computer Design (ICCD)*, 2022, pp. 59–66.

[7] D. Kang *et al.*, "A 512Gb 3-bit/Cell 3D 6 th-Generation V-NAND flash memory with 82MB/s write throughput and 1.2 Gb/s interface," in *2019 IEEE International Solid State Circuits Conference (ISSCC)*. IEEE, 2019, pp. 216–218.

[8] J. Cho *et al.*, "A 512Gb 3b/Cell 7 th-Generation 3D-NAND Flash Memory with 184MB/s Write Throughput and 2.0 Gb/s Interface," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 426–428.

[9] M. Kim *et al.*, "A 1Tb 3b/Cell 8th-Generation 3D-NAND Flash Memory with 164MB/s Write Throughput and a 2.4Gb/s Interface," in *2022 IEEE International Solid State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 136–137.

[10] A. Tavakkol *et al.*, "MQSim: A Framework For Enabling Realistic Studies of Modern Multi-Queue SSD Devices," in *16th USENIX Conference on File and Storage Technologies (FAST)*, 2018, pp. 49–66.

[11] Y. Hu *et al.*, "Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity," in *Proceedings of the International Conference on Supercomputing (ICS)*, 2011, pp. 96–107.

[12] D. Narayanan *et al.*, "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 145–158.

[13] H. Li *et al.*, "IODA: A Host/Device Co-Design for Strong Predictability Contract on Modern Flash Storage," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 263–279.

[14] J. Cui *et al.*, "Exploiting Latency Variation for Access Conflict Reduction of NAND Flash Memory," in *32nd Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2016, pp. 1–7.

[15] Y. Lv *et al.*, "Latency Variation Aware Read Performance Optimization on 3D High Density NAND Flash Memory," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 411–414.

[16] J. Cui *et al.*, "Improving 3D NAND SSD Read Performance by Parallelizing Read-Retry," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 3, pp. 768–780, 2023.

[17] S. Nie *et al.*, "Layer RBER Variation Aware Read Performance Optimization for 3D Flash Memories," in *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[18] T. Higuchi *et al.*, "A 1Tb 3b/Cell 3D-Flash Memory in a 170+ Word-Line-Layer Technology," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 428–430.