# Need for a Deeper Cross-Layer Optimization for Dense NAND SSD to Improve Read Performance of Big Data Applications: A Case for Melded Pages

Arpith K
*Indian Institute of Science, Bangalore*

K. Gopinath
*Indian Institute of Science, Bangalore*

## Abstract

In the case of dense NAND flash such as TLC, the LSB, CSB and MSB[1] pages in a wordline can be combined to form a larger logical page called melded-page. In this paper, we propose melding TLC/QLC pages to improve the performance of SSD by mitigating the overheads involved in the read operation. Melded-pages are read in their entirety. The controller schedules the write requests in such a way that, during reads later, requests for data present in LSB, CSB and MSB pages are guaranteed to be present in the request queue. This method works reliably when the workload performs its read operations in large chunks or has a predictable I/O pattern. We obtain performance improvements of up to 45% on workloads that use large block sizes such as the Hadoop Distributed File System (HDFS). Big data solutions that exhibit such read patterns can vastly benefit from melding pages.

## 1 Introduction

Solid State Drives based on NAND flash use transistors to store information persistently. Data is stored as a threshold voltage in each flash cell[2]. Due to its energy efficiency, small form factor and resilience to physical shock, they are the storage medium of choice in a variety of mobile devices. Internal parallelism facilitates SSDs to deliver a significantly higher I/O performance when compared to traditional magnetic disks, and are hence used in data centers. Modern flash memories have transistors that allow it to store multiple bits, thus enabling the production of SSDs with higher capacities and a low cost-per-bit. Cells with an ability to store three bits are being widely used, with Intel and Micron announcing even the availability of the commercial SSD with quad-level cells.

However, such high-density SSDs suffer from longer latencies to program and read data, resulting in reduced throughputs, when compared to flash memories that store a single bit per cell.

[1] In a TLC Flash, each transistor can store three bits of data. The bits are referred to as Least Significant Bit (LSB), Center Significant Bit (CSB) and Most Significant Bit (MSB) in this paper

[2] TLC cell can accommodate eight distinct threshold voltage states, thus, enabling it to store 3 bits per cell.

### 1.1 Sources of Read-Overheads

When a read request arrives via the host interface, the controller needs to determine the physical page address of the requested data in the underlying flash memory. To do this, the controller runs a firmware called the Flash Translation Layer to enable this address translation. It maintains a Flash Translation Table to manage the mappings from logical addresses to physical addresses. This process of mapping adds to read latency.

The controller then sends this address along with the read command to the control unit in the flash chip. Block and page decoders are then used to access the correct wordline [6].

However, before reading the data from the selected wordline, all other wordlines in the selected block are made conductive. This setup operation is required because the sense amplifiers are connected to the selected cells on a shared bitline [3, 7, 10]. A voltage, greater than the maximum threshold voltage of the transistor, called the passthrough voltage is applied to all the unselected cells to ensure they are conductive. The conductivity of the selected cell can only be determined after this setup operation. The operation to set up a block to ensure the readability of the chosen wordline further adds to the read overheads.

The controller may also choose to perform various post-processing operations after the page is read. One such task involves detecting and correcting bit errors. Such operations also add to the read latencies.

Furthermore, in a dense SSD, such as TLC flash used in this paper, one, two and four read reference voltages are required to read LSB, CSB and MSB pages respectively. Table 1 shows the latencies to read different page types from a triple-level cell [5]. Ideally, one would expect the latency to read an MSB page to be four times that of an LSB page and two times for a CSB page. This is not observed in practice due to overheads involved in the read operation.

## 2 Melded-Pages in Dense SSDs

Our approach to alleviate the effects of read overheads involves combining the LSB, CSB and MSB pages in a wordline into a single larger page called a melded-page. When a read request arrives, the entire melded-page is read in one command cycle. The scheduling algorithm ensures that the request for all the data in a melded-page is present in the request queue (or will arrive soon).

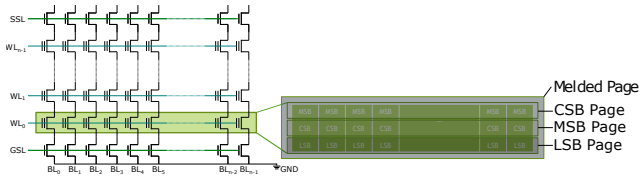


Figure 1: Melded-Page

To guarantee this, the scheduler modifies the order in which the pages are programmed. It must be noted that data is written to melded-page at page granularity, as opposed to melded-page granularity, using shadow programming sequence and data stripping across parallel units. The pattern in which data is programmed to the flash memory remains unchanged.

### 2.1 Modern SSD Background

#### 2.1.1 Parallelism and Data Striping in SSD

The high throughput of a NAND flash SSD can be attributed to its internal parallelism. There are four levels of parallelism in an SSD. When a write request arrives, the controller stripes the data across channels, packages, dies, and planes. Multiple schemes to stripe data and allocate this striped data across parallel units exist [4]. Figure 2 shows the Plane-Channel-Package-Die allocation scheme. The exact method used depends on the manufacturer.

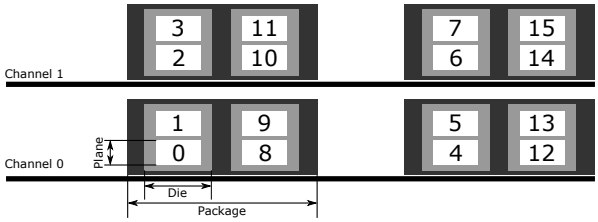


Figure 2: Plane-Channel-Package-Die page allocation strategy

#### 2.1.2 Shadow Program Sequencing

Modern processes to fabricate semiconductors has resulted in the production of dense integrated circuits. The reduced distance between the transistor cells creates a significant parasitic capacitance between them. Because of this, programming a flash cell results in a threshold shift in adjacent cells as well. This unwanted shift in threshold values of victim cells is known as cell-to-cell interference [8]. Shadow page programming sequence minimizes the effect of this interference by ensuring the interleaving of writes between the wordlines. Figure 3 shows the sequence in which the pages are programmed in a TLC flash block [3].

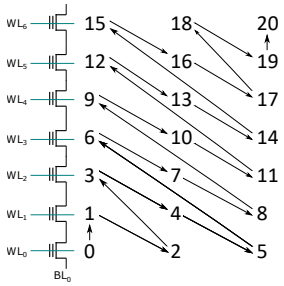

Figure 3: Shadow program sequence. Here, the three columns represent the LSB, CSB and MSB pages of the first seven wordlines

### 2.2 Scheduling

Melded-page mitigates overheads by reading all data in a wordline of a TLC flash in one command cycle. The process of writing pages in shadow program sequence and striping I/O request across parallel units may prevent the data, when requested later during the read operation, to be in the same melded-page. Reading an entire melded-page when only a part of it is used to fulfill a read request has a negative impact on the read performance. To prevent this, the scheduler rearranges the order in which striped data is pipelined for writes in a manner which compensates for the change in order due to the processes mentioned earlier. This reordering of writes by the scheduler ensures that, later, during reads, requests for the data present in LSB, CSB and MSB pages are guaranteed to be present in the read request queue.

As an example, let us consider an SSD with just one parallel unit. Also assume that we know the requests or read stripes 0, 1 and 2 will all be in the read request queue. With this knowledge, we reorder the way in which the stripes are written as shown in Figure 4. Stripe 3 is written before 1 and stripes 4 and 5 are written before 2, so that 0, 1 and 2 can all be in the same melded page.

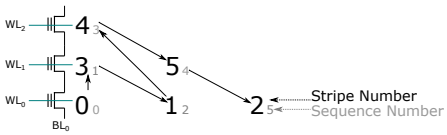


Figure 4: Reordering of stripes

We take another example of an SSD with two channels, each having two packages. Assuming the resources are allocated using a Plane-Channel-Package-Die allocation scheme, Figure 6 and Figure 5 illustrates the spread of data across

flash memory with and without the scheduler rearranging the order in which the pages are programmed. Only the contents of package 0 are shown completely for the sake of simplicity. In general, the scheduling algorithm depends upon the I/O write patterns, striping policies and page allocation strategies.

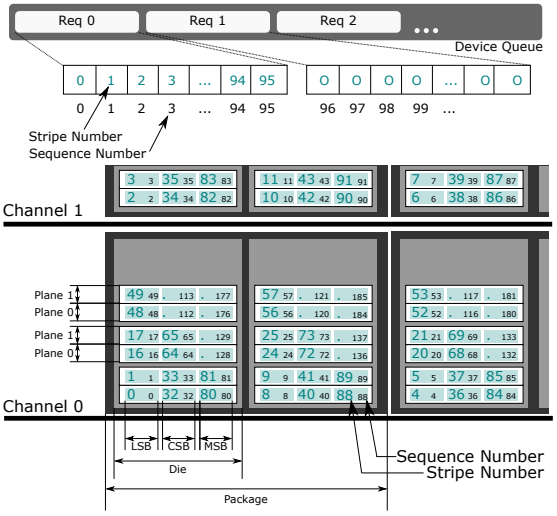


Figure 5: An example of how the striped data is spread across parallel units

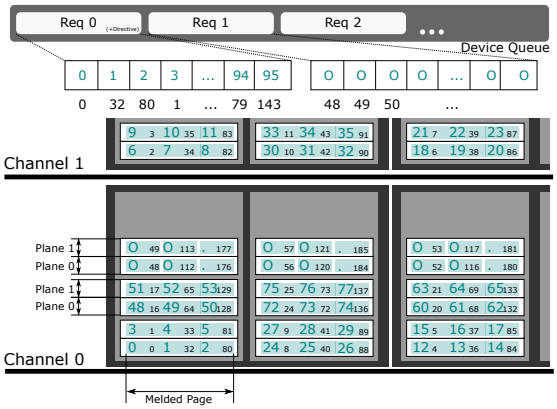


Figure 6: An example of how the striped data is spread across parallel units with scheduling (to guarantee that later, while reading, requests for data in LSB, CSB and MSB pages are all present in the read request queue.)

Ideally, for a system to benefit from melded pages, it must perform large read operations (or, read large blocks), in a predictable pattern. Systems that use large block sizes such as Hadoop Distributed File System (HDFS) are good use cases for melded-pages.

### 2.2.1 Hadoop Distributed File System

Hadoop and Spark are popular open-source cluster-computing frameworks used for big data analytics and large-scale data processing. One of the most popular distributed data storage system to manage large data sets is Hadoop Distributed File System [9].

Like most filesystems, HDFS allows the user to create and delete directories, and read, write and delete files. HDFS uses master-slave architecture. A master NameNode server manages the filesystem and DataNodes are responsible for serving the client's read and write requests.

When an application or user writes a file, the system splits it into smaller chunks, knows as blocks. These blocks are stored in various slave nodes in the HDFS cluster. The block size, by default, is 128MB [1]. With an exception of the final data block for a file, which uses as only as much space as it is needed, a block is the smallest unit of data that HDFS uses to read or write.

When an application or a user reads a file, the HDFS client raises a request to the NameNode for the list of DataNodes that host the requested file blocks. The client then contacts the topologically closest DataNode and directly initiates the transfer of the desired blocks.

Since HDFS performs all write and read operations in large chunks and in a sequential manner, the scheduler can predict read patterns and rearrange write requests to guarantee that, later, all requests for data present LSB, CSB, and MSB pages are present in the read request queue. The use of melded-pages shows a considerable improvement in read-throughput of SSDs in DataNodes.

## 2.3 NVMe Directives

The scheduler only reorders those requests that can benefit from melding pages. In the case of HDFS, for example, only write requests, to SSD, which correspond to the writing of HDFS block data needs to be reordered. The controller identifies such requests with the help of hints provided to it, by the host, known as directives.

The specifications for NVM Express v1.3 [2] defines a framework for directive support. It introduces a mechanism to exchange extra metadata between the host and the controller. I/O command directives add an ability for the host to tag write commands with directive type and an associated directive specific value.

Currently, the NVM Express specs define two directive types; identity and stream [2]. We propose a new use case for steams. For a controller that supports melded-pages, all write requests to SSD generated from writing an HDFS block forms a stream and are tagged as such. This is used by the controller to identify write requests that need to be reordered by the scheduler.

It is possible for the host application to start writing a new HDFS block while another HDFS block is already being written. Write requests to SSD with data from different HDFS blocks are interleaved in this case. The host uses a different stream identifier to prevent the melding of pages that contain data from various HDFS blocks. The controller opens a new

stream for each HDFS block and pages that originate from different sources are written to different SSD blocks. Stream identifiers and allocated blocks are released after the SSD completes writing an entire HDFS block.

## 2.4 Limitations and workarounds

The following are the limitations of using melded-pages.

1. **The scheduler needs to predict the read pattern of the workload**
   To benefit form melding pages, the scheduler needs to predict the read pattern of the workload. This enables the scheduler to combine multiple pages (three in case of TLC flash), which are guaranteed to be read simultaneously in the future, and program them to the same wordline. Melded-pages cannot be used if this prediction is not possible. The host provides directives to the SSD controller, along with the write request, to enable or disable the use of melded-pages for that request.

2. **Small read requests have a negative impact**
   Read requests which are fulfilled from different parallel units have a better performance than those served from a single die/plane. Also, the lack of contentions in I/O bus for requests served form parallel units in different channels results in the best performance. Thus, melding pages in a single plane when they could have been assigned different parallel units impacts read throughput.

   When request sizes are large enough, all data stripes cannot be assigned to pages in different parallel units. Pages are melded only in this case. The minimum size of a request to give an improvement in read-throughput with the use of melded-pages is discussed later in section 3.2.1.

3. **A page buffer with a higher capacity is required**
   In case of TLC flash, an operation to read from a melded-page retrieves three times the amount of data in one command cycle when compared to reading a normal page. Page buffers in each plane must be three times as large to accommodate this. The cost of this hardware change is justifiable only if majority of I/O operations performed by the workload can take advantage of melded-pages.

# 3 Evaluation

## 3.1 Experimental Setup

For the purposes of evaluation, we run Hadoop v2.9.1 with Hadoop Distributed File Systems. The workload used does not matter since the I/O pattern of reads and writes from the perspective of the host is consistently sequential due to large block sizes used by HDFS. For our simulation environment, we use SimpleSSD [5], a high-fidelity simulator, and implement Melded-Pages on top of it. Table 2 summarizes the configuration of the SSD. The following describes various parameters used in the implementation.

### 3.1.1 Estimated Latency to Read a Melded-Page

We assume the time it takes to read a melded page is 166$\mu$s. Assuming $X$ denotes the latency due to the overheads involved in accessing a page and $Y$ is the amount of time to apply a read reference voltage and test the conductivity of a cell, to read an LSB page, which requires only one read reference voltage, the latency can be approximated to $X+Y$. Similarly, the latency to read a CSB page, which requires two read reference voltages, is $X+2Y$. Reading an MSB page from a TLC flash requires four reference voltages. Hence, the latency to read an MSB page can be approximated to $X+4Y$. The total time to read all three pages will hence be $3X+7Y$. However, the latency to read an entire melded-page in one go is $X+7Y$. In this work, we approximate the value of $X$ to 40$\mu$s and $Y$ to 18$\mu$s. Table 1 shows the latencies to read different page types form a flash chip.

| Page Type | Latency ($\mu$s) | Latency MP ($\mu$s) |
|---|---|---|
| LSB Page | 58 | 166 |
| CSB Page | 78 | |
| MSB Page | 107 | |

Table 1: Read Latency with Melded TLC

| Configuration Parameter | | Value |
|---|---|---|
| NAND Type | | TLC |
| Page Size | | 2KB/4KB/8KB/16KB |
| Melded-Page Size | | 6KB/12KB/24KB/48KB |
| Number of Parallel Units per Channel | | 8/4 |
| Number of Channels | | 8/16 |
| Total Number of Parallel Units | | 64 |
| Channel Properties | Data Frequency | 400MT/s, 800MT/s, 1600MT/s |
| | Channel Width | 8/16 bits/transfer |

Table 2: Configuration parameters of the simulated SSD

## 3.2 Results

To quantify the performance improvements with the use of melded-pages, we measure the read throughputs of the SSD. We estimate the minimum read size that an application must request to benefit from melded-pages and then report the improvements in read-throughput performance for reading HDFS blocks.

### 3.2.1 Minimum Read Size

As mentioned in section 2.4, melding pages is only beneficial for workloads that read a large amount of data from an SSD. To determine the minimum size of data, that needs to be read in a predictable pattern, to benefit from melded-pages, we increment the read size and measure the time to complete each read operation. We simulate experiments on an SSD with 4KB page size and with a data frequency of 800MT/s. The

SSD has eight channels and each channel has eight parallel units, giving us a total of sixty-four parallel units. The results are tabulated in table 3.

| Read Size (B) | Normal TLC ($\mu$s) | Melded TLC ($\mu$s) |
|---|---|---|
| $2^{12}$ | 63 | 183 |
| $2^{13}$ | 63 | 183 |
| $2^{14}$ | 63 | 183 |
| $2^{15}$ | 63 | 183 |
| $2^{16}$ | 69 | 183 |
| $2^{17}$ | 81 | 200 |
| $2^{18}$ | 104 | 218 |
| $2^{19}$ | 188 | 270 |
| $2^{20}$ | 364 | 401 |
| $2^{21}$ | 708 | 636 |
| $2^{22}$ | 1406 | 1134 |
| $2^{23}$ | 2791 | 2103 |
| $2^{24}$ | 5572 | 4068 |
| $2^{25}$ | 11124 | 7971 |
| $2^{26}$ | 22236 | 15803 |
| $2^{27}$ | 44452 | 31440 |

Table 3: Determining the minimum read size

Small reads have better throughput if the pages that hold the requested data is placed in different parallel units as opposed to melding them on the same die/plane. Requests that are served from pages on parallel units in different channels show the best performance. For an SSD, with an aforementioned configuration, the minimum size of data that must be read to benefit from melded-pages is 2MB (table 3).

### 3.2.2 Performance Improvements

Figures 7 and 8 show read throughputs with various SSD configuration for reading an HDFS block. We vary the page size, channel width, data frequency and report results for every combination with and without the use of mended-pages.

We observe a significant improvement in the read-throughputs of up to 45% while reading an HDFS block. This is the upper limit of the performance that can be achieved by using melded-pages with our configuration. Increasing data frequency or channel width will not improve the performance any further because of the latencies to read from the page. In other cases, when reading large melded-page sizes, the throughput of the channel can become a bottleneck. Increasing the channel's operating frequency/width or reducing the number of parallel units per channel solves this problem.

## 4 Conclusion

In this paper, we presented a solution to improve the read throughput of SSDs using melded pages, if knowledge of I/O patterns is available. For this to work, the scheduler rearranges the writes to ensure that the request for all data in a wordline is present in the read request queue. For workloads that use distributed file system, we can improve the read performance by up to 45%.

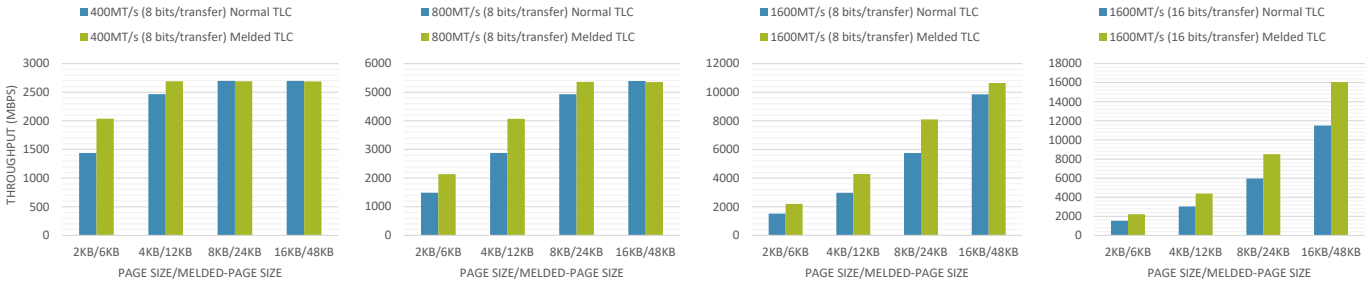


Figure 7: Read throughputs of SSD (8 channels; 8 parallel units per channel)

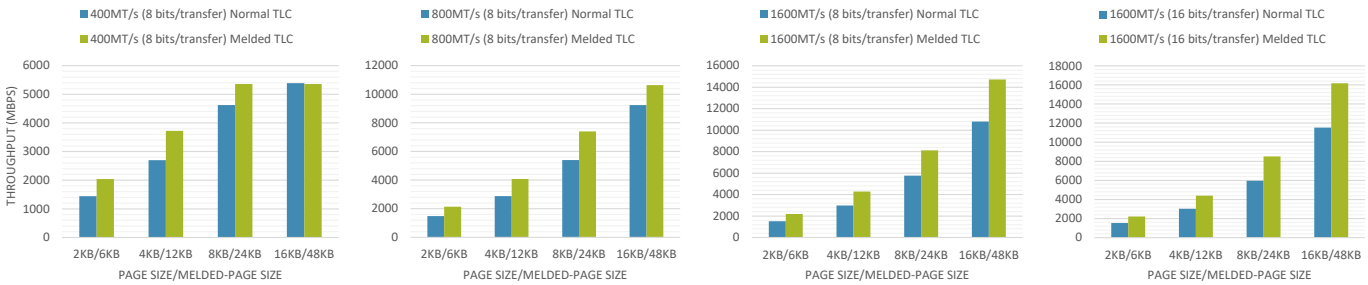


Figure 8: Read throughputs of SSD (16 channels; 4 parallel units per channel)

## References


[1] Hadoop Cluster Setup. https://hadoop.apache.org/docs/r1.2.1/cluster_setup.html.

[2] NVM Express specifications, 2017. http://nvmexpress.org/wp-content/uploads/NVM_Express_Revision_1.3.pdf.

[3] CAI, Y., GHOSE, S., HARATSCH, E. F., LUO, Y., AND MUTLU, O. Error characterization, mitigation, and recovery in flash-memory-based solid-state drives. *Proceedings of the IEEE 105*, 9 (Sept 2017), 1666–1704.

[4] JUNG, M., AND KANDEMIR, M. An evaluation of different page allocation strategies on high-speed ssds. In *Proceedings of the 4th USENIX Conference on Hot Topics in Storage and File Systems* (Berkeley, CA, USA, 2012), HotStorage'12, USENIX Association, pp. 9–9.

[5] JUNG, M., ZHANG, J., ABULILA, A., KWON, M., SHAHIDI, N., SHALF, J., KIM, N. S., AND KANDEMIR, M. SimpleSSD: Modeling solid state drives for holistic system simulation. *IEEE Computer Architecture Letters PP*, 99 (2017), 1–1.

[6] MICHELONI, R., CRIPPA, L., AND MARELLI, A. *Inside NAND Flash Memories*. 2010. http://link.springer.com/10.1007/978-90-481-9431-5.

[7] MOHAN, V. Modeling the physical characteristics of nand flash memory, 2010. https://pdfs.semanticscholar.org/207f/501498e1b1dc2efea16110a18e485357666c.pdf.

[8] PARK, K. T., KANG, M., KIM, D., HWANG, S. W., CHOI, B. Y., LEE, Y. T., KIM, C., AND KIM, K. A zeroing cell-to-cell interference page architecture with temporary lsb storing and parallel msb program scheme for mlc nand flash memories. *IEEE Journal of Solid-State Circuits 43*, 4 (April 2008), 919–928.

[9] SHVACHKO, K., KUANG, H., RADIA, S., AND CHANSLER, R. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (May 2010), pp. 1–10.

[10] SUH, K.-D., SUH, B.-H., LIM, Y.-H., KIM, J.-K., CHOI, Y.-J., KOH, Y.-N., LEE, S.-S., KWON, S.-C., CHOI, B.-S., YUM, J.-S., CHOI, J.-H., KIM, J.-R., AND LIM, H.-K. A 3.3 v 32 mb nand flash memory with incremental step pulse programming scheme. *IEEE Journal of Solid-State Circuits 30*, 11 (Nov 1995), 1149–1156.