International Journal of Advances in Electronics Engineering– IJAEE
Volume 5 : Issue 1 [ISSN : 2278-215X]
Publication Date : 30 April, 2015

A Data Layout strategy to Enhance the Internal SSD Parallelism

Soraya Zertal[1]

[1] PRiSM, University of Versailles, 45 Av. des Etats-Unis, 78000 Versailles, France

Abstract—Solid State Disks (SSDs) are promising data storage devices in term of performance and energy consumption comparing to Hard Drive Disks (HDDs). They are more and more used, even in data-intensive applications where hard constraints are put on throughtput and response time. Accessing data in parallel becomes capital to satisfy these performance requirements. The SSD internal structure provides a potential for parallel access at different levels, up to its dies and planes. However, this parallel access relies completely on the data layout. In this paper, a data placement algorithm is presented. It distributes data across the SSD components and maintains this layout after write operations. This makes the parallel requests execution possible at the lowest levels. The impact of this data layout on the reduction of the waiting time is evaluated and the obtained results show a significant gain for OLTP and scientific applications up to a factor of 10.

Index Terms—SSD, Parallel I/O, Data layout, Waiting time, OLTP and scientific workloads, Simulation.

I. Introduction

NAND-Flash Memories and Solid State Disks (SSDs) enregistred a substansial and continous increasing use. It exceeds the limits of mobile devices, personnal and enterprise machines. Recently, data centers incorporated SSDs into their storage systems and made them a target storage media for data-intensive applications. This use will be strengthened in the near future with the increasing storage capacity and the reducing cost of SSDs.
After a long exclusive use of HDD, hybrid systems or SSDs as an intermediate storage to accelerate the access to special data [17], data-intensive applications start to consider whole SSD storage systems.

An intensive work had been achieved to model and optimize the Inputs/Outputs execution on HDDs [12, 18, 21, 20, 16, 22, 23]. However, switching to SSDs, makes these optimizations inappropriate because the former has a moving part which is inexistant on the second one. Thus, SSDs have a certain advantage in term of performance as all mechanical delays are eliminated as well as their associated access strategies as "buffering I/O requests to execute them in sequence on HDDs for a reduced seek time" [5]. With their unique property of internal parallelism, SSDs allow their different components (packages, dies and planes) to process I/O requests in parallel and improve significantly the delivered performance.

components (packages, dies and planes) to process I/O requests in parallel and improve significantly the delivered performance.

However, this property is poorly investigated and exploited at this stage and only few works were achieved in this direction [14, 4]. The internal SSD parallelism matches with the natural concurrency present in data-intensive applications resulting from a concurrent calculation. It helps to fulfil the performance requirements in term of response time and delivered throughput and overcomes the poor random write performance. This property cannot be exploited, neither its gain reached, if the data layout does not garantee an optimal distribution across the SSDs internal components. In this paper, we present a strategy for the initial data distribution, then its conservation during write operations execution as reads have no impact on the data layout.
This paper addresses these issues, using simulation to show that the proposed data layout strategy allows and enhance the exploitation of the internal SSDs parallelism and makes it a challenging option for data-intensive applications as OLTP and scientifc applications.

II. Background and Related Works

A. Flash memory and SSD background

SSDs are storage devices based on NAND-Flash memory. There are two types: Single Level Cell (SLC) storing one bit per memory cell and Multiple Level Cell (MLC) storing many bits per memory cell. The MLC_n technology multiplies the storage capacity by having n-bits per cell, where n=2,3 or 4. The higher is n, the higher is the capacity but the lower is the reliability and the performance. The basic structure unit of a NAND-Flash is the page and it is the access unit of read and write operations. As shown on fig. 1, every page is composed of a large field for user data and a small one (spare) for metadata [1]. A fixed number of pages are gathered to form a block which is the unit of erase operations. Blocks are organized into planes to compose a Flash chip or die (see fig. 2). Finally, an SSD is an array of Flash packages, connected through channels. Every package is composed of one or many dies (see fig. 3).

Several specific characteristics make the Flash memory different from other classical storage devices. First, a big disparity between the execution time of read, write and erase operations. Second, the No-In-Place update, leads to write the new version on a valid (free) page, then invalidate the page containing the old version. Third, the erase before write constraint, requiring the erase of a location before writing a new data in it. A garbage collector is run periodically to write valid pages in target blocks and erase the blocks with only invalid pages to make free room for future writes. Last, the limited number of erase cycles makes the wear leveling a critical task.

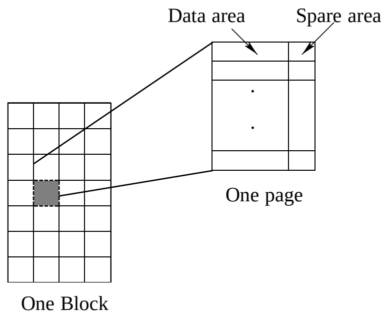

Fig. 1. A NAND Flash block structure

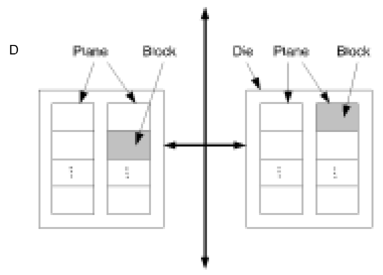

Fig. 2. A Die structure

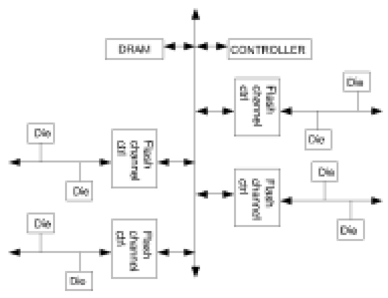

Fig. 3. A Solid State Drive structure

B. Related works

The deployment of SSDs in different environments raised up an intensive research work, proposing: (1) mapping and data placement strategies [19,

ing: (1) mapping and data placement strategies [19,

8, 1] to determine when it is more efficient to adopt a page mapping rather than a block one or a mixture of both with respect to metadata size and performance; (2) caching policies [15] to deal with the disparity between the operations latencies and consider not only hit count as usual for ”classical” memory devices but also the replacement cost caused by selecting dirty victims as its calculation is different for Flash in terms of time and energy; (3) wear leveling [3, 13] and (4) garbage collection [11, 10] to handle the No-In-Place updates, the erase-before-write and the limited number of erase cycles, in order to preserve the longevity of the device, and (5) simulation and evaluation tools [7, 2, 9]. In addition, several file systems have been developed to manage data on Flash memories as: JFFS2 (Journal Flash File System) and its successor UBIFS (Unsorted Block Image File System), YAFFS (Yet Another Flash File System) and its second version (YAFFS 2) and more recently, LogFS (Log Flasf File System) and F2FS (Flash-Friendly File System) and finally PFFS for hybrid Flash/RAM architectures.
Few works dedicated to SSDs, were achieved on its internal parallelism. Park et al. [14] studied the behaviour of the SSD submitted to parallel scientific I/O workloads using the completion time. Comparing to the HDD, the obtained gain is modest due to the high proportion of writes. The data layout changes after every SSD write and decreases dramatically the use of parallel execution. Chen et al. [4] studied more precisely the impact of the channel level parallelism on the performance of the execution queries in database systems. Their results are significant as the speed up of a join plan (read operations only), is 5.

III. Data Layout for Internal SSD Parallelism

From both studies [14, 4], one can deduce that :

1) Parallel access to the SSD can provide a substantial performance improvement but it relies completely on the physical data layout
2) Read operations have no impact on the parallel execution
3) Write operations change the data layout and prevent this parallel access.

Consequently,
Initially, data must be perfectly distributed across the concurrent components of the SSD, up to the lowest levels (dies and planes) with the page as a unit. This data layout should be protected against write operations. When a write operation occurs, the page holding the old data is invalidated and an other page is chosen to hold the new version. The location of this page is capital for keeping the

International Journal of Advances in Electronics Engineering– IJAEE
Volume 5 : Issue 1 [ISSN : 2278-215X]
Publication Date : 30 April, 2015

Trace	RD/WR ratio	I/O pattern	
		RD	WR
S trace 1	0.51/0.49	Seq.	Seq.
S trace 2	0.50/0.50	Seq.	Seq.
S trace 3	0.14/0.86	Rand.	Rand.
S trace 4	0.67/0.33	Rand.	Rand.
S trace 5	0.00/1.00	Rand.	Rand.

TABLE I
I/O PATTERN AND RD/WR RATIO FOR SCIENTIFIC WORKLOADS [14]

Parameter	Value
Packages per SSD	1
Dies per package	16
Planes per die	2
Blocks per die	4092
Pages per block	64
Block size	256 KB
Page size	4 KB
Erase duration	1.5 ms
Read duration	130 µs
Write duration	305 µs

TABLE II
THE SSD HARDWARE CHARACTERISTICS

distribution of data as optimal as in the initial layout. It is determined using our proposed algorithm DL_algo below:

```
1/ If a free page exists on the
   plane and the GC_indicator is
   lower than a certain threshold,
   this page is chosen, END

2/ If the condition (1) cannot be
   satisfied, select another plane
   not checked yet, in the same die
   and go to (1), otherwise (3)

3/ If all the planes of the die
   cannot provide a free page,
   choose the one with the
   highest GC_indicator, force
   a garbage collection, go to (1).
```

IV. Evaluation

First, the evaluation methodology is presented including the workloads generation and the simulation process, then the conducted tests are detailed.

A. Methodology

a) Workload generation: the OLTP workloads are generated according to the specific characteristics captured from an OLTP real trace [6]. It is composed of 67% of reads against 33% of writes. Operations are of one page (4KB) each, randomly distributed and coming at a rate of 1600 req/s. Scientific Workloads are generated using the characteristics captured in [14] from five real traces generated by real applications, called here S_trace 1 to 5. Table I summarizes the scientific I/O patterns and their read/write ratios. For the requests size distributions, all the granularities going from 1KB to 2MB are included with different percentages.

b) The simulated hardware: As the focus is on the internal structure of the SSD, only one was considered, with multiple dies and multiple planes within each die to provide such internal parallel access. Details are on table II.

B. Tests

Simulations were conducted using our algorithm DL_algo and compared to a random allocation policy. These tests were performed for the two parallelism degrees (1: interDies parallelism, 2: interPlanes parallelism) and no-parallelism as a reference (degree 0).

A set of tools were developed: a generator to produce the workloads and an event-driven simulator in C dedicated to the I/O requests execution. It implements all the possible parallel execution schemes and handles all the related synchronisation mechanisms associated with the three parallelism degrees (0, 1 and 2). It implements both the DL_algo allocation algorithm and the random one for comparison.

The parallel execution subdivides user requests into physical ones and distributes them across the dies, then the planes. Thus, the system is absolutely invariant whatever user requests are sequential or random. Also, it is considered under permanent use. So, time to mount metadata or to gather physical requests responses is not considered.

The focus is put here on the performance in term of response time, mainly composed of waiting and access time. This one is fixed on SSDs with respect to the operation type. For degree 0, the whole SSD package is seen as a unit and no other request is processed if the previous one is not completed. For degree 1, the dies can operate independently from each others and requests on different dies can be executed in the same time. For degree 2, not only dies can operate independently but also the planes within each die. It results in a highly parallel execution.

The simulation of every case is run for 250K user requests with 25% to reach the permanent state. The initial mapping and allocation algorithms spread widely data on the SSD components to allow the parallel access. the proposed DL_algo keeps this data layout whilst a random allocation choose a random block for the valid page to hold the new version. Thus, the spatial locality is broken without any impact as the pattern layout (seq/rand) is useless with SSD internal parallelism.

V. Results and Discussion

A. A parallelism oriented data layout strategy

A serie of simulations using OLTP and scientific workloads was performed to analyse the impact of the proposed data allocation algorithm (DL_algo) and compares it to the random one.
In case of no parallelism (degree 0), no change is observed which was expected because no parallelism is performed and only one request is executed in the whole package at once. The choice of the target page location for a write request has no impact then. The most important result, is that the DLalgo allocation algorithm does not introduce any delays and it is usefull only if a parallel execution is possible.

B. A parallelism oriented data layout strategy and interDies concurrent execution

In this case, Dies can operate in parallel, executing an I/O request each in the same time. The results are shown on figures 4 and 5 for OLTP and scientific workloads (write) respectively.

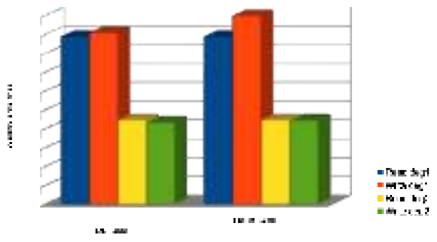

Fig. 4. Data layout Vs Random for OLTP workload

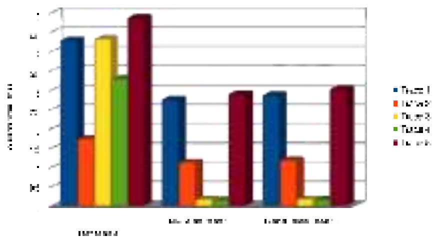

Fig. 5. Data layout Vs Random (degree 1) for scientific workloads

We can see for OLTP (fig. 4) that the waiting time for reads is the same for all data allocation strategies. This is not the case for the writes as the waiting time is significantly reduced using DL_algo. For scientific workloads (fig. 5), the waiting time is significantly reduced for the five traces even for the full write S__trace5.

C. A parallelism oriented data layout strategy and the interPlanes concurrent execution

C. A parallelism oriented data layout strategy and the interPlanes concurrent execution

The interPlanes parallelism is the extreme level to achieve parallel access. Thus, its exploitation can reduce drastically the waiting time. The hardware used in this study has two planes for each die: one plane composed of the odd blocks and the second plane composed of the even ones. The obtained results are shown on figures 4 and 6 for OLTP and scientific workloads (write) respectively.

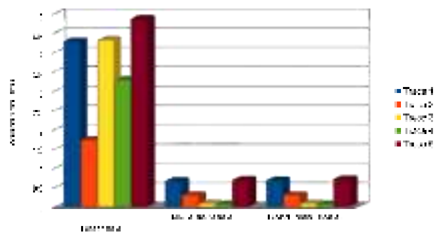

Fig. 6. Data layout Vs Random (degree 2) for scientific workloads

From both figures, we can see that the interPlanes parallelism reduce drasticaly the waiting time up to ×10. Both DL_algo and random allocation stratrgies allow this reduction with an advantage for DL_algo especially for long term execution when the data layout changes due to write operations make the parallel access hard to achieve.

VI. Conclusion

In this paper, a data layout algorithm DL_algo was proposed to enhance the exploitation of the SSD internal parallelism. Its impact was analysed at a fine grain parallelism: up to the planes level for data intensive applications such as OLTP and scientific applications using traces-based workloads. The results of the conducted simulations show a significant reduction factors for the waiting time, up to x10. The proposed DL_algo allows and enhance the parallel access to the lowest level of the SSD internal components. It is more performant comparing to the random algorithm at long term.
For the near future, considering the interPackages/interChannels parallelism will complete our study. Also, considering both external and internal parallelism in an SSD array will be a step forward to efficient large SSD-based storage systems for data-intensive applications.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, Mark, and R. Panigrahy. Design tradeoffs for ssd performance. In USENIX 2008 Annual Technical Conference

on Annual Technical Conference, pages 57–
70, 2008.

[2] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger. The disksim simulation envi- ronment version 4.0 reference manual cmu- pdl-08-101), 2008.

[3] L. P. Chang. On efficient wear leveling for large for large-scale flash memory storage systems. In ACM Symposium on Applied Computing, pages 1126–1130, 2007.

[4] F. Chen, R. Lee, and X. Zhang. Essantiel roles of exploiting internal parallelism of flash memory based solid state drives in high- speed data processing. In IEEE Symposium on High Performance Computer Architecture, pages 266–277, 2011.

[5] G. Graefe. the five-minute rule 20 years later and how flash memory chenges the rules. Queue, 6(4), 2008.

[6] P. G. Harrison, S. K. Harrison, N. M. Patel, and S. Zertal. Storage workload modeling by hidden markov models: Application to flash memory. Performance Evaluation Journal (PEVA), 69(1), 2012.

[7] Y. Hu, H. Jiang, D. Feng, Dan, L. Tian, H. Luo, and S. Zhang. Performance im- pact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In Proceedings of the In- ternational Conference on Supercomputing, pages 96–107, 2011.

[8] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang. Achiev- ing page-mapping ftl performance at block- mapping ftl cost by hiding address translation, 2010.

[9] K. Y. Kim, B. Tauras, A. Gupta, and B. Ur- gaonkar. Flashsim: A simulator for nand flash- based solid-state drives. In First International Conference on Advances in System Simula- tion, 2009. SIMUL ’09, pages 125–131, 2009.

[10] Y. Kim, S. Oral, G. Shipman, J. Lee, D. Dil- low, and F. Wang. Haormonia: A globally coordinated garbage collector for arrays of solid-state drives. In IEEE Symposium on Mass Storage Systems, pages 1–12, 2011.

[11] J. Lee, Y. Kim, G. Shipman, S. Oral, F. Wang, and J. Kim. A semi-preemptive garbage collector for solid state drives. In IEEE Sym- posium on Performance Analysis of Systems and Software, pages 12–21, 2011.

[12] K. Maruchi, S. Takakura, M. Yoshida, T. Ak- iba, and H. Nakamura. Hard disk drive and command execution method, 2009. US Patent
7,477,477.

[13] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimal overhead. In IEEE conference on MAss Storage Systems and Technologies: Research Track, 2011.

[13] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimal overhead. In IEEE conference on MAss Storage Systems and Technologies: Research Track, 2011.

[14] S. Park and K. Shen. A performance eval- uation of scientific i/o workloads on flash- based ssds. In Workshop on the Interfaces for Scientific Data Storage, 2009.

[15] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim, and J. Lee. Cflru: a replacement algorithm for flash memory. In International conference on Compilers, architecture and synthesis for embedded systems, pages 234–241, 2006.

[16] Sangsoo Park and Heonshik Shin. Rigourous Modeling of Disk Performance for Real-Time Applications, volume 2986. Springer Berlin,
2004.

[17] H. Payer, M. A. A. Sanvido, Z. Z. Bandic, and C. M. Kirsch. Combo drive: Optimizing cost and performance in a heterogeneous storage device. In First Workshop on Integrating Solid-state Memory into the Storage Hierar- chy, volume 1, pages 1–8, 2009.

[18] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. 27(3):17–28, 1994.

[19] Y. K. Suh, B. Moon, A. Efrat, J. S. Kim, and S. W. Lee. Extent mapping scheme for flash memory devices. In MASCOTS, pages 331–
338. IEEE Computer Society, 2012.

[20] P. Triantafillou, S. Christodoulakis, and C. A.
Georgiadis. A Comprehensive Analytical Performance Model for Disk Devices Un- der Random Workloads. IEEE Transactions on Knowledge and data Engineering, 14(1),
2002.

[21] Y. Wang, K. Davis, X. Yuehai, and S. Jiang. iharmonizer: Improving the disk efficiency of i/o-intensive multithreaded codes. In Par- allel and Distributed Processing Symposium (IPDPS), pages 921–932, 2012.

[22] S. Zertal and P. G. Harrison. Multi-raid queue- ing model with zoned disks. In High Perfor- mance Computing and Simulation, 2007.

[23] S. Zertal and P. G. Harrison. Non-linear seek distance for optimal accuracy of zoned disks seek time in multi-raid storage systems. In High Performance Computing and Simulation,
2008.