International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7254-7258

Verification of Performance Improvement of Multi-plane Operation in SSDs

I. Shin
Associate Professor, Department of Electronic Engineering
Seoul National University of Science & Technology, Nowon-gu, Seoul, South Korea.

Orcid ID: 0000-0003-2819-0398

Abstract

Solid state drives (SSDs) provide significantly higher average performance than hard disk based storage by implementing various levels of parallelism internally. As a result, it quickly replaces hard disks in the server storage market. Parallel processing can be done in a channel level by simultaneously transmitting data through multiple channels, in a chip level by striping I/O requests to multiple chips on a page basis, in a die level by interleaving requests on multiple dies of a chip, and in a plane level by processing requests in multiple planes of a die at the same time. Although various policies have been proposed to maximize the parallel processing capability of SSDs, researches on the effect of the plane level parallel processing have not been sufficiently performed. This is because, unlike the channel level, chip level, and die level parallelism, in order to perform multi-plane operation, the types of operations must be the same, and the target pages of the planes must be located at the same position. Therefore, this study aims at verifying whether the multi-plane operation improves the overall performance of the SSD. Trace-driven simulation shows the followings. First, the wise policy that performs multi-plane operation only when the constraints of the multi-plane operation are satisfied always shows better performance than the policy that does not perform multi-plane operation. Second, the greed policy which performs the multi-plane operation with wasting clean pages even when the constraints of the multi-plane operation are not satisfied has excellent performance in the traces having the large requests. However, in the trace with a relatively small request size, its performance is lower than that of not performing multi-plane operation.

Keywords: Multi-plane operation, NAND flash memory, parallelism, SSD

INTRODUCTION

Solid state drives (SSDs) have a much higher IOPS than hard disks and are quickly replacing hard disks in high-end storage markets owing to their low energy consumption and heat generation. The reason why SSDs have much higher performance than hard disks is that it supports high-speed interface such as PCIe or Dimm and internally parallelizes I/O requests by connecting several NAND flash chips in parallel structure [1-6]. In other words, SSDs usually contain several NAND flash chips, which are connected to the NAND controller using multiple channels. Since each channel can transmit data at the same time, the parallel processing capability of the SSD increases in proportion to the number of channels. In addition, one chip is composed of multiple dies that can independently perform NAND operations such as read, write, and erase, and each die is composed of multiple planes.

The SSD's parallel processing capability can be utilized at various levels. First, the processing time of I/O requests can be reduced through chip level striping. That is, one large I/O request is split into multiple smaller requests on a page-by-page basis, and then these small requests are distributed over multiple chips. For example, if there is a write request for two pages, it is divided into two small write requests and these small requests are sent to two chips. If each chip is connected to a different channel, two small write requests are processed simultaneously. On the other hand, if the same channel is connected, the data of the sub request 1 is transmitted to the chip 1 first, and when it is completed, the write operation is initiated to the chip 1. At the same time, the data of the sub request 2 is transmitted to the channel, and when it is completed, a write operation is performed on the chip 2. As a result, even when the channel is shared, the NAND write time of sub request 1 can be hidden through the chip level striping.

The second way to exploit the parallelism is die interleaving [6]. Die interleaving means that multiple requests are distributed to multiple dies on the same chip. For example, if one chip consists of two dies, the write request 1 is first transmitted to the die 1 through the channel, and when the data transmission is completed, the NAND write operation for the die 1 is initiated. At the same time, the data of the write request 2 is transmitted to the die 2, and when the transfer is completed, the NAND write operation of the die 2 is executed. Therefore, the effect of die interleaving is the same as the effect of chip striping sharing channels.

The third way to exploit the parallelism is to use multi-plane operations. In general, a die consists of multiple planes, which cannot independently process NAND operations, but can perform the same kind of operations concurrently. The multi-plane operation refers to using the characteristics of the plane to execute the same operation simultaneously on the planes

belonging to one die. For example, if one die consists of two planes, the data of the write request 1 is first transmitted to the plane 1, and when the transmission is completed, the data of the write request 2 is transmitted to the plane 2 through the channel. When the transfer is completed, NAND write operations for planes are simultaneously initiated. The effect of shortening the I/O request processing time through the multi-plane operation is the same as the die interleaving.

Unlike die interleaving, there are some constraints on multi-plane operations [6]. First, the types of operations must be the same as described above. Second, the position of the target page and block must be the same on the target planes. For example, if the target page of the plane 1 is the 23rd page in the 35th block, the target page of the plane 2 must be page 23 in the same 35th block to perform the multi-plane operation. If the block number or the page number is different, multi-plane operation cannot be performed. Since the probability of the same target pages is low, the possibility of performing multi-plane operations is also low.

One way to utilize the multi-plane operation is to enforce the target pages to have the same location. For example, if the target page number in plane 2 differs from plane 1 in the above example, it is checked whether page 23 is currently clean in the same block 35 in plane 2, and if it is clean, the multi-plane operation is applied to this page. At this time, pages before page 23 in block 35 of plane 2 cannot be written even though they are clean. This is because NAND flash memory has a restriction that pages must be sequentially written in one block. Therefore, if page 23 is written, data cannot be written to page 0-22 even when they are clean. In other words, enforcing the multi-plane operations have the problem of wasting clean pages and causing more frequent garbage collections.

Previous study insisted that enforcing the multi-plane operation, despite its drawbacks, has a performance advantage [6]. However, there was a problem in the experiment setting, which set the number of clean pages of all blocks the same initially. Thus, the probability that the target pages of planes are the same is high. However, in reality, as the I/O request processing is accumulated, the number of clean pages in each block is likely to be distributed quite differently, and therefore the target pages of the planes are likely to be different also. In addition, the performance evaluation was done without applying the die interleaving. Therefore, in this study, we try to verify the effectiveness of the multi-plane operations by correcting the problems of the existing study.

First, the initial state is set so that all blocks are cleaned to reflect the actual SSD. It then measures performance by processing I/O requests. We processed the same trace multiple times to initiate the garbage collection and measure the sustained performance, because the traces have a short-term collection period and thus does not trigger the garbage collection. Also, we verified the effect of the multi-plane operation when applying the die interleaving.

The experimental results show the following facts. First, performing the multi-plane operation only when the constraints are satisfied always shows better performance than not performing the multi-plane operation. Second, enforcing the multi-plane operation shows an excellent performance in the traces having the large requests. However, in the trace with a relatively small request size, its performance is the lowest. Thus, it should be adopted considering the characteristics of the workloads.

BACKGROUND AND RELATED WORKS

NAND flash memory is an electrically erasable programmable read only memory consisting of blocks and pages. A page is a basic unit of read/write operation, and a block composed of several pages is a basic unit of erase operation. NAND flash memory has some limitations. First, you can write data only to clean pages. That is, once data is written to the clean page, new data cannot be written to the same page. In order to write new data, the block to which the page belongs is changed to a clean state through erase operation. This is called erase-before-write. Second, pages within a block must be sequentially written. For example, if a block consists of 64 pages, it should be written in the order of page 0 to page 63. If data is once written to page 20, pages 0 to 19 cannot be written even though they are clean. The subsequent data should be written sequentially starting from page 21. Third, each block has the limitation in the number that it can be erased. If this threshold is exceeded, the block becomes a bad block and can no longer hold data reliably. The maximum number of erase operations depends on the type of NAND.

Storage devices that use NAND as storage media use the flash translation layer (FTL) to hide these constraints of NAND. The FTL always searches for a new clean page and writes the data to it on a write request. Since the location of the data changes on every write, the FTL maintains a mapping table to store the current data location. According to the unit size of this mapping table, FTL is usually classified as page mapping [7], block mapping [8], hybrid mapping [9]. Since page mapping shows the best performance though memory consumption is high, SSDs generally use the page mapping FTL [2, 6], and this work also assumes that the target SSDs use the page mapping.

NAND flash memory has excellent read performance, but write performance is relatively low. Furthermore, since it does not support overwrite, it writes the data to the clean page every time, so the garbage collection must be performed to reproduce the clean page. Because garbage collection involves operations such as reading and writing of multiple pages, block erase, the garbage collection overhead worsens the write performance. SSD overcomes this low write performance problem through parallel processing. That is, a plurality of NAND chips are connected in parallel to the NAND controller through multiple channels. Each chip is composed of multiple

dies capable of performing independent NAND operations, and each die is composed of multiple planes. As described in the introduction, it reduces the processing time of requests by striping and interleaving large I/O requests to multiple chips and dies on a page basis. Similarly, multiple small I/O requests are distributed and processed simultaneously on multiple chips and dies to reduce latency and achieve high performance for each request.

In order to maximize the parallel processing capability of SSDs, it is important to distribute I/O requests evenly over idle channels, idle chips, and idle dies. For example, if the I/O request is concentrated on a particular die or on a particular chip, the parallel processing capability of the SSD will be limited. Early SSD models and related studies have mainly determined the chip and die to process the request statically based on the sector number of the I/O request [1-3]. However, this method has a problem that it can not fully utilize the parallel processing capability of the SSD. Thus, a method of determining a chip and a die in a round-robin manner in accordance with the order of a write request [10], a method for determining the shortest die as the target die [4], and a method for determining the target die in consideration of the current state of each die and channel [6] have been proposed. However, there is no study on the effect of the multi-plane operation on the parallelism of SSD except [6].

Meanwhile, it is difficult to conduct experiments using a commercial SSD product because internal firmware of the SSD must be modified in order to evaluate the parallel processing of the SSD. Therefore, simulators that model the parallel structure of SSDs have been developed [2, 6]. Among these simulators only SSDSim reflects constraints of the multi-plane operations [6]. But, there was a problem that the initial state of the SSD was erroneously set as having all the blocks have equal clean pages. In addition, the effect of applying both the multi-plane operation and the interleaving is not verified, and the completion time of the write request is erroneously set as the moment at which the data is transmitted to the target plane through the channel. The completion time should be set to the moment that the NAND operation is finished.

Verifying the Effect of Multi-plane Operation

In order to verify the effectiveness of the multi-plane operation, it is the most accurate method to modify the actual SSD internal firmware, but only a SSD manufacturer can use this method. Therefore, various SSD simulators implementing the internal structure of SSD have been proposed and used in various studies [1-6]. However, other simulators other than the SSDSim Simulator have a limitation that they do not accurately reflect constraints of the multi-plane operation. Therefore, this study evaluates the effect of the multi-plane operation using SSDSim simulator.

In the simulator, a policy which does not apply multi-plane, a wise multi-plane policy which applies multi-plane operation only when the constraint is satisfied, and a greed multi-plane which finds a target page having the same location even when the constraint is not satisfied are already implemented. However, when applying both the multi-plane operation mode and the die interleaving mode, the existing simulator code has the bug that it transmit multiple data simultaneously using a single channel, and the greed policy was wrongly implemented in part. Therefore, we modified them.

The physical structure of the SSD and the parameters related to NAND flash memory were used without modification. However, the overprovision of storage capacity was set at 20%, and the garbage collection was executed when the clean pages were less than 10% of the total pages. Table 1 shows the main parameter values related to the SSD physical structure and NAND flash memory. The decision of the target channel, the chip, the die, and the plane during the write request processing is dynamically determined in consideration of the state of the channel and the chip. That is, when there is an idle chip and idle chip, the write request is processed. In the chip, the target die and the target plane are determined in a round robin manner.

MSRC server traces (hm_0, prn_0) [11] and SPC server trace (financial2) [12] were used as the input traces. Table 2 shows the main characteristics of the traces. The financial2 trace shows that storage utilization is higher, the percentage of read requests is higher, the average size of requests is smaller and requests arrive more frequently than other traces. That is, MSRC traces show the workload pattern close to the backup server, and the financial2 trace shows the characteristics of the front-end server.

Table 1: SSDSim Parameters

Parameters	**Values**
Number of channels	2
Number of chips per channel	2
Number of dies per chip	2
Number of planes per die	2
Block size	128KB
Page size	2KB
NAND page read latency	20μs
NAND page write latency	200μs
NAND Block erase latency	1.5ms
One byte transfer time via channel	25ns
Overprovision	20%

Table 2: Trace Attributes

Trace	Storage utilization (%)	Read request ratio (%)	Avg. Read Req. Size (sectors)	Avg. Write Req. Size (sectors)	Avg. Inter-arrival time (ms)
hm_0	17.93	32.73	14.74	16.67	0.15
prn_0	22.46	22.21	45.67	23.31	0.11
financial2	74.86	78.46	4.56	5.84	0.01

Figures 1-3 show the results. The x-axis represents the number of repetitions of the simulation, and the y-axis represents the average response time of the requests at each iteration in µs. No multi-plane is a policy that does not perform multi-plane operation at all and Wise is a policy that performs multi-plane operation only when the constraints are satisfied. Finally, Greed is a policy that performs a multi-plane operation by matching the target block and page at the expense of wasting the clean pages even if the constraints are not satisfied. The results show that the performance of all policies decreases as the number of iterations increases. This is because the SSD is initially in a clean state, that is, all blocks are clean. Therefore, when the trace is repeated for the first time, garbage collection is not performed until the clean page is exhausted. However, once the clean page falls below 10% of the total and garbage collection is started, it is performed at a certain frequency afterwards. Therefore, when the iteration increases, the performance tends to be flat, which is the sustained performance of the SSD.

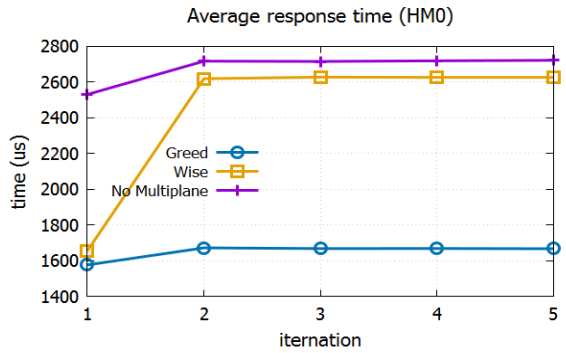

Figure 1: Average response time of I/O requests in HM0

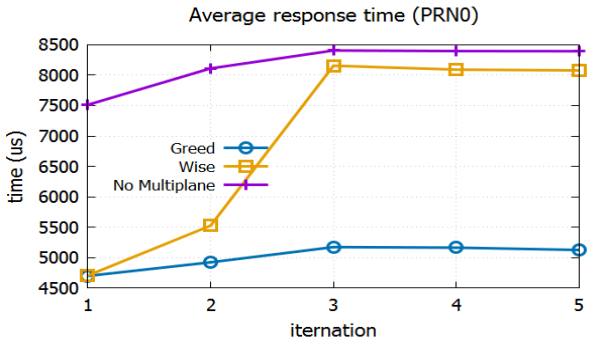

Figure 2: Average response time of I/O requests in PRN0

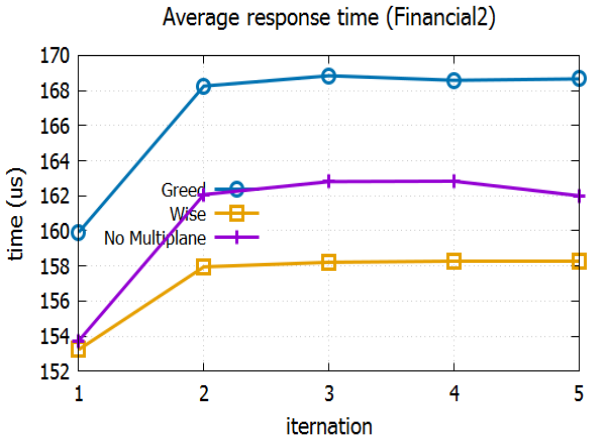

Figure 3: Average response time of I/O requests in Financial2

The common result for all traces is that the performance of wise multi-plane is always better than no multi-plane. Since wise multi-plane performs only when the constraints of the multi-plane operation are satisfied, there is no disadvantage compared to no multi-plane, and when the constraints are satisfied, it can improve the performance by performing multi-plane operation. The degree of performance improvement is different for each trace because it is related to the possibility of the constraints being satisfied.

The greedy policy is much higher than the wise method in MSRC traces. That is, even if the target pages are not positioned in the same location, it is much better to perform multi-plane operation wasting clean pages. However, the financial2 trace showed the opposite result. The greed policy has lower performance than no multi-plane policy. That is, the disadvantage of more frequent garbage collection becomes more apparent. This is because the average request size in the financial2 trace is much smaller than those of the MSRC traces, so the chip striping and the die interleaving sufficiently utilize the parallel processing capability of the SSD. Also, since in the financial2 traces I/O requests arrive more frequently, it is likely that the performance has been more degraded by frequent garbage collection execution.

In conclusion, the superiority of greed policy and wise policy is different according to the trace, and it is necessary to select the optimum policy according to the characteristic of the workload.

CONCLUSION

In this paper, we verified the effectiveness of multi-plane operation by simulating various server traces. As a result, the following results were obtained. First, the wise policy that performs multi-plane operation only when the constraints of the multi-plane operation is satisfied has always performed better than the policy that does not perform multi-plane operation. Second, the greed policy that performs multi-plane

operation with wasting clean pages even when the constraints of multi-plane operation are not satisfied is superior to the wise policy in traces with large request. However, the greed policy had a significant negative side effect on traces with relatively small requests. To support the conclusions of this paper, we plan to verify the effectiveness of multi-plane operations with more server traces in the future.

ACKNOWLEDGEMENTS

This study was supported by the Research Program funded by the Seoul National University of Science and Technology.

REFERENCES

[1] J. Y. Shin, Z. L. Xia, N. Y. Xu, R. Gao, X. F. Cai, S. Maeng, and F. H. Hsu "FTL design exploration in reconfigurable high-performance SSD for server applications," ACM In Proceedings of the 23rd international conference on Supercomputing, pp. 338–349, June 2009.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy "Design tradeoffs for SSD performance," USENIX Annual Technical Conference, pp. 57-70, June 2008.

[3] S. Ruan, M. Alghamdi, X. Jiang, Z. Zong, Y. Tian, and X. Qin "Improving write performance by enhancing internal parallelism of Solid State Drives," IEEE 31st Int. Performance Computing and Communications Conference, pp. 266–274, December 2012.

[4] C. Park, E. Seo, J. Shin, S. Maeng, and J. Lee "Exploiting internal parallelism of flash-based SSDs," IEEE Computer Architecture Letters, vol. 9, no. 1, pp. 9–12, 2010.

[5] Y. A. Winata, K. Sanghoon, and I. Shin "Enhancing internal parallelism of solid-state drives while balancing write loads across dies," Electronics Letters, vol. 51, no. 24, pp. 1978-1980, November 2015.

[6] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in Proc. ICS, pp. 96-107, 2011.

[7] A. Ban "Flash file system," United States Patent. No. 5,404,485, April 1995.

[8] A. Ban, "Flash file system optimized for page-mode flash technologies", United States Patent, no. 5,937,425, 1999.

[9] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems", IEEE Trans. Consumer Electron., vol. 48, no. 2, pp. 366–375, 2002.

[10] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," in Proc. IEEE HPCA, pp. 266–277, 2011.

[11] Microsoft Research Center, "MSRC I/O traces," ftp://ftp.research.microsoft.com/pub/austind/MSRC-io-traces

[12] Storage Performance, "SPC I/O traces," http://skuld.cs.umass.edu/traces/storage/Financial1.spc.bz2