International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 15 (2017) pp. 4825-4829
© Research India Publications. http://www.ripublication.com

State-based Die Binding for Enhancing SSD Internal Parallelism

S. Jin[1] and I. Shin[2,*]

Graduate Student[1], Associate Professor[2]
Department of Electronic Engineering, Seoul National University of Science & Technology
Nowon-gu, Seoul, South Korea.
**Corresponding author's*

Abstract

Solid state drives (SSDs) implement large capacity, high performance storage devices by connecting multiple NAND flash memory chips in parallel using multiple channels. Channels can transfer data simultaneously, and each NAND package is composed of multiple dies, which can independently perform NAND operations such as read, write, and erase. Therefore, maximizing the parallel processing capability inside the SSD is important for performance improvement. However, the existing logical address based die binding policy has a disadvantage in that it can concentrate requests only on specific dies, and the existing dynamic binding policy that considers the state of each die has a limitation that it can be used only for the SSD where each die is connected with a dedicated channel. Therefore, in this paper, we propose a dynamic die binding policy that considers both channel state and die state for enhancing internal parallelism of general SSDs where multiple dies share a channel. The performance evaluation results show that the proposed policy shows a performance improvement of more than 20% over the logical address based static binding policy.

Keywords: Die binding, NAND flash memory, parallelism, SSD

INTRODUCTION

Solid state drives (SSDs) are rapidly replacing hard disks in the high-end server storage market as well as in the mobile storage market with its high performance, low power consumption, low heat generation and support for various form factors. SSDs use NAND flash memory as its internal storage medium, and the advantages and limitations of SSDs inherit mainly the physical properties of NAND flash memory. The storage capacity of NAND flash memory is significantly lower than that of the hard disk and the writing speed is much slower than the reading speed. In particular, there is a performance overhead in that overwriting is not supported and garbage collection must be performed as write operations accumulate. However, SSDs have overcome the performance and storage capacity constraints of NAND flash memory by connecting several NAND flash memory packages with parallel channels [1-5]. In other words, SSDs include several channels that can transfer data at the same time, and connects multiple NAND flash packages to each channel. Each NAND flash package is composed of several dies, and each die is again composed of several NAND planes. Because the die can independently perform NAND operations, SSDs can improve overall performance by splitting one large I/O request into several sub requests and striping them to multiple die [1-5].

In order to effectively utilize this parallel processing capability, I/O requests must be efficiently distributed to multiple dies [5]. For example, if the I/O operations are concentrated on several dies, the parallel capability of the SSD cannot be used as a full extent. However, many SSDs currently use a sector address based die binding policy that determines the target die by the sector address [1-3]. In this method, I/O operations can be concentrated on specific sectors, and thus some dies are busy with processing I/O requests while the other dies remain idle.

In order to solve this problem, a method of transmitting an I/O request to an idle die considering the current states of dies has been proposed [5]. However, this method has a limitation in that it is targeted to an SSD where each die is connected with a dedicated channel. However, using dedicated channels increase the cost of manufacturing SSDs, and thus most commercial SSDs have a structure in which multiple dies share channels [1-4].

Therefore, in this study, we propose a policy that determines the target tie considering the status information of both channels and dies in the SSDs where multiple dies share the channel. At the same time, the accumulated writes to dies are also considered, so that dies are worn out in a balanced way and the stability of SSDs is improved.

Simulation results using realistic workloads show that the proposed policy improves the average response time by more than 20% compared to the address based die binding method.

BACKGROUND AND RELATED WORKS

NAND flash memory is an electrically erasable programmable read only memory that consists of blocks and pages. A block is an erase unit and has multiple pages. A page is a read/write unit. NAND flash memory does not support an overwrite operation. Once a page is written, it cannot be overwritten before erased. Thus, NAND-based block devices such as SSDs perform the out-of-place update through flash translation layer (FTL). In the out-of-place update, locations of valid data are changed on every write. This implies that the target die for write requests can be dynamically determined regardless of the sector address.

As the location of data becomes different on every write, FTL maintains the mapping table between a logical address and its current physical address. According to the mapping granularity, FTL is classified to page mapping [6], block mapping [7], and hybrid mapping [8]. The page mapping scheme uses a NAND page as a mapping unit and delivers a good performance. Thus, SSDs generally employ the page mapping scheme, and this work also assumes that the target SSDs use the page mapping scheme.

The internal structure of a SSD is illustrated in fig. 1. The internal processor executes FTL, and the mapping table is maintained in RAM. For large capacity and high performance, multiple NAND packages are connected to NAND controllers via multiple channels that can transfer data at the same time. A NAND package consists of one or multiple dies that can process NAND operations independently (fig. 2). On an arrival of a request, FTL decides to which dies to send the request. For high performance, how to fully exploit the parallel processing capability should be considered when deciding the target dies.

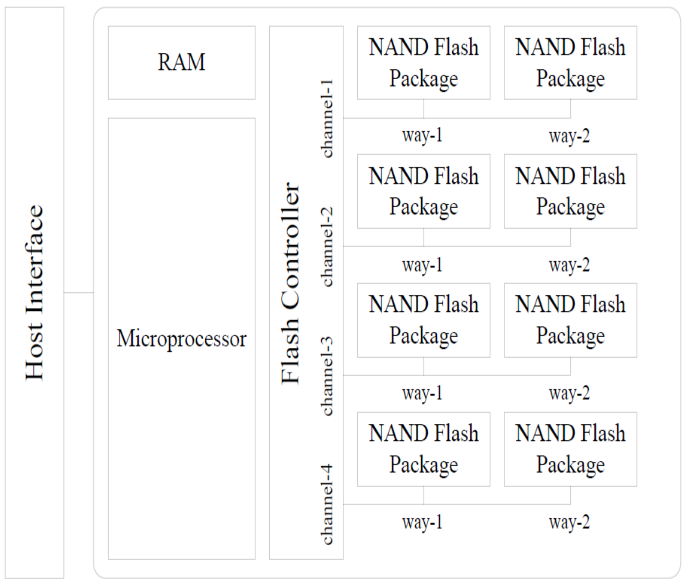

Figure 1. Internal structure of SSDs

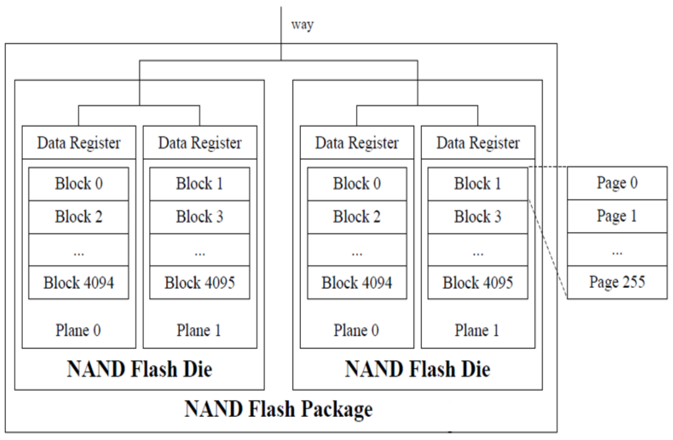

Figure 2. Internal structure of NAND flash package

A simple way to exploit the parallelism of SSDs is to split a large write request into multiple sub requests on a page-by-page basis and stripe the sub requests across multiple dies. By dividing large requests into sub requests of page units and processing them simultaneously, the response time of large write requests is greatly reduced. Therefore, to maximize parallelism, we should choose dies that can handle the requests immediately when determining the target die to process the requests. If a request is sent to a die that is busy in processing other request, the request will not be processed immediately and wait until the die becomes ready.

However, the logical address based target die binding method [1-3] that are widely used in commercial SSDs does not consider this state information. Only the logical address is used to determine the target die. For example, a logical page number is calculated from a sector number, and the target die is determines by applying the modular operation. This causes the write requests to be concentrated on a particular area, and the write requests will not spread evenly across all the dies, but only on particular dies. In other words, some dies are not utilized even though they are idle.

In order to solve the problem of the address based binding, some SSDs determine the target die based on write order [9]. In this scheme, the first write request is sent to the first die and the second write request is sent to the second die. In other words, the target die is determined in a round robin manner, and thus there is an advantage that all dies process write requests equally. However, because read requests are not evenly distributed, there is still the problem that read requests are concentrated on particular dies, which limits the parallel processing.

The queue based die biding policy [4] maintains a separate request queue for each die and sends an I/O request to the die having the shortest queue length. This scheme maximizes parallel processing power by distributing I/O requests evenly across all dies in the short term. However, in the long term, there is a problem that the stability of the die is degraded because the write requests can accumulate on specific dies [5].

The state based die binding method determines the target die in consideration of the state of the dies [5]. In other words, when transmitting requests, idles dies are preferentially chosen. If the number of idle dies is more than required, the request is sent to the die with less cumulative writes. Its limitation is that it assumes that all dies are connected to the NAND controller through a dedicated channel. However, it is common for commercial SSDs to share a single channel across multiple die because of cost issues. In the environment where a channel is shared, if the channel is busy transmitting data, the new request cannot be sent to a die connected to the corresponding channel. Therefore, the status information of the channel should also be considered. In this paper, we propose a new die binding method that determines the target die by considering the states of the channels as well as the states of the dies.

Meanwhile, in the SSDsim simulator [10], a similar die binding method was implemented. This method also considers the states of the channels and the dies. However, the algorithm of the policy is not clearly described in the paper that proposes the SSDsim [10].

PROPOSED SCHEME

In SSD, a write request is processed as follows. First, the write request is split into several sub write requests per page. For each sub request, the target die is determined, and we transmit the data to be written to the target die through the channel. Then, a write command is sent to the target die and NAND write operation is performed in the target die.

After determining the target die, if the channel associated with the target die is busy with another data transfer, the request cannot be processed immediately. Also, if the target die is busy with performing other operations, the request should wait until the die becomes ready.

Therefore, the proposed policy determines the target die by considering both channel state and die state. For example, if a request to be processed needs to write to three pages, it is divided into three sub write requests that will be sent to three target dies. Then, we gather the status information of channels and dies to find the dies where both the channel and the die are idle. If the number of dies satisfying these conditions is equal to the number of sub write requests, the requests are instantly sent to them. If the number of dies satisfying the condition is larger than the number of sub write requests, the dies having a small cumulative writes are chosen. This is to improve the stability of the SSD by maintaining uniform wear-out of each die. Finally, if the number of dies satisfying the condition is smaller than the number of sub write requests, the same number of sub write requests are transmitted to the dies satisfying the condition. The remaining unsent sub write requests are inserted into the undecided queue in FCFS order. These requests will be processed later in sequence when a die satisfies the condition. In other words, if an event occurs in which a channel becomes idle or a die becomes idle, it is checked whether a die satisfying the condition that both the channel and the die are idle exists. If yes, the sub write request added first to the undecided queue is transmitted to the die.

PERFORMANCE EVALUATION

In order to evaluate the performance of the proposed policy, simulation using real workload was performed. Two representative groups of server workloads were used. The first group is the five server workloads collected by the MSRC [11] and the finantial1 trace was downloaded from the SPC [12]. The attribute information of each trace is shown in Table 1.

Table 1. Trace Attributes

Group	Trace	# of Request	Write Ratio (%)
MSRC	hm_0	151	65
	src2_0	449	89
	ts_0	388	82
	wdev_0	529	80
	web_0	298	70
SPC	financial1	8	77

The SSDsim simulator that was proposed in the previous study [10] was used for performance evaluation. This simulator can configure multi-channel and multi-way SSDs, and it already implements the address based die binding policy and a dynamic binding policy using state information of channel and die which is similar to the proposed policy. We additionally implemented the write order based binding, queue based binding that choose the die with the shortest queue length, and the proposed binding policy.

The target SSDs are configured in two way. SSD1 is configured to four channel and two way structure. In other words, there are four parallel channels and two dies share a single channel. SSD2 is configured to four channel and four way structure. In other words, there are four parallel channels and four dies share a single channel. In each SSD, advanced commands such as multi-plane and interleaving operations were turned off, and other parameters of the simulator were not modified.

Figures 3 and 4 show the evaluation results. Fig. 3 shows the result of SSD1 and Fig. 4 shows the result of SSD2. In order to clarify the performance comparison, the relative response time of each policy is shown by scaling the average response time of the address based binding policy to 1. In the figures, static refers to the address based binding, and queue refers to the queue based policy that chooses the shortest queue length. Dynamic is the dynamic binding policy implemented in the simulator. Finally, UQ refers to the proposed policy.

Fig. 3 shows that the performance of the address based binding policy is the lowest, as expected. The average response time of all other policies is less than one. In overall, the queue based policy has a performance somewhat worse than the write order binding. In particular, the write order method shows better performance than the queue based binding in ts_0, src2_0, and financial1. Meanwhile, the proposed method and the dynamic policy implemented in the simulator shows the similar best performance. In overall, the response time is improved by more than 20% compared to the address based binding policy. The dynamic delivers a better performance in the financial1 than UQ by about 10%.

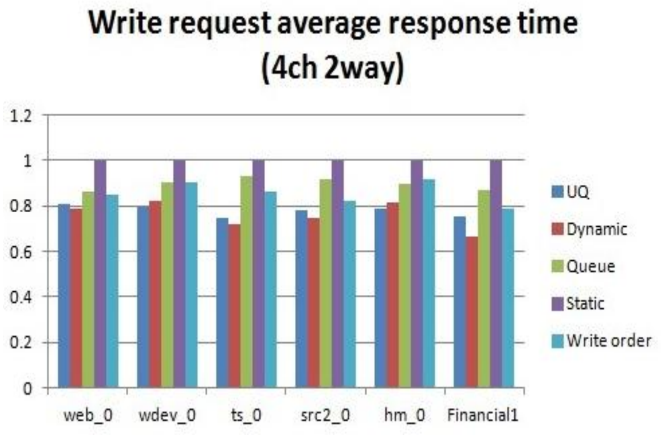

Figure 3: Average response time of write requests in SSD1 (four channel and two way)

The result of SSD2 shown in fig. 4 is similar. The address based binding policy shows the lowest performance. The queue based policy and the write order policy showed similar performance. The difference between the results of SSD1 and SSD2 is that the proposed UQ performance is higher than the dynamic policy. In particular, performance differences in the financial1 traces is noticeable, reaching about 14%. In ts_0, src2_0, and hm_0, UQ shows better performance than the dynamic policy.

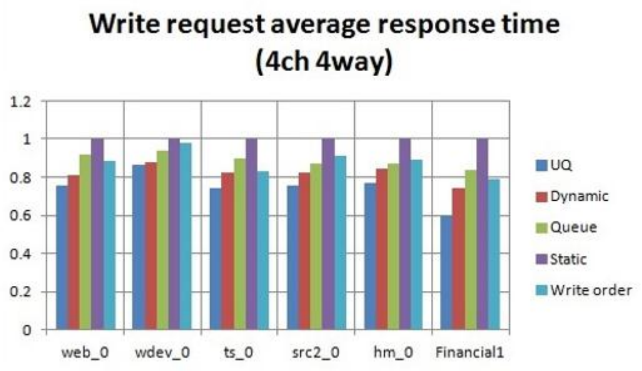

Figure 4: Average response time of write requests in SSD1 (four channel and four way)

CONCLUSION

This study proposed the dynamic die binding policy to maximize the parallel processing capability of SSD. The proposed policy considers both the state of each die and the state of the channel connected to the die, and selects the die that can process the request immediately as the target die. If the number of possible dies is larger than the required number of dies, the target die is selected considering the cumulative amount of writing of each die. Conversely, if the number of possible dies is less than the required, then the same number of sub requests with the ready dies are sent to the ready dies, and the remaining unprocessed sub requests are inserted into the undecided queue. The requests inserted into the undecided queue are processed in sequence each time the die and channel become ready. The performance evaluation results showed that the proposed policy improves the performance by more than 20% over the address based static binding policy. Compared to other existing dynamic binding policies, the proposed one delivered the better performance.

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

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

[8] 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.

[9] 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.

[10] 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.

[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