A RAM cache approach using Host Memory Buffer of the NVMe interface

JuHyung Hong, SangWoo Han and Eui-Young Chung
Department of Electrical and Electronic Engineering
Yonsei University
Seoul, Korea
{jh.hong, swhan0330}@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

Abstract— **This paper proposes new methods with Host Memory Buffer to improve IO performance in NVMe interface. Although Host Memory Buffer is a versatile memory architecture, it has been considered limitedly as metadata cache such as Logical-to-Physical table. The proposed architecture uses Host Memory Buffer as data cache with modification of an NVMe command process and the additional DMA path between system memory and Host Memory Buffer. The proposed architecture improves the performance of IO request by 23% in case of sequential writes compared to a device buffer architecture.**

Keywords; NVMe, Host Memory Buffer (HMB), RAM cache

I. INTRODUCTION

NAND Flash Memory (NFM)-based storage has become the mainstream with many innovation techniques despite of some drawbacks such as high cost per bit, limited lifecycle and reliability. Especially, the appearance of host interface suitable to NFM stimulates the growth of NFM-based storage. The interface of Solid State Drive (SSD) for client or database computing is changing from SATA based on AHCI or SAS to NVMe over PCIe [1]. In addition, Universal Flash Storage (UFS) is emerging for consumer devices. NVMe provides parallel operation by supporting each up to 64K commands within 64K queues and reduces latency with efficient command protocols. The maximum performance of NVMe is limited by physical IO bandwidth of the PCIe. Nowadays, Intel Skylake has PCIe 3.0 4 lanes in Platform Controller Hub (PCH) chipset and its unidirectional bandwidth is 3.91GB/s.

RAM Disk is a software which takes exclusively a portion of system memory and shows it to user as a drive. Therefore, the performance is an order of magnitude faster than a storage device. RAPID mode of SAMSUNG SSD eliminates IO bottlenecks by using main memory as a write/read cache [2]. This mode is used primarily to accelerate write performance or to decrease queue depth 1 latency. Though the cache-based acceleration improves the completion time for IO transactions, RAPID mode is available by setup of software and a kernel driver of a host and has to manage the large buffer. In addition, this mode only takes effects in one SSD installed OS.

II. HOST MEMORY BUFFER

In specification of NVMe 1.2, Host Memory Buffer (HMB) enables using main memory as the device working buffer memory. The device can use HMB exclusively like belonging to the device logically whatever an application is. However, it has been considered merely as L2P table [3]. This is because of a Remote DMA (RDMA) function for the NVMe device and high bandwidth of PCIe. The host submits command into a Submission Queue (SQ) in Host memory. The physical memory address used for data transfers is located in Physical Region Page (PRP) entries of SQ. As the NVMe device fetches SQ, PCIe DMA transfers data between the host and the device through Transaction Layer Packets (TLPs). It enables a zero-copy operation by handing over only the address of data. If we exploit HMB to apply RAM cache, the additional copy operation is required. The bandwidth of PCIe offers higher bandwidth and lower latency compared to a SATA interface. However, the bandwidth of 2.4GHz 20 lanes QPI system bus is 19.2GB/s, which is twice or more higher than PCIe 3.0 4 lanes. Therefore, it is plausible to apply HMB as RAM cache.

UFS extents its specification to use Host memory as if it would be in the device [4]. Unified Memory (UM) Extention adds new commands to transfer data between UM and the device. New commands can copy either from Host memory to UM or within UM. However, NVMe does not support schemes like UM commands. This paper proposes new methods which exploit HMB as RAM cache to improve performance in NVMe.

III. THE PROPOSED METHODS

A. The Proposed NVMe command process

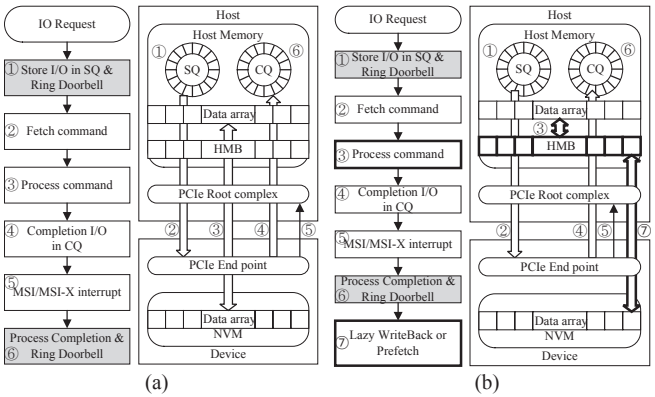


Figure 1. NVMe command process. (a) the conventional command process, (b) the proposed command process

Fig. 1 (a) shows the NVMe command process. First, the host stores command entries in SQ and writes the updated SQ

tail pointer to doorbell ①. The device fetches commands and processes read/write commands ②-③. And the device writes completion to Completion Queue (CQ) and sends interrupt to the host ④-⑤. Lastly, the host processes completion ⑥. Fig. 1 (b) represents the modified command process to apply HMB as RAM cache. When the device processes read/write commands. The device does not transfer data between Host memory and the device but between Host memory and HMB ③. When HMB is full or the device needs to store data into permanent storage, the device tries to do lazy write-back from HMB ⑦.

B. The proposed architecture for RAM cache

We leverage a DMA controller of Host to exploit the new path between system memory and HMB as depicted in Fig. 2. Once the host notifies the device of the specific memory mapped addresses through a register level interface of NVMe, the device can access those addresses. When the host sends Set Features command to enable HMB feature, the host let the device know the specific function register of DMA controller additionally. Finally, the device can run DMA and check the status of DMA controller inside the host.

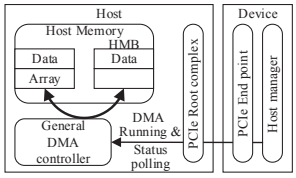


Figure 2. The proposed architecture for RAM cache using HMB

IV. EXPERIMENTS

A. Experimental setup

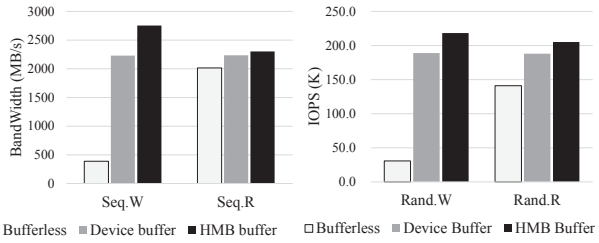
We used QEMU x86_64 virtual machine to implement the modified NVMe interface because we cannot recognize the memory mapped address of a real host system. We modified the VSSIM/eVSSIM simulator to evaluate the performance of the proposed architecture with timing parameters of NFM and the access latency of Host memory through NVMe Interface [5]. The simulator used an 8 channel 4 way NFM array of which parameters are specified by values such as Table 1.

TABLE I. SPECIFICATIONS OF SIMULATOR

Parameters	Values	Parameters	Values
Page size	8192B	NFM Read latency	40us
Pages per a block	256	NFM Write latency	900us
Number of Blocks	256	NVMe W/R latency	2.8us
Number of Planes	2	NAND CH bandwidth	400MB/s
HMB Write buffer	64MB	PCIe bandwidth	3.91GB/s
HMB Read buffer	512KB	Host DMA bandwidth	4.8GB/s

B. Experimental results

We run FIO to generate synthetic IO events into Linux 3.13.0 kernel on Ubuntu 12.04. FIO is one of the most popular benchmark tools for IO performance measurement. We compare the HMB buffer architecture to both the device embedded buffer and the device without buffer. HMB improves the bandwidth up to 6.0x and 23% respectively compared to Bufferless and the device buffer in sequential writes as shown in Fig.3 (a). There are the performance improvement up to 6.1x and 15% corresponding in random writes as shown in Fig. 3 (b). Write performance shows remarkable enhancement than read cases because of eliminating long write latency of NFM and the low hit ratio within buffer in a read operation.



(a) (b)

Figure 3. Performance comparision. (a) Average Bandwidth in Seq.W/R, (b) Average IOPS in Rand W/R

The limitation of HMB buffer is that the bottleneck is changed to bandwidth of NFM after the buffer overflow as shown in Fig. 4. Therefore, the NVMe device should expand the parallelism of NFM or the host manager of the device should handle HMB buffer entries efficiently.

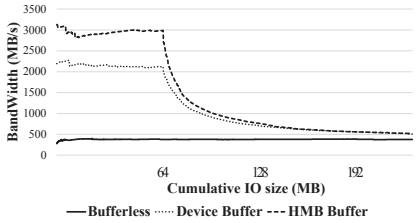


Figure 4. Buffer overflow impact on cumulative IO size

V. CONCLUSION

We exploit HMB as RAM cache to overcome the bandwidth limitation of NVMe over PCIe. We leverage the Host DMA controller which is controlled by the device. We implement the proposed architecture into QEMU and modify the released SSD simulator to evaluate the improvement. Experiments show that the average bandwidth with HMB buffer is increased up to 12.8% compared to device buffer.

ACKNOWLEDGMENT

This work was supported by the ICT R&D program of MSIP/IITP 2016(R7177-16-0233) and by Samsung Electronics.

REFERENCES

[1] "NVMe 1.2 Specifications," http://www.nvmexpress.org/specifications/.

[2] "RAPID mode white paper," http://www.samsung.com/semiconductor /minisite/ssd/download/consumer.html.

[3] J. G. Hahn, E. Erez, S. A. Jean, G. B. Desai and V. K. Nadh, "Methods systems and computer readable media for proving flexible host memory buffer," U.S. Patent US20160026406, Jul. 30, 2015.

[4] "UFS UNIFIED MEMORY EXTENTION,Version 1.0," https:// www.jedec.org/standards-documents/results /jesd220-1/.

[5] J. Yoo, Y. Won, J. Hwang, S. Kang, J. Choi, S. Yoon, and J. Cha, "VSSIM: Virtual machine based SSD simulator," in IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2013, pp. 1–14.