Architecture Exploration of High-Performance PCs with a Solid-State Disk

Dong Kim, Kwanhu Bang, *Student Member*, *IEEE*, Seung-Hwan Ha, Sungroh Yoon, *Member*, *IEEE*, and Eui-Young Chung, *Member*, *IEEE*

Abstract—As the cost per bit of NAND flash memory devices rapidly decreases, NAND-flash-based Solid-State Disks (SSDs) are replacing Hard Disk Drives (HDDs) used in a wide spectrum of consumer computing devices. Although typical SSDs can deliver higher performances than HDDs can, the full capabilities of SSDs are currently not exploited in most systems. This is because an SSD is interfaced with its host system using the architectures and interface protocols designed for HDDs, due to compatibility issues. Given the pace at which the stand-alone performance of SSDs improves, the performance loss of SSDs due to the legacy interface and system architecture will soon become intolerable. To address this issue, we propose several architectural choices to fully exploit the performance of SSDs used in consumer PC architectures. More specifically, we explore its interface scheme, and data transfer concurrency with the change of the conventional PC architecture if necessary. We evaluated the performance of the architectural choices by prototyping them with SystemC. The experimental results guide us how to trade off the performance enhancement and the change of the PC architecture. The performance improvement was maximized by 2.67 times when the PC architecture is changed to support a dual-port SSD connected to the North Bridge via the Double-Data Rate (DDR) interface in real trace environments.

Index Terms—Solid-State Disk (SSD), NAND flash memory, dual-port DRAM, North Bridge, direct path.

1 Introduction

Hard Disk Drives (HDDs) have been widely used as mass storage devices for the last several decades in many electronic devices including PCs. Recently, a breakthrough in technology introduced a light and small electronic device called a Solid-State Disk (SSD) as an alternative mass storage device. An SSD consists of nonvolatile memory elements (i.e., NAND flash memory) and a controller that is in charge of data management and communication with the host machine. SSDs have several superior properties such as small form factor, light weight, low power consumption, and shock resistance compared to HDDs which comprise of platters, heads, spindle motors, and other mechanical moving parts. SSDs are thus more advantageous for harsh and rugged environments, such as in military and aerospace applications, since SSDs are fully electronically operated. Nevertheless, the biggest obstacle to deploy SSDs widely is the higher bit cost than that of HDDs. However, as the density of NAND flash memory has doubled, on average, every 12 month, it is expected that SSD bit cost will rapidly decrease and that SSDs will thus lead the massive storage market in the near future. According to International Data Corporation (IDC), the price gap between 128 GB SSD and the 120 GB HDD will keep narrowing, and eventually become negligible in 2012 [27].

Such cost reduction will accelerate the adoption of SSDs in many consumer electronics devices when the benefits of using SSDs are maximized. Among the various benefits of SSDs, the performance factor is currently receiving the largest attention, since the IO performance gap has become severe[1] [26]. Hence, slow second mass storages have become a major performance bottleneck of high-performance computing machines. The gap will become even severer in the near future due to the appearance of multicore- and many-core-based parallel computers.

Especially, we select PCs as our target architecture, since PCs account for the largest share in the mass storage device market. For enhancing the conventional PC architecture, we explore the PC architecture in two aspects—host interface scheme and data transfer concurrency. Also, we can define the host interface scheme by two parameters—location and interface protocol.

First, the location implies the place to which an SSD is connected. In the conventional architecture, an HDD is connected to a chipset called South Bridge that is designed for low-speed peripherals. However, SSDs are usually much faster than HDDs, hence, the South Bridge may not be the best place to attach an SSD. Second, the interface protocol means a communication protocol between the host machine (PC, in our case) and the SSD used. Currently available SSDs inherit the interface protocols for HDDs, such as Parallel Advanced Technology Attachment (PATA) and Serial ATA (SATA). PATA or SATA may provide a sufficient bandwidth to HDDs, but this is questionable for SSDs, since they are

• D. Kim is with Flash Solution R&D Center, Samsung Electronics Co., (Hwasung Plant) San #16 Banwol-Dong, Hwasung-City, Gyenggi-Do 445-701, Korea. E-mail: dong.09.kim@samsung.com.
• K. Bang and E.-Y. Chung are with the School of Electrical and Electronic Engineering, Yonsei University, 134 SinChon-Dong, Seodaemun-Gu, Seoul 120-749, Korea.
E-mail: khbang@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr.
• S.-H. Ha is with Flash Software Development Project, Samsung Electronics Co., Hwasung Plant, San #16 Banwol-Dong, Hwasung-City, Gyenggi-Do 445-701, Korea. E-mail: seunghwan.ha@samsung.com
• S. Yoon is with the School of Electrical Engineering, Korea University, Engineering Building #207, Anamdong, Seongbukgu, Seoul 136-713, Korea. E-mail: sryoon@korea.ac.kr.

Manuscript received 15 Jan. 2009; revised 1 June 2009; accepted 19 Aug. 2009; published online 18 Mar. 2010.
Recommended for acceptance by R. Gupta.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-01-0024.
Digital Object Identifier no. 10.1109/TC.2010.66.

1. According to Intel's measurement in 2006, CPU operating frequency has been improved by 30 times for the last decade, while HDD latency has been enhanced by 1.3 times for the same period.

typically much faster than HDDs. Added to this, finally, we consider the data transfer concurrency to minimize the conflicts between CPU-to-memory accesses and memory-to-SSD accesses, which can critically degrade the system performance. For this purpose, we propose a PC architecture that uses dual-port DRAM. In this architecture, one of the two DRAM ports is directly connected to the SSD which also has two ports.

Among above-mentioned aspects, we have addressed a PC architecture exploring the host interface scheme (location and interface protocol) in [1]. More specifically, we proposed a PC architecture equipped with an SSD connected to North Bridge rather than South Bridge through DDR DRAM interface in [1]. Even though it is the first work to address the PC architecture exploration for fully exploiting SSD performance, it lacks some other aspects such as concurrency. In this paper, we extend our previous work by considering the concurrency in PC architecture exploration. Furthermore, we analyze not only the impact of individual enhancements by each of the above three factors, but also their combined effects.

The remainder of this paper is organized as follows: In Section 2, we summarize the previous works related to SSDs from architecture perspective. In Section 3, we present the internal architecture of SSDs and the conventional PC architecture that uses SSDs as preliminaries. Sections 4 and 5 address the details of proposed three different PC architectures with qualitative analysis. Finally, we show our experimental results in Section 6, followed by conclusion in Section 7.

2 Related Works

Many of architectural works for SSDs are mainly related to the internal architecture for improving the performance of an SSD itself and only a few works studied the interaction between the system and an SSD to maximally exploit the SSD. In this section, we first briefly review the work related to the SSD internal architecture, and then summarize the works from the host architecture perspective.

One of major approaches to increase the performance of an SSD is increasing the parallelism. In [2], Chang and Kuo proposed an adaptive striping architecture to introduce I/O parallelism by using a hot-cold identification mechanism. In [7], Kang et al. proposed a multichannel SSD controller to increase I/O parallelism and exploited the parallelism by use of traditional throughput increasing techniques such as striping, interleaving, and pipelining. Similarly, multichannel and way interleaving schemes were proposed for increasing data throughput in [6]. Kim et al. presented an architecture exploration methodology for NAND flash storage to explore a wide design space including channel numbers using a QEA algorithm in [10]. Also, in [5], Min and Nam introduced an improved NAND flash controller architecture that places a data path between the host and NAND flash memory interfaces which enables the concurrent execution of data transfer and control operations. The *HyperLink NAND Flash Architecture* is proposed in [9], which has ring topology interface and multiple independent banks for high-performance SSD.

On the other hand, some other works focused on exploiting memory hierarchy. Lee et al. [4] proposed a new NAND flash memory package with a smart buffer cache that enhances the exploitation of spatial and temporal localities to achieve high performance and low power

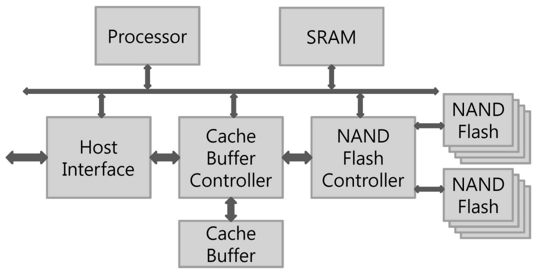


Fig. 1. Internal architecture of SSD.

consumption. Also, an energy-aware demand paging method was presented in [3] to minimize the number of write or erase operations.

In addition, some approaches proposed to adopt hybrid nonvolatile memory array to trade off the performance and cost. Chang et al. proposed an SSD architecture which combines multilevel cell (MLC) flash and single-level cell (SLC) flash [8]. Yoon et al. proposed an SSD architecture which consists of FRAM and NAND flash, where metadata is maintained in a small FRAM to provide high performance [11]. In [12], Kim et al. proposed a hybrid storage architecture in which memory array is the combination of PRAM and NAND flash to increase its performance and lifetime.

While the aforementioned studies focused on the internal SSD architecture itself, there has been less attention to the interaction between SSD and its host machine, which may be crucial for enhancing the overall system performance. Only a few works addressed the overall system performance when an SSD is employed. One well-known approach is *Robson Architecture*, where a nonvolatile memory (NAND flash) layer is introduced as a cache of SSDs or HDDs to reduce data transfer time and power consumption [13]. Also, a high-performance SSD has been released in the market [14]. The SSD adopts PCI-Express interface which provides much higher bandwidth compared to the traditional slow interfaces such as PATA and SATA, hence, it resolves the performance bottleneck issue due to the slow host interface. Recently, several works focused on the high-performance server systems. In [15], Lee et al. showed the case study of NAND-flash-based SSD in enterprise server systems. Kgil et al. [16] presented NAND-flash-based disk cache architecture as well as programmable NAND flash controller in server platforms. In [17], several hundreds of NAND-flash-based storage nodes were plugged into an ethernet-style backplane to build clusters for data-intensive applications.

To the best of the authors' knowledge, none of previous work addressed the performance issue of PC architecture with an SSD except our previous work in [1]. The work in this paper further extends to achieve higher performance in PCs by exploring more architectural choices.

3 Preliminaries

In this section, we describe the internal architecture of a typical SSD unit and the conventional PC architecture that uses SSDs. Although both architectures presented here can differ depending on each vendor's design, the basic architectures do not differ much from one to another.

3.1 Internal Architecture of SSD

A typical SSD internal architecture is shown in Fig. 1. A processor manages the overall system behavior and the

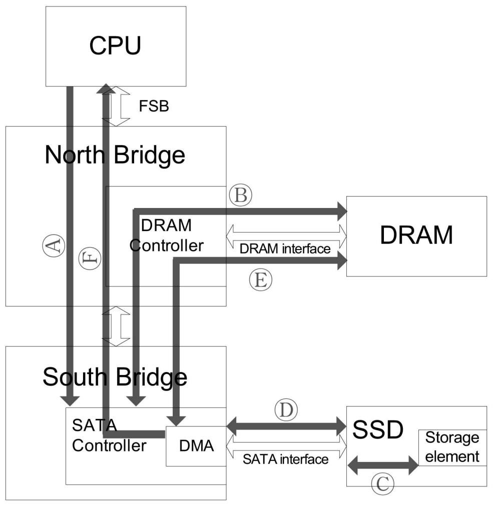


Fig. 2. Conventional PC architecture with SSD.

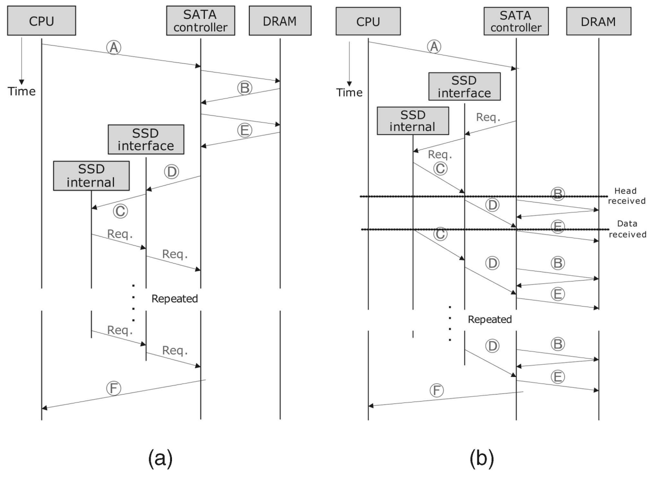


Fig. 3. DMA transfers of conventional PC architecture. (a) Conventional DMA WRITE (consecutive). (b) Conventional DMA READ (pipelined).

TABLE 1
DMA Transfer Time

	Operation	Description
Ⓐ	DMA command	CPU issues a DMA command
Ⓑ	PRD fetch	Read the physical memory region
Ⓒ	SSD internal	SSD internal read/write transfer
Ⓓ	SSD interface	SSD interface read/write transfer
Ⓔ	Memory access	Memory access to or from SSD
Ⓕ	Interrupt transfer	Interrupt generation after DMA transfer completion

software called Flash Translation Layer (FTL) runs on it for wear-leveling. By wear-leveling, a logical address in the host machine is translated into a physical address in NAND flash memory. This is to ensure that all flash cells are written uniformly, since the lifetime of flash cells is directly related to write frequencies. It is known that SLC-type NAND flash cells die after 100,000 writes. The lifetime of MLC-type NAND flash cells is 10 times less than that of the SLC-type NAND flash cells. Also, its access time is much longer than that of SLC-type flash cells. Hence, the NAND flash interface supports multiple channels and ways to increase parallel bandwidths and to hide flash memory's program latency. The cache buffer in the diagram utilizes DRAM devices, and the host interface is the communication path between the host and the SSD.

3.2 Conventional PC Architecture with SSD

The block diagram of a conventional PC architecture with an SSD is shown in Fig. 2. In general, the conventional PC architecture has two bridges for peripheral devices—North Bridge and South Bridge. The North Bridge allows the CPU to access high-speed peripherals, such as DRAM and video adapters, while the South Bridge manages relatively low-speed peripherals, such as SSDs (HDDs), USB, and LAN.

In this architecture, our focus is on the SSD access that occurs when CPU reads (writes) data from (to) memory due to a page fault. If a page fault occurs during a read operation, a victim page is selected from the main memory (DRAM) according to the virtual page management policy, and the victim page is written to the SSD by the DMA WRITE command issued from CPU. The required new page is then loaded into the main memory by the DMA READ command issued from CPU. In case of a write operation, CPU issues a DMA WRITE command only when the victim page is dirty if the write-back policy is employed in the virtual page management scheme. On the other hand, every write operation incurs DMA WRITE if the virtual page management scheme uses the write-through policy. In this work, we consider only the write-back policy, which is more widely used.

More details of DMA WRITE and DMA READ are depicted in Fig. 3. The suboperations used in this figure are listed in Table 1. Note that a large-size data block is packetized and transferred between DRAM and SSD as many times as the number of packets. The packet transfer is composed of two steps. The first step is the header transfer and the second step is the payload (or data) transfer. The header information is used to access the Physical Region Descriptor (PRD) table in DRAM (Ⓑ), while the payload transfer corresponds to suboperation Ⓔ.

Fig. 3a shows a DMA WRITE transfer between main memory (DRAM) and SSD. First, the software used issues a DMA WRITE command to the SATA controller in the South Bridge (Ⓐ). The SATA controller then fetches the starting address and size of the data to be transferred from the PRD table in DRAM (Ⓑ). According to the DMA transfer information, the DMA in SATA controller reads data from the DRAM (Ⓔ) and then, the data pass through the SSD interface (Ⓓ) and are delivered to the SSD (Ⓒ). The steps from Ⓑ to Ⓒ are repeated until the entire data is completely transferred. At the end of the DMA transfer the SSD signals an interrupt and then the transfer is completed (Ⓕ). Note that the DMA WRITE is performed in a consecutive manner. More precisely, when there are two consecutive DMA WRITEs, the second DMA WRITE is started only after the first DMA WRITE completely writes the data to the NAND flash memories in the SSD for protecting write failure.

DMA READ transfer sequence is also shown in Fig. 3b. After a DMA READ command is issued by CPU, the DMA in SATA controller requests data to the SSD (Ⓐ). As soon as the packet header including data ID arrives at the DMA, the

SATA controller fetches the PRD table in the DRAM (Ⓑ). At the same time, the payload of the packet is transferred to DMA through the SSD interface (Ⓒ, Ⓓ). Then, the payload is transferred to DRAM through the South Bridge and the North Bridge (Ⓔ). During the system transfer (Ⓔ), the next packet is transferred from SSD to the DMA in a pipelined manner. Finally, SSD signals an interrupt, and then the transfer is completed at the end of the DMA transfer (Ⓕ).

3.3 Limitations of Conventional Architecture

The SSD in conventional PC architecture simply replaces a HDD while keeping the same interface protocol to maintain the compatibility. This architecture, however, is not suitable for providing higher performance for the following reasons: First, the maximum bandwidths of PATA and SATA2 are only 133 and 300 MB/s, respectively. The interface may become a bottleneck due to its insufficient bandwidth as the performance of SSDs gets higher. Second, every access to SSD has to pass through the North Bridge and the South Bridge, meaning that a single data transfer request for SSD needs to be arbitrated twice. Last, a page fault during read operations serializes DMA WRITE and DMA READ commands due to a limited communication means between DRAM and SSD. To achieve higher performance by resolving these issues, several architectural choices will be discussed the Section 5.

4 Techniques for Architecture Exploration

In this section, we describe two techniques to overcome the limitations of the conventional architecture mentioned in Section 3.3. Section 4.1 presents the first technique to resolve the location and interface protocol issues. The second technique, which is to provide data transfer concurrency, is described in Section 4.2.

4.1 SSD with DDR DRAM Interface

The first technique we describe is to enable SSDs to communicate outside using the DDR DRAM interface for higher performance [1]. A typical SSD cannot be connected to the host via the DDR DRAM interface. This is because the interface mechanism DRAM devices use is different from that normal SSDs use. Typical DDR DRAMs have a fixed Column Address Strobe (CAS) latency, which is the time to get ready requested output data in read mode. In contrast, SSDs do not have a constant response time due to the internal cache buffer used for reducing response time.

To resolve this discrepancy, [1] exploited DQS, a feedback clock for supporting synchronization in high-speed DRAMs. DQS is defined in the DDR DRAM standard protocol (JEDEC, [18]) and can be found in most DDR DRAMS available in the market. Usual DDR DRAMs issue DQS in sync with CAS when the data requested by the host is ready. In this scheme, an SSD asserts DQS when the requested data is ready either from the internal NAND devices or from the inside cache, rather than synching DQS with CAS. The timing diagrams of this DQS-based signaling scheme for a cache miss and a cache hit are shown in Figs. 4a and 4b, respectively.

In order to incorporate this technique into the conventional architecture, the host memory controller needs to be designed to support the DQS scheme, which is not against the standard DDR DRAM interface.

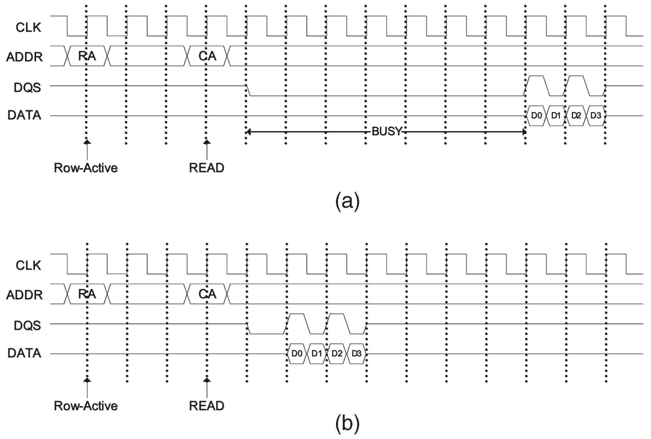


Fig. 4. Timing diagram of SSD with DDR DRAM interface [1]. (a) Cache buffer read miss. (b) Cache buffer read hit.

Note that we do not intend to change the function of SSD, but to keep it identical to the conventional HDD over the all proposed architectures to be discussed. Hence, the data access unit of SSD is also the same as that of HDD. In modern PC architecture, the second mass storages such as SSD and HDD are accessed in the unit of virtual page size which is typically a few Kbytes, thus the transaction between the host machine and SSD will take place in the unit of virtual page size. (Also, it is possible to divide a single large transaction into several small transactions depending on the implementation.) In other words, we only need to consider the block-level data transfers for SSDs.

4.2 Direct Path between Main Memory and SSD

The second technique is using a direct path between main memory and an SSD. This opens up another communication channel between them in addition to the conventional channel and can result in higher performance by diversifying memory access traffics, especially in memory-intensive applications. In order to implement this idea, we need a dual-port DRAM device and a dual-port SSD in addition to a DDR DRAM controller that should be embedded in the SSD to transfer data over the direct path. In this dual-port SSD, one port is connected to either the SATA controller in the South Bridge or the DDR DRAM controller in the North Bridge, and the other port is directly connected to main memory.

4.2.1 Dual-Port DRAM

OneDRAM [19], [20], a recently announced dual-port DRAM, may be a good candidate for the dual-port DRAM unit needed in the proposed technique. OneDRAM allows two separate controllers to access each memory bank in the device. Due to a shared memory bank, it is also possible to exchange data much faster between the two controllers. However, OneDRAM can incur excessive synchronization overhead because there is only one shared bank, and a hardware semaphore should be acquired in order to access this shared region.

To alleviate this limitation, we introduce a new synchronization scheme as depicted in Fig. 5. A dual-port DRAM that follows this timing diagram is compatible with our direct path scheme. When two memory controllers (e.g., one

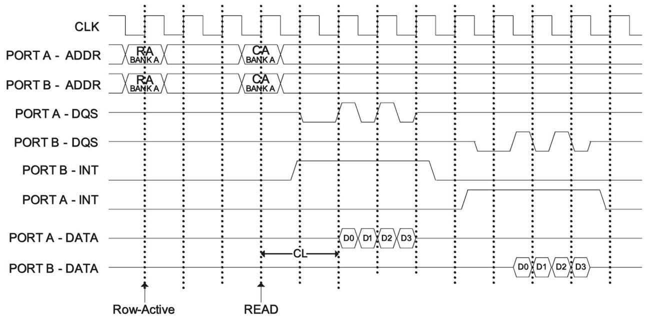


Fig. 5. Timing diagram of dual-port DRAM.

connected to the North Bridge and the other to the SSD) give commands to an identical bank simultaneously, only one command can be processed. To schedule these commands properly, each bank has its own INT (Interrupt) pin that informs both memory controllers of the state (busy or not) of this bank. When a certain bank is being accessed by both ports at the same time, an exclusive access per port is performed in a round-robin manner. More precisely, when one port (e.g., Port A in Fig. 5) accesses a certain bank, the INT signal of the other port (Port B) is driven HIGH continuously to notify the memory controller that this bank is now busy. After the data transfer to/from the bank is over, the INT signal goes to LOW to inform the memory controller that the bank is now available, and then the waiting port (Port B) gets access permission. The memory controller connected to each port should be designed to receive the INT signal.

4.2.2 Dual-Port SSD

The aim of a dual-port SSD is to increase the throughput of an SSD by allowing concurrent transactions. The interface scheme of each SSD port can be SATA or DRAM interface depending on the target architectures which are discussed in Section 5. In order to allow the concurrent transactions from two SSD ports, they must be free from the data hazard conditions. In this section, we assume that the two transactions are independent to each other, since the Operating System (OS) packs two independent transactions, as discussed in Section 4.2.3. In order to access an SSD concurrently, the cache buffer in the SSD should also be concurrently accessible. Therefore, we propose to use dual-port DRAM as the cache buffer in an SSD. The behavior of a dual-port DRAM is already discussed in Section 4.2.1.

Even though two concurrent transactions are independent from the data dependency perspective, they may still cause conflicts to access shared resources, such as channels and DRAM ports of the cache buffer. Fig. 6a shows how the cache buffer controller manages the resource conflict of two concurrent transactions.[2] First, there is a *Cache-Hit Detector* dedicated to each SSD port to check whether the requested data can be accessed from the cache buffer. For instance, if the requested data from the upper SSD port (PORT 0) is found from the cache buffer, the *Cache-Hit Detector 0* raises HIT0 to "1." HIT0 becomes "0," if the cache miss is detected by the *Cache-Hit Detector 0*. The *Cache-Hit Detector 1* also

2. We only show the important control signals in Fig. 6a for simplicity, but the data lines, address lines, and other control lines also follow the specified lines.

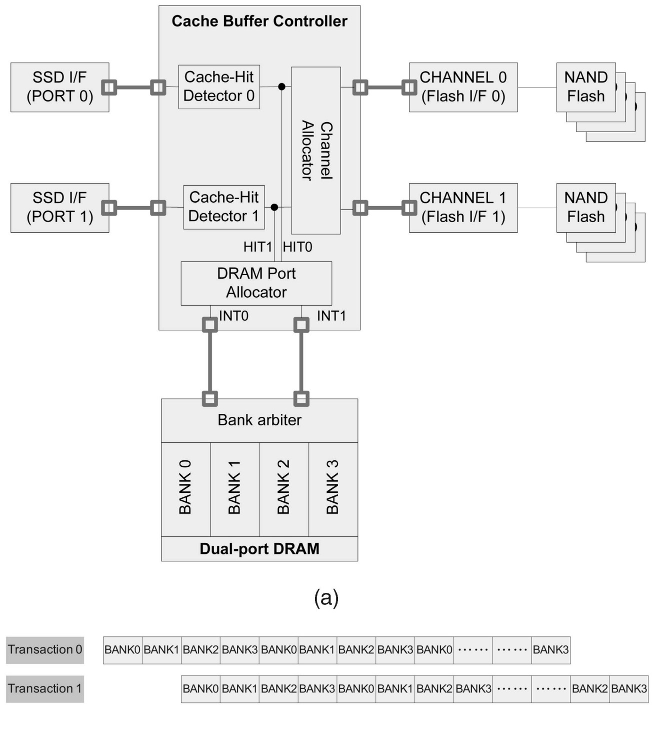


Fig. 6. Dual-port SSD. (a) Architecture of dual-port SSD. (b) Transaction process of dual-port SSD.

behaves in the same manner for the lower SSD port (PORT 1). Depending on the decisions of two *Cache-Hit Detectors*, there are four possible combinations of HIT0 and HIT1. If HIT0 (HIT1) is "1" and HIT1 (HIT0) is "0," there will be no resource conflict, since the upper (lower) SSD port will access the cache buffer and the lower (upper) SSD port will access the channels to NAND Flash memories. If both signals are zeros or ones, however, two SSD ports will attempt to concurrently access the DRAM ports or channels. To resolve such contention, the proposed cache buffer controller includes *DRAM Port Allocator* and *Channel Allocator*.

Since there are two SSD ports and two DRAM ports, the upper SSD port is dedicated to DRAM port 0 and the lower SSD port is dedicated to DRAM port 1. The major role of *DRAM Port Allocator* is to appropriately manage the requests to the dual-port DRAM, according to the signal statuses of INT0 and INT1 for avoiding bank conflict. For instance, if INT0 is in busy state, it means that the request through DRAM port 1 is already accessing the bank which is also the target of the request through DRAM port 0. In this case, *DRAM Port Allocator* delays the access through DRAM port 0 until *Bank Arbiter* in a dual-port DRAM raises INT0 to "1" (ready-state), when the request through DRAM port 1 completes the target bank access. In other words, it is allowed to access different banks simultaneously, but different regions of a bank cannot be accessed at the same time in our scheme. As shown in Fig. 6b, when there are two transactions (i.e., write and read transactions), each transaction does not access the bank which is being accessed by the other transaction in an interleaved fashion.

On the other hand, *Channel Allocator* behaves in a somewhat different way, since the number of channels does not always match to the number of SSD ports. Like the

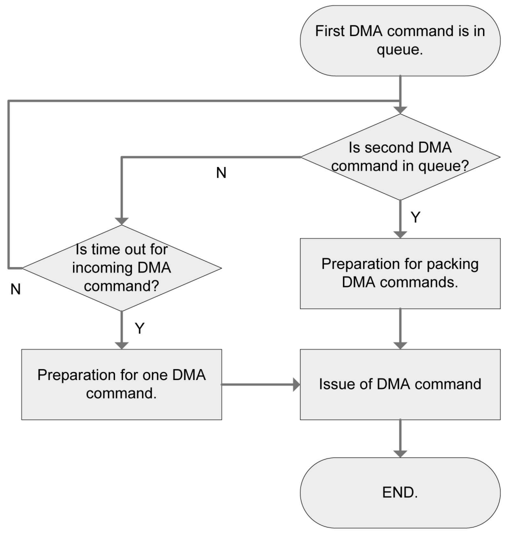

First DMA command is in queue.
Is second DMA command in queue?
N
Y
Is time out for incoming DMA command?
N
Y
Preparation for packing DMA commands.
Preparation for one DMA command.
Issue of DMA command
END.

Fig. 7. Procedure for DMA command packing.

conventional SSD, the logical address given by an SSD port is translated into a physical address by the FTL running on the processor and a channel corresponding to the physical address is allocated to the request. When both requests from two SSD ports should be allocated to the same channel, *Channel Allocator* arbitrates these requests in a round-robin manner. Also, when the channel corresponding to the physical address of a request from an SSD port is already in use by the request from the other SSD port, the request is pended until the completion of the current use of the channel.

4.2.3 DMA Command Packing

We explain how to exploit the concurrency provided by the dual channel between main memory and an SSD. In the conventional architecture, when a DMA transfer is required, OS generates a single DMA command. Only after it is finished completely, the next DMA command can be generated. In contrast, in the dual-port SSD architecture we propose, OS can pack a pair of DMA commands into one, thereby exploiting the concurrency existing in the dual path. This packing of DMA commands occurs only when different main memory and SSD regions are accessed. Otherwise, DMA commands are issued one by one just as the conventional scheme, in order to guarantee data consistency. Fig. 7 shows a flowchart for the DMA packing procedure. OS manages a queue for storing DMA commands and checks if there exists any DMA command in the queue. If there are two compatible (i.e., accessing different regions) DMA commands waiting in the queue, they are packed into one. If there is a single DMA command and no other DMA command comes in within a time-out limit, the command in the queue is issued as it is.

5 Architecture Exploration

Depending upon the SSD interface location and the existence of a direct path between main memory and an SSD, four architectural combinations are possible as listed in Table 2. The conventional architecture explained in Section 3 corresponds to one of these four architectures. Based upon the techniques described in Section 4, we explain each of the

TABLE 2
Four Architectural Choices

Direct Path	Location of the SSD Interface	
	South Bridge	North Bridge
X	Conv. Architecture (Section 3, Fig. 2)	NBSP Architecture (Section 5.1, Fig. 8) 1. DMA in DRAM controller in North Bridge 2. DQS scheme supported DRAM controller in North Bridge
O	SBDP Architecture (Section 5.2, Fig. 10) 1. Dual-port DRAM supported DRAM controller in North Bridge 2. DMA command packing supported OS	NBDP Architecture (Section 5.3, Fig. 11) 1. DMA in DRAM controller in North Bridge 2. DQS scheme supported DRAM controller in North Bridge 3. Dual-port DRAM supported DRAM controller in North Bridge 4. DMA command packing supported OS

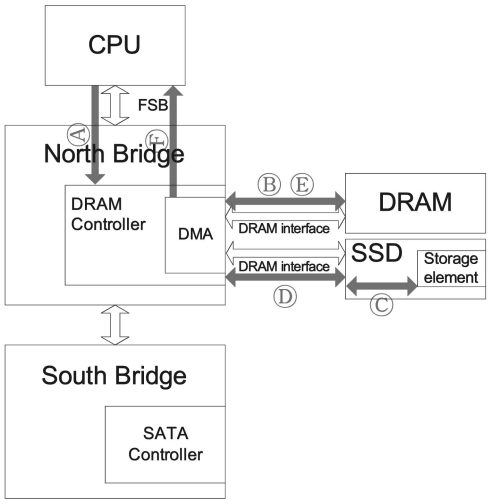


Fig. 8. NBSP architecture.

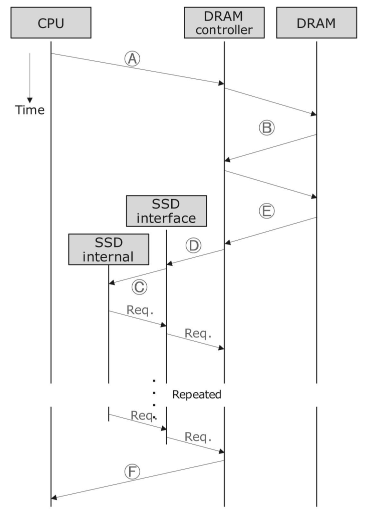
remaining three architectures, which we call North Bridge Single Port (NBSP) Architecture, South Bridge Dual Port (SBDP) Architecture, and North Bridge Dual Port (NBDP) Architecture. Table 2 also includes the information on what needs to be added into the conventional architecture in order to realize NBSP, SBDP, and NBDP architectures.

Of note is that we assume the DRAM interface exists in the North Bridge and the SATA controller exists in the South Bridge, as is the case with virtually all systems in the market. Placing the DRAM interface in the South Bridge and the SATA controller in the North Bridge is thus not considered.

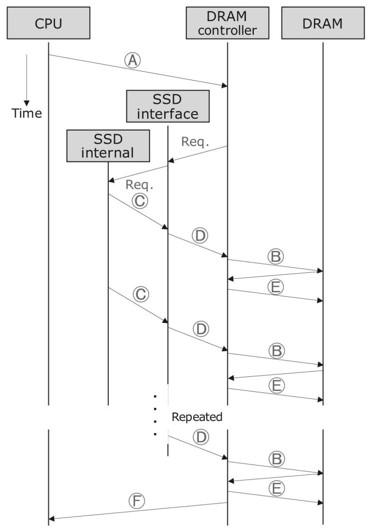
The limitations of the proposed architectures include that they require some modifications to the existing PC architecture as summarized in Table 2. For instance, to exploit NBSP architecture, the PC DRAM controller needs to be equipped with DMA as well as it should be able to understand the proposed DQS scheme.

5.1 NBSP Architecture

Fig. 8 shows the block diagram of NBSP Architecture [1]. In this architecture, an SSD is attached to the North Bridge



(a)



(b)

Fig. 9. DMA transfers of NBSP architecture. (a) DMA WRITE (consecutive). (b) DMA READ (pipelined).

using the DDR DRAM interface described in Section 4.1. In addition, a DMA controller is integrated in the North Bridge, unlike the conventional architecture. The major advantages of NBSP Architecture include the following: First, the DDR DRAM interface can provide a higher bandwidth (e.g., the bandwidth of a DDR2-800 module is 6,400 MB/s) than the conventional SATA interface. Second, the South Bridge is eliminated in the communication path between CPU and the SSD, thus reducing the arbitration overhead.

Due to this elimination, the DMA transfer protocol of the conventional architecture must be changed. OS now issues a DMA command to the DRAM controller in the North Bridge not to the SATA controller in the South Bridge. PRD fetching and the data transfer between DRAM and the SSD are also performed by the DMA controller in the DRAM controller. The rest of the DMA transfer steps are identical to the read and write modes in the conventional architecture as described in Section 3. The new transfer protocols are shown in Figs. 9a and 9b.

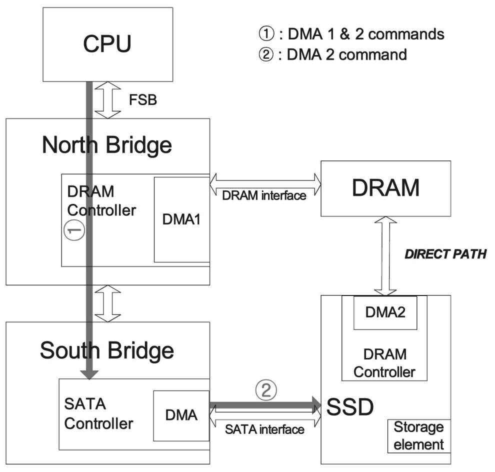


Fig. 10. SBDP architecture.

5.2 SBDP Architecture

In this architecture, an SSD is attached to the South Bridge as the conventional architecture, but there exists a direct path between the SSD and main memory, as shown in Fig. 10. Architecture SBDP can provide some performance improvements in multitasking environments. For instance, when a page fault occurs, write and read operations from main memory to the SSD should be performed consecutively. In SBDP Architecture, write and read operations can be performed virtually in a concurrent fashion, since the second operation can start immediately without waiting for the first operation to be finished.

Note that the SSD used in SBDP should have a DMA controller because of the need to access main memory directly. By handling two DMA controllers in the SSD and the SATA controller, data can be transferred to/from different regions at the same time, and multitasking will thus be supported effectively.

For simultaneous DMA operations, we should consider the data consistency issue. When two DMA controllers access different main memory and SSD regions, there is no consistency issue. In contrast, when the same main memory region is accessed by two DMA controllers, the data consistency problem may arise (e.g., when a page fault occurs by the OS virtual memory management). If this is the case, a victim page is saved to the SSD, and the new, valid page from the SSD is read to the same main memory region. To maintain data consistency, the victim page saving should precede the transfer of the new page to the main memory. That is, by making the DMA controller in the SSD access main memory earlier than the DMA controller in the SATA controller does, data consistency can be guaranteed.

To process two consecutive DMA commands that can be concurrently executed, the following steps are, therefore, needed as indicated in Fig. 10. ① CPU gives a combined command which consists of two DMA commands to the DMA controller in the SATA controller (DMA1). ② The first command is transferred to the DMA controller in the SSD (DMA2), while DMA1 controller performs the second command. This is to follow the policy to enforce data consistency as described above.

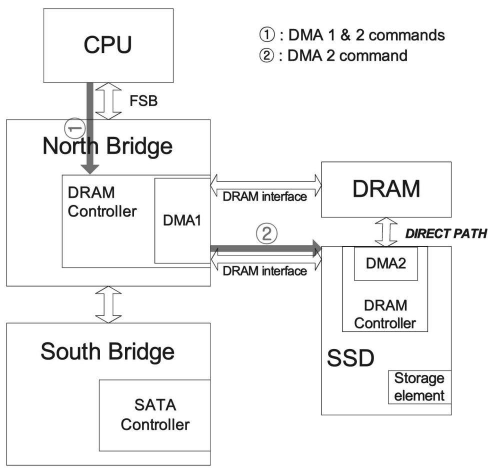


Fig. 11. NBDP architecture.

5.3 NBDP Architecture

In this architecture, an SSD is connected to the North Bridge, and there is a direct path between the SSD and main memory, as shown in Fig. 11. There are two DMA controllers, one in the North Bridge and the other in the SSD. By controlling these two controllers, two consecutive DMA commands can be executed almost concurrently by the same mechanism as used in SBDP Architecture. Of note in this architecture is how an interrupt operation works when DMA commands in the SSD are completed. Since the interface point of the SSD is the DDR DRAM, there is no way to transfer the interrupt to the North Bridge directly. To address this problem, an interrupt pin which signals the completion of DMA transfers should be assigned to the interrupt controller in a PC system.

6 Experimental Results

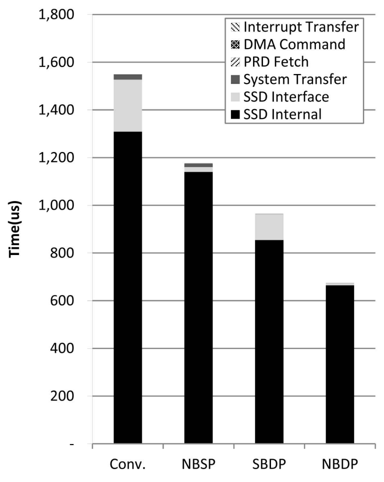
We implemented four transaction-level PC architectures with SystemC [21] to compare their performances. The first one is conventional PC architecture with SSD as shown in Fig. 2, and the second one is enhanced PC architecture with DDR DRAM interfaced SSD (NBSP Architecture) as shown in Fig. 8. The others are conventional PC architecture with direct path (SBDP Architecture) and enhanced PC architecture with DDR DRAM interfaced SSD and direct path (NBDP Architecture) in Figs. 10 and 11, respectively.

The PC system modeling was carried out based on Intel's 965 chipset (North Bridge) and ICH8 (South Bridge) specifications in [22] and [23], respectively. Also, the SSD modeling referred to the SATA specification [24] and an SSD datasheet of Samsung Electronics [25]. To show the peak performance of each interface, the SATA bandwidth was set to 300 MB/s and that of the DDR DRAM was set to 6,400 MB/s (DDR2-800). In the first set of experiments, we measured the performance of DMA operations of all four architectures when a single page is transferred between DRAM and an SSD in Sections 6.1.1 (DMA READ) and 6.1.2 (DMA WRITE). In the second set of experiments, we compared four architectures when they handle page-fault occurrences (Section 6.2). In this case, two pages are transferred at the same time—the victim page is moved from DRAM to SSD and vice versa for the required page. In the next set of experiments, we consider a scenario in which the PC downloads streaming data through the network (Section 6.3) to appreciate their performance when a large size of data is sequentially written to an SSD. Finally, we conducted the experiments for four architectures with the real traces collected from a real PC to show their effectiveness in real situations.

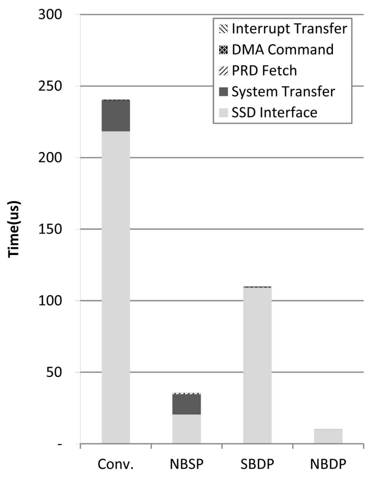
6.1 Performance Measure in Single Page Transfer

6.1.1 DMA READ Operation

We carried out the experiments on 64 KB DMA READ transfers by simulating the four architectures. The data response time was measured for two extreme DMA read cases. The first case always causes a cache miss for every access to an SSD, as shown in Fig. 12a, while the other case represents 100 percent cache hit, as shown in Fig. 13. In NBSP Architecture, for the 100 percent cache read miss case, the performance improvement ratio is about 1.31, while it is 8.70 in case of 100 percent cache read hit. This is because a data access to the cache buffer is much faster than NAND flash in the best case (100 percent of cache read hit). Note

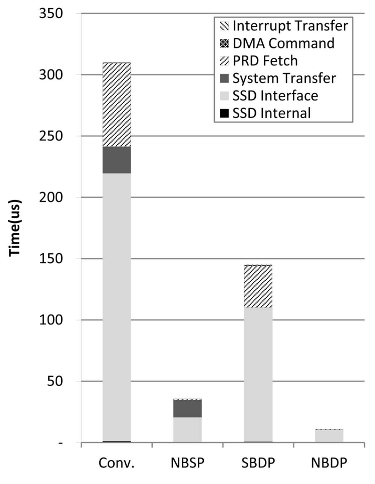


(a)

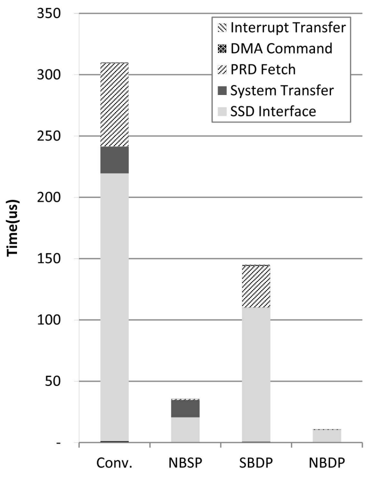


(b)

Fig. 12. DMA READ cache buffer read miss. (a) DMA READ transfer time. (b) Transfer time excluding SSD internal.



(a)



(b)

Fig. 13. DMA READ cache buffer read hit. (a) DMA READ transfer time. (b) Transfer time excluding SSD internal.

that the performance improvement of this architecture will become larger as the performance of an SSD itself is increasing, since the internal latency of the SSD accounts for most of the overall data transfer time. Such a claim is clearly supported by Fig. 12b, in which we analyzed DMA READ transfer time excluding the internal latency of the SSD. The performance improvement ratio of NBSP Architecture is about 6.76 in cache read miss. On the other hand, the improvement ratio is almost identical in case of 100 percent cache read hit (Figs. 13a and 13b), since its internal latency is negligible compared to other factors.

In case of SBDP and NBDP Architectures, there were also performance improvements compared with the conventional architecture, since these architectures use a direct path and the conventional data path at the same time. As shown in Figs. 12a and 13a, their performance improvement ratios are 1.60 and 2.29 in 100 percent read miss, 2.13 and 28.56 in 100 percent read hit in SBDP and NBDP Architectures, respectively. Notice that the SSD internal is reduced when we change the interface scheme from the conventional SATA to the proposed DRAM interface. The major reason of such reduction is basically due to the elimination of Cyclic Redundancy Check (CRC) [28] which is mandatory in high-speed interfaces with small voltage swing such as SATA for robustness.

In particular, the performance improvement ratio of NBDP Architecture in cache read hit is much higher than those of the other cases. The performance improvement ratio of NBDP Architecture, 28.56, is about 3.28 times higher than that of NBSP Architecture, 8.70. This is because the direct path can transfer about two times more data than North Bridge path in case of 100 percent cache read hit, hence, the overall performance improvement ratio of NBDP Architecture over NBSP Architecture becomes above 3. However, in case of cache read miss, this improvement is partially eliminated by long SSD internal reads.

We also measured how our proposed schemes have an effect on performance improvement in terms of the interface improvement, South Bridge elimination and use of the direct path. In NBSP Architecture, as shown in Table 3, the South Bridge elimination and DDR DRAM interface schemes contributed 1.89 and 98.11 percent, respectively, and the use of direct path did not contribute when a 100 percent cache read miss occurs. More specifically, since the reduction of the transfer time (DMA command, PRD fetch, and system transfer) by South Bridge elimination is hidden by the long internal latency of the SSD, the improvement caused by South Bridge elimination is marginal. On the contrary, when a 100 percent cache read hit occurs, South Bridge elimination, the DDR DRAM interface scheme, and the use of the direct path contributed 27.48, 72.52, and 0 percent, respectively, as shown in Table 4. In this case, since the SSD's internal latency is no longer large, the benefits caused by eliminating South Bridge path are less hidden and take more effects.

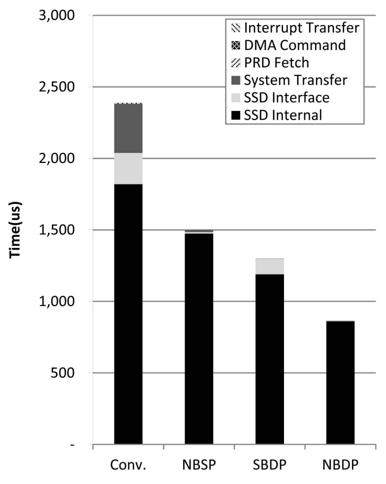
In case of SBDP Architecture, the only contributor for the performance improvement is the direct path scheme, whereas all of three factors are fully utilized in NBDP Architecture. Note that the contribution of a direct path reaches up to 85.58 percent when a 100 percent cache miss occurs. In other words, NBDP Architecture can largely exploit the concurrency of data transfers. This is mainly due to the page data layout originally designed for interleaving in a single-port DRAM. The maximum performance improvement ratio by use of a direct path is limited by NB, where NB

TABLE 3
Contribution Breakdown in DMA READ Miss

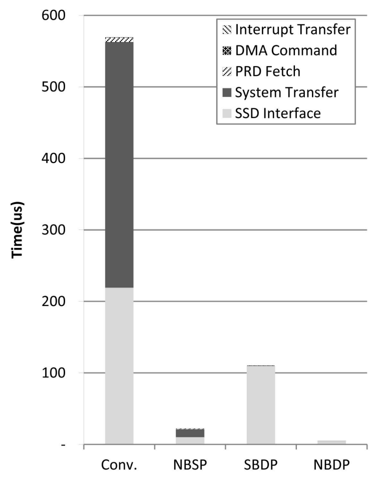
	SB elilmination	DRAM I/F	Direct path
NBSP	1.89%	98.11%	-
NBDP	2.52%	11.85%	85.58%

TABLE 4
Contribution Breakdown in DMA READ Hit

	SB elilmination	DRAM I/F	Direct path
NBSP	27.48%	72.52%	-
NBDP	30.04%	34.79%	35.16%



(a)



(b)

Fig. 14. DMA WRITE. (a) DMA WRITE transfer time. (b) Transfer time excluding SSD internal.

represents the number of banks in a dual-port DRAM. Even though overall system performance becomes better when a 100 percent cache hit occurs, its impact is lessened, since the SSD internal latency becomes much shorter and the other two factors become equally important.

6.1.2 DMA WRITE Operation

We also carried out the experiments on 64 KB DMA WRITE transfers by simulating the four architectures. As shown in Fig. 14a, the performance improvement ratio of the architectures with DDR DRAM interface scheme (NBSP Architecture) is 1.59, which is greater than the performance improvement ratio in DMA READ transfers with 100 percent of cache miss. This is much the same as the great performance improvement ratio by 25.47, when we exclude SSD's internal latency as shown in Fig. 14b. The result is closely linked to the elimination of South Bridge in the NBSP Architecture. In DMA WRITE, the data is fetched from the main memory and then written to the NAND flash memories inside an SSD. However, there is a big difference such that DMA WRITE operations are performed in a consecutive fashion, whereas the read operations are performed in a pipelined fashion as mentioned in Section 3. In other words, thanks to its sequential operational property, the elimination of South Bridge is more beneficiary to DMA WRITE operations. In case of SBDP and NBDP Architectures, there were performance improvements compared to the conventional architecture by a factor of 1.83 and 2.75, respectively, due to the same reason.

We also analyzed the contribution of the three factors as we did for DMA READ operations. As shown in Table 5, South Bridge elimination and DDR DRAM interface scheme contributed 37.92 and 62.08 percent, respectively, in NBSP Architecture. In SBDP Architecture, the use of the direct path was the only contributor, while all three factors fully contributed due to the same reason as mentioned in DMA READ operation in NBDP Architecture.

TABLE 5
Contribution Breakdown in DMA WRITE

	SB elilmination	DRAM I/F	Direct path
NBSP	37.92%	62.08%	-
NBDP	23.00%	7.03%	69.97%

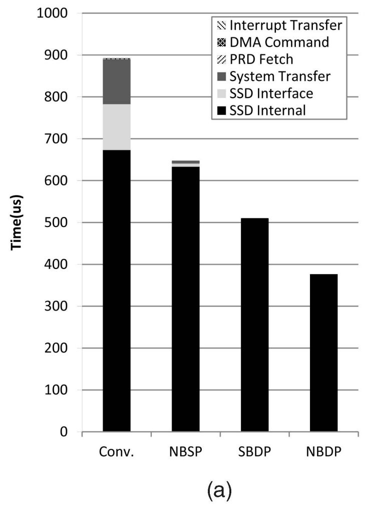
6.2 Performance Measure in Page-Fault Occurrence

In this situation, two pages need to be transferred simultaneously in the opposite direction, and the architectures with dual data paths (i.e., NBDP and SBDP) are expected to outperform the other architectures. To verify this reasoning, we analyzed a 16 KB page fault occurring in the four architectures. As shown in Fig. 15a, the performance improvement ratios of SBDP and NBDP Architectures are 1.74 and 2.37, respectively, with respect to the conventional architecture. NBDP Architecture also outperforms the NBSP Architecture by a factor of 1.72. (The performance improvement ratio are somewhat less than two due to internal overhead.)

We also measured how much contribution each of the proposed techniques makes to the overall performance. As shown in Table 6, the South Bridge elimination and the SSD with DDR DRAM interface schemes contribute 57.77 and 42.23 percent, respectively, in NBSP Architecture. In NBDP Architecture, all the proposed schemes are used in cooperation and provide the highest performance among all the architectures tested. In SBDP Architecture, the performance boost is totally due to the use of the direct path, and SBDP is not included in the table.

6.3 Performance Measure in Network Downloading

This is to analyze the situation in which multiple pages are transferred to an SSD. In the conventional architecture, a data packet received by a LAN controller is transferred to main memory and then arrives at an SSD via the North and South Bridges in order. Thus, the write operation to the SSD is performed only after the read operation from the LAN controller to main memory. However, SBDP and NBDP can exhibit better performance using the dual data paths. While an incoming data packet through the external network is transferred to the main memory, the previous data packet is



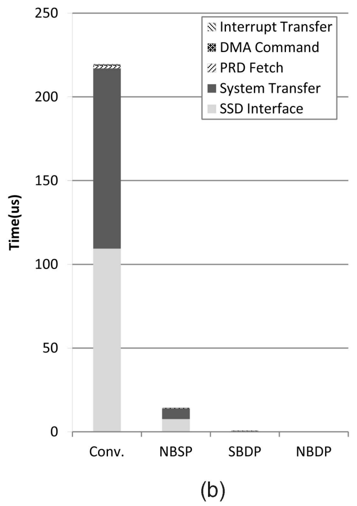
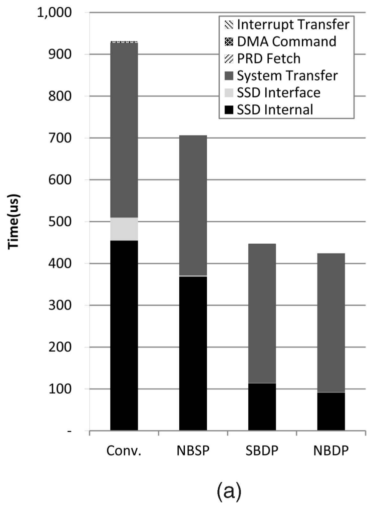


Fig. 15. Page-fault occurrence. (a) Page-fault processing time. (b) Transfer time excluding SSD internal.



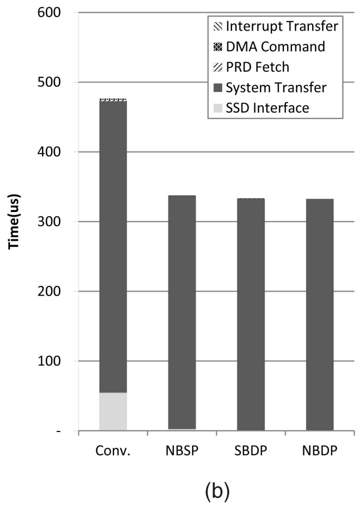


Fig. 16. Performance measure in network downloading. (a) Network downloading processing. (b) Transfer time excluding SSD internal.

delivered to the SSD through the direct path at the same time. Hence, SBDP and NBDP need not wait until the previously requested command is completed. Fig. 16a shows the result obtained from an experiment on 16 KB data packet transfers in the network downloading at 100 Mbps. SBDP and NBDP show performance improvement ratios of 2.08 and 2.19, respectively, with respect to the conventional architecture. Since the latency of a transfer from main memory to an SSD is mostly hidden, while a LAN-to-main-memory transfer is performed, the performance improvements are nearly doubled. Table 7 shows how much contribution each of the proposed techniques makes to the overall performance. Because the communication between main memory and the SSD occurs solely via the direct path between them in SBDP and NBDP, the contribution of the direct path scheme is most dominant in these architectures.

6.4 Performance Measure in Real Traces

To evaluate the performance of the four architectures under real workloads, we utilized the real trace data addressed in [2]. Table 8 lists more details of the traces we used. Each trace was collected from emulating web-surfing applications, e-mail sending/receiving, movie playing, downloading, document typesetting, gaming, and other applications on a PC.

The access patterns in these traces are more random than those in the synthetic workloads used in the previous set of experiments. Hence, the hit ratios of these traces

TABLE 6
Contribution Breakdown in Page-Fault Occurrence

	SB elilmination	DRAM I/F	Direct path
NBSP	42.23%	57.77%	-
NBDP	21.31%	10.60%	68.09%

TABLE 7
Contribution Breakdown in Network Downloading

	SB elilmination	DRAM I/F	Direct path
NBSP	38.52%	61.48%	-
NBDP	-	-	100%

TABLE 8
Summary of Traces

	Trace 1	Trace 2	Trace 3	Trace 4
Application	web-surfing, email sending/receiving, movie playing, downloading, document typesetting, gaming			
Duration	1 month	1 day	12 hours	12 hours
Read/Write ratio	54.04%/45.95%	31.48%/68.51%	83.22%/16.77%	46.55%/53.44%
Mean Read/Write size	8.2 sectors/5.7 sectors	6.5 sectors/14.0 sectors	34.38 sectors/6.9 sectors	19.8 sectors/22.7sectors

running on the conventional PC architecture were only 20.35, 19.32, 2.81, and 12.63 percent in Trace1, Trace2, Trace3, and Trace4, respectively. Also, we found that there is a correlation between the size of transfer and the hit ratio: The larger the transfer size is, the lower the hit ratio is. It means that the data transferred in a large chunk is mostly used once, whereas small data chunks tend to be reused. Even in this situation, NBDP stably shows about 2.6 times performance improvement over the conventional architecture, as shown in Fig. 17. Similarly, NBSP and SBDP run about 1.5 and 1.7 times faster than the conventional architecture.

As shown in Table 9, all architectures showed the largest improvement for Trace 2, which has the highest portion of write transfers. On the other hand, they commonly showed the least improvement for Trace 3, which has the smallest portion of write transfers. One reason is that write operations are benefitted more by eliminating the South Bridge due to its nonpipelined operational nature, as explained in Section 6.1. The direct path also accounts for some fraction of such improvements. Namely, the latency of the original data path is larger for writing, hence, the direct path can exploit more parallelism in write transfers than in read transfers. Therefore, SBDP shows the trend identical to those of NBDP and NBSP, even though the impact is relatively marginal.

To summarize, Table 9 shows the comprehensive experimental results we obtained in terms of performance improvement ratio.

7 Conclusion

In this paper, we proposed North Bridge interfaced SSD and a direct path between main memory and the SSD. By eliminating South Bridge path, arbitration overhead and data transfer time were reduced between CPU and SSD. In addition to that, the direct path provides a significant performance improvement under the concurrent memory access environment. By using above two techniques, we propose several architectural choices (NBSP, SBDP, and NBDP) to fully exploit the performance of SSDs used in consumer PC architectures.

The experimental results guide us how to trade off the performance enhancement and the change of PC architecture. On the whole, NBDP Architecture outperformed other architectures in the most of the cases. Especially, the performance improvement was maximized by 2.67 times in the NBDP Architecture for real workload. Also, the SBDP Architecture can be a proper alternative in terms of trade-off between the performance and compatibility. Even though the internal latency of the SSD is still dominant factor in data response time, we expect that our proposed architectures with SSD provide significant performance improvement in PC system by improving the internal architecture of the SSD.

Acknowledgments

This work was supported in part by Samsung Electronics Company, by the Korea Research Foundation Grant funded by the Korean Government (MEST) (313-2007-2-D00578), and by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korean government (MEST) (No. 2010-0015504, No. 2010-0000407, and No. 2010-0000631). Dong Kim and Seung-Hwan Ha were with the School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea.

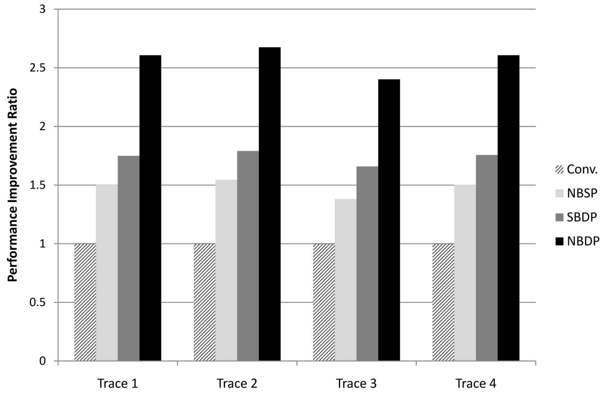


Fig. 17. Performance measure in real traces.

TABLE 9
Maximum Performance Improvement Ratio

Sub-operations	Conv.	NBSP	SBDP	NBDP
DMA READ miss	1	1.31	1.60	2.29
DMA READ hit	1	8.70	2.13	28.56
DMA WRITE	1	1.59	1.83	2.75
Page-fault	1	1.37	1.74	2.37
Network downloading	1	1.31	2.08	2.19
Real trace 1	1	1.50	1.74	2.60
Real trace 2	1	1.54	1.79	2.67
Real trace 3	1	1.38	1.65	2.40
Real trace 4	1	1.50	1.75	2.60

References

[1] D. Kim, K. Bang, S.-H. Ha, C. Park, S.W. Chung, and E.-Y. Chung, "Solid-State Disk with Double Data Rate DRAM Interface for High-Performance PCs," *IEICE Trans. Information and Systems,* vol. E92-D, no. 4, pp. 727-731, Apr. 2009.

[2] L.-P. Chang and T.-W. Kuo, "An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems," *Proc. IEEE Eighth Real-Time and Embedded Technology and Applications Symp. (RTAS),* pp. 187-196, Sept. 2002.

[3] C. Park, J. Kang, S.Y. Park, and J. Kim, "Energy-Aware Demand Paging on NAND Flash-Based Embedded Storages," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '04),* pp. 338-343, 2004.

[4] J.-H. Lee, G.-H. Park, and S.-D. Kim, "A New NAND-Type Flash Memory Package with Smart Buffer System for Spatial and Temporal" *J. Systems Architecture,* vol. 51, pp. 111-123, Feb. 2005.

[5] S.-L. Min and E.-H. Nam, "Current Trends in Flash Memory Technology," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC),* pp. 332-333, Jan. 2006.

[6] C. Park, P. Talawar, D. Won, M. Jung, J. Im, S. Kim, and Y. Choi, "A High Performance Controller for NAND Flash-Based Solid State Disk," *Proc. 21st IEEE Non-Volatile Semiconductor Memory Workshop (NVSMW),* pp. 17-20, Feb. 2006.

[7] J.-U. Kang, J.-S. Kim, C. Park, H. Park, and J. Lee, "A Multi-Channel Architecture for High-Performance NAND Flash-Based Storage System" *J. Systems Architecture,* vol. 53, pp. 644-658, Sept. 2007.

[8] L.-P. Chang, "Hybrid Solid-State Disks: Combing Heterogeneous NAND Flash in Large SSDs," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC '08),* pp. 428-433, Mar. 2008.

[9] R. Schuetz, H. Oh, J.-K. Kim, H.-B. Pyeon, S.A. Przybylski, and P. Gillingham, "HyperLink NAND Flash Architecture for Mass Storage Applications," *Proc. 22nd IEEE Non-Volatile Semiconductor Memory Workshop (NVSMW),* pp. 26-30, Aug. 2007.

[10] S. Kim, C. Park, and S. Ha, "Architecture Exploration of NAND Flash-Based Multimedia Card," *Proc. Design, Automation and Test in Europe (DATE '08),* pp. 218-223, Mar. 2008.

[11] J.H. Yoon, E.H. Nam, Y.J. Seong, H. Kim, B.S. Kim, S.L. Min, and Y. Cho, "Chameleon: A High Performance Flash/FRAM Hybrid Solid State Disk Architecture" *IEEE Computer Architecture Letters,* vol. 7, no. 1, pp. 17-20, Jan.-June 2007.

[12] J.K. Kim, H.G. Lee, S. Choi, and K.I. Bahng, "A PRAM and NAND Flash Hybrid Architecture for High-Performance Embedded Storage Subsystems," *Proc. Seventh ACM Int'l Conf. Embedded Software,* pp. 31-40, Oct. 2008.

[13] H. Pon and K. Rao, "A NAND Flash PC Platform Read Write Cache," *Proc. 22nd IEEE Non-Volatile Semiconductor Memory Workshop (NVSMW),* pp. 21-22, Aug. 2007.

[14] Fusion IO white paper, www.fusionio.com, 2010.

[15] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A Case for Flash Memory SSD in Enterprise Database Applications," *Proc. ACM SIGMOD,* pp. 1075-1086, 2008.

[16] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND Flash Based Disk Caches," *Proc. 35th Int'l Symp. Computer Architecture (ISCA),* pp. 327-338, 2008.

[17] A. Caulfield, L. Grupp, and S. Swanson, "Gordon: Using Flash Memory to Build Fast, Power-Efficient Clusters for Data-Intensive Applications," *Proc. 14th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS),* pp. 217-228, 2009.

[18] *JEDEC Standard, Double Data Rate (DDR) SDRAM Specification,* JEDEC Solid State Technology Assoc., 2005.

[19] H. Yang, S. Kim, H.-W. Park, J. Kim, and S. Ha, "Performance Evaluation and Optimization of Dual-Port SDRAM Architecture for Mobile Embedded Systems," *Proc. Int'l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASES '07),* pp. 53-57, 2007.

[20] K. Nam, J.-S. Kim, C.S. Oh, H. Sohn, D.H. Lee, C. Lee, S. Kim, J.-W. Park, Y. Kim, M. Kim, J. Kim, H. Lee, J. Kwon, D.I. Seo, Y.-H. Jun, and K. Kim, "A 512 Mb 2-Channel Mobile DRAM (OneDRAM) with Shared Memory Array," *Proc. IEEE Asian Solid-State Circuits Conf. (ASSCC '07),* pp. 204-207, 2007.

[21] *IEEE Standard SystemC Language Reference Manual,* 2010.

[22] Intel Corporation "Intel 965 Express Chipset Family," data sheet, July 2006.

[23] Intel Corporation "Intel I/O Controller Hub 8 (ICH8) Family," data sheet, July 2006.

[24] *SATA Specification (Revision 1.0a),* http://www.serialata.org, 2010.

[25] Samsung Electronics Company, "Nand Flash Based SSD Preliminary Specification," data sheet, July 2007.

[26] R.H. Katz, G.A. Gibson, and D.A. Patterson, "Disk System Architectures for High Performance Computing" *Proc. IEEE,* vol. 77, no. 12, pp. 1842-1858, Dec. 1989.

[27] "ASP Comparison for 120GB HDD versus 128GB Flash-Based SSD, 2007-2012," Int'l Data Corporation (IDC), 2008.

[28] W.W. Peterson and D.T. Brown, "Cyclic Codes for Error Detection," *Proc. IRE,* vol. 49, pp. 228-235, Jan. 1961.

Dong Kim received the BS and MA degrees in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2007 and 2009, respectively. He is currently an engineer with Flash Solution R&D center, Samsung Electronics in Hwasung-City, Gyenggi-Do, Korea. His research interests include embedded systems, NAND-flash-based mass storage architecture, and computer architecture.

Kwanhu Bang received the BS degrees in computer science and in electronic engineering and the MS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2006 and 2008, respectively. He is currently a PhD candidate in the School of Electrical and Electronic Engineering at Yonsei University. His research interests include biocomputation, flash memory applications, and system-level low-power design. He is a student member of the IEEE.

Seung-Hwan Ha received the BS and MA degrees in electrical and electronic engineering from Yonsei University in Seoul, Korea, in 2008 and 2010, respectively. He is currently an engineer with Flash Software Development Project, Samsung Electronics in Hwasung-City, Gyenggi-Do, Korea. His research interests include system-on-chip, NAND-flash-based mass storage architecture, and system architecture.

Sungroh Yoon received the BS degree in electrical engineering from Seoul National University, Korea, in 1996 and the MS and PhD degrees in electrical engineering from Stanford University, USA, in 2002 and 2006, respectively. From 2006 to 2007, he was with Intel Corporation, Santa Clara, USA, where he participated in developing Atom and Core i7 microprocessors. Previously, he held research positions at Stanford University and Synopsys Inc. Mountain View, USA. He is currently an assistant professor at School of Electrical Engineering at Korea University, Seoul, Korea. His research interests include system-level low-power design, flash memory applications, and biocomputation. He is a member of the IEEE.

Eui-Young Chung received the BS and MS degrees in electronics and computer engineering from Korea University, Seoul, in 1988 and 1990, respectively, and the PhD degree in electrical engineering from Stanford University, California, in 2002. From 1990 to 2005, he was a principal engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. He is currently an associate professor with the School of Electrical and Electronic Engineering, Yonsei University, Seoul. His research interests include system architecture, biocomputing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications and storage systems including Solid-State Disks (SSDs). He is a member of the IEEE and the IEEE Computer Society.