Article

Implications of NVM Based Storage on Memory Subsystem Management

Hyokyung Bahn ^{1,*} and Kyungwoon Cho ²¹ Department of Computer Engineering, Ewha University, Seoul 03760, Korea² Embedded Software Research Center, Ewha University, Seoul 03760, Korea; cezanne@ewha.ac.kr

* Correspondence: bahn@ewha.ac.kr; Tel.: +82-2-3277-4247

Received: 22 November 2019; Accepted: 30 January 2020; Published: 3 February 2020

Featured Application: The authors anticipate that memory and storage configurations explored in this article will be helpful in the design of system software for future computer systems with ever-growing memory demands and the limited density of DRAM.

Abstract: Recently, non-volatile memory (NVM) has advanced as a fast storage medium, and legacy memory subsystems optimized for DRAM (dynamic random access memory) and HDD (hard disk drive) hierarchies need to be revisited. In this article, we explore the memory subsystems that use NVM as an underlying storage device and discuss the challenges and implications of such systems. As storage performance becomes close to DRAM performance, existing memory configurations and I/O (input/output) mechanisms should be reassessed. This article explores the performance of systems with NVM based storage emulated by the RAMDisk under various configurations. Through our measurement study, we make the following findings. (1) We can decrease the main memory size without performance penalties when NVM storage is adopted instead of HDD. (2) For buffer caching to be effective, judicious management techniques like admission control are necessary. (3) Prefetching is not effective in NVM storage. (4) The effect of synchronous I/O and direct I/O in NVM storage is less significant than that in HDD storage. (5) Performance degradation due to the contention of multi-threads is less severe in NVM based storage than in HDD. Based on these observations, we discuss a new PC configuration consisting of small memory and fast storage in comparison with a traditional PC consisting of large memory and slow storage. We show that this new memory-storage configuration can be an alternative solution for ever-growing memory demands and the limited density of DRAM memory. We anticipate that our results will provide directions in system software development in the presence of ever-faster storage devices.

Keywords: NVM; storage performance; buffer cache; prefetching; PCM; STT-MRAM

1. Introduction

Due to the wide speed gap between DRAM (dynamic random access memory) and HDDs (hard disk drives), the primary goal of memory hierarchy design in traditional computer systems has been the minimization of storage accesses [1]. The access time of HDD is limited to tens of milliseconds, which is 5–6 orders of magnitude slower than DRAM access time. However, with the recent advances in fast storage technologies such as NAND flash memory and NVM (non-volatile memory), the extremely wide speed gap has been reduced [2–4]. The typical access time of NAND flash memory is less than 50 milliseconds, and thus the speed gap between storage and memory is reduced to three orders of magnitude. This trend has been accelerated by the appearance of NVM, of which the access time is about 1–100 times that of DRAM [5,6].

Patents published by Intel describe detailed micro-architectures to support NVM in various kinds of memory and storage subsystems, implying that the era of NVM is imminent [7,8]. Due to its good characteristics such as small access latency, low-power consumption, and long endurance cycles, NVM is expected to be used as secondary storage in addition with NAND flash memory and HDD [9–12].

NVM has also been considered as a candidate of the main memory medium as it is byte-addressable and has substantial density benefits [13]. However, its access time is rather slower than DRAM, and thus it is being considered as fast storage or far memory used along with DRAM. Though NVM is considered for both memory and storage, the focus of this study is in storage. As storage becomes sufficiently fast by adopting NVM, legacy software layers optimized for HDD need to be revisited. Specifically, storage performance becomes close to that of DRAM, and thus memory management mechanisms need to be reconsidered.

As flash-based SSD (solid state drive) was originally designed for substituting HDD, it is perceived as a fast HDD device from the operating system's viewpoint. Thus, some additional software layers such as FTL (flash translation layers) are installed inside the flash storage to be seen as an HDD-like block device. Due to this reason, major changes in the operating system are not necessary for flash-based SSD. In contrast, as the access latency of storage becomes close to that of DRAM by adopting NVM, we need to revisit the fundamental issues of the operating system's memory management subsystem. Early flash memory products suffered from the freezing phenomenon in which the performances of storage I/Os (input/outputs) are deteriorated seriously when the garbage collection starts [14]. Moreover, there are performance fluctuations in flash storage as the internal state of the device changes over time. Such problems have been improved significantly by adopting an internal buffer and/or cache in flash-based SSD products, and some recent SSDs consist of NVM for accelerating the storage performance even more.

In this article, we analyze the performance implication of NVM based storage and discuss issues in the design of memory subsystems with NVM storage. In particular, we consider two types of NVM storage media, PCM (phase-change memory) and STT-MRAM (spin transfer torque magnetic random access memory), and look into the effectiveness of various I/O mechanisms.

In deploying NVM to replace current storage in the traditional storage hierarchy, there are fundamental questions that may be raised with respect to the I/O mechanisms supported by current operating systems. We list some of the questions below and present our answers to these questions obtained through empirical evaluation.

- Q1. What will be the potential benefit in the memory-storage hierarchy of a desktop PC if we adopt fast NVM storage?
- Q2. Is the buffer cache still necessary for NVM based storage?
- Q3. Is prefetching still effective for NVM based storage?
- Q4. Is the performance effect of I/O modes (i.e., synchronous I/O, buffered I/O, and direct I/O) similar to HDD storage cases?
- Q5. What is the impact of concurrent storage accesses on the performance of NVM based storage?

This article answers the aforementioned questions through a wide range of empirical studies on systems with HDD and NVM storage emulated by a RAMDisk, which provides NVM performance by stalling DRAM access for the proper duration. To do so, we implement an NVM emulator whose performance can be measured with the setting of appropriate configurations for target NVM storage media. The use of a RAMDisk with no timing delay provides an upper-bound performance estimate of NVM storage, which can be interpreted as the optimistic performance of STT-MRAM. We also set an appropriate time delay for another type of NVM, PCM. Through this measurement study, we make the following findings.

- We can decrease the main memory size without performance penalties when NVM storage is adopted instead of HDD.

- Buffer caching is still effective in fast NVM storage but some judicious management techniques like admission control are necessary.
- Prefetching is not effective in NVM storage.
- The effect of synchronous I/O and direct I/O in NVM storage is less significant than that in HDD storage.
- Performance degradation due to the contention of multi-threads is less severe in NVM based storage than in HDD. This implies that NVM can constitute a contention-tolerable storage system for applications with highly concurrent storage accesses.

Based on these observations, we discuss a new memory and storage configurations that can address the issue of rapidly growing memory demands in emerging applications by adopting NVM storage. Specifically, we analyze the cost of a desktop PC consisting of small memory and fast storage based on Intel's Optane SSD, which is the representative NVM product on the market at this time, in comparison with the traditional PC consisting of large memory and slow storage. We show that such a new memory-storage configuration can be an alternative solution for ever-growing memory demands and the limited density of DRAM memory.

The remainder of this article is organized as follows. Section 2 describes the background of this research focusing on the hardware technologies of PCM and STT-MRAM. Section 3 explains the experimental setup of the measurement studies and the performance comparison results on systems with HDD and NVM storage. In Section 4, we summarize the related work. Finally, Section 5 concludes this article.

2. PCM and STT-MRAM Technologies

PCM (phase-change memory) stores data by using a material called GST (germanium-antimony-tellurium), which has two different states, amorphous and crystalline. The two states can be controlled by setting different temperatures and heating times. As the two states provide different resistance when passing the electric current, we can distinguish the state while reading the resistance of the cell. Whereas reading the resistance value is fast, modifying the state in a cell takes long time, and thus writing data in PCM is slower than reading. The endurance cycle, i.e., the number of the maximum write operations allowed for a cell is in the range of 10^6 – 10^8 in PCM, which is shorter than that of DRAM, and thus it is difficult to use PCM as a main memory medium. Thus, studies on PCM as the main memory adopt additional DRAM in order to reduce the number of write operations on PCM [6,13]. When we use PCM as storage rather than memory, however, such an endurance problem is not a significant issue any longer. Note that the endurance cycle of NAND flash memory is in the range of 10^4 – 10^5 , which is even shorter than that of PCM.

Although we consider PCM as storage rather than memory, we need to compare its characteristics with DRAM as the two layers are tightly related. For example, adopting fast storage can reduce the memory size of the system without performance degradations due to the narrowed speed gap between the two layers. We will discuss this issue further in Section 3. With respect to the density of media, PCM is anticipated to provide more scalability than DRAM and NAND flash memory. It is reported that the density of DRAM is difficult to be under 20 nm and NAND flash memory also has reached its density limit. Note that the cell of NAND flash memory will not be cost effective if its density is less than 20 nm. Accordingly, instead of scaling down the chip size, 3D stacking technologies have been attempted alternatively in order to scale the capacity. 3D-DDR3 DRAM and V-NAND flash memory have been produced by using this technology [15,16]. Unlike DRAM and NAND flash memory cases, it is anticipated that PCM will be stable even in the 5 nm node [17]. Samsung and Micron already announced 8 Gb 20 nm PCM and 45 nm 1 Gb PCM, respectively. By considering overall situations, PCM is expected to find its place in the storage market soon.

MLC (multi-level cell) is another technology that can enhance the density of PCM. Even though the basic prototype of PCM is based on SLC (single-level cell) technologies, which can represent only

the two states of crystalline and amorphous, studies on PCM have shown that additional intermediate states can be represented by making use of the MLC technologies [18]. That is, MLC can represent more than two states in a cell by selecting multiple levels of electrical charge. By virtue of MLC technologies, the density of PCM can be an order of magnitude higher than that of other NVM media such as MRAM (magnetic RAM) and FRAM (ferroelectric RAM), which have difficult structures to adopt MLC. Due to this reason, major semiconductor vendors, like Intel and Samsung, have researched PCM and are now ready to commercialize PCM products.

STT-MRAM (spin transfer torque magnetic random access memory) is another notable NVM technology. STT-MRAM makes use of magnetic characteristics of a material, of which magnetic orientations can be set and detected by using electrical signals. In particular, STT-MRAM utilizes the MTJ (magnetic tunneling junction) device in order to store data [19]. MTJ is composed of two ferromagnetic layers and a tunnel barrier layer, and data in MTJ can be represented by the resistance value, which depends on the relative magnetization directions of the two ferromagnetic layers [20]. If the two layers' magnetic fields are aligned in the same direction, the MTJ resistance is low, which represents the logical value of zero. On the other hand, if the two layers are aligned in the opposite direction, the MTJ resistance is high, which represents the logical value of one.

To read data in an STT-MRAM's cell, we should apply a small voltage between the sense and bit lines, and detect the current flow. To write data in an STT-MRAM's cell, we should push a large current through MTJ in order to modify the magnetic orientation. According to the current's direction, the two ferromagnetic layers' magnetic fields are aligned in the same or the opposite direction. The level of the current necessary for storing in MTJ is larger than that required for reading from it.

Table 1 lists the characteristics of STT-MRAM and PCM in comparison with DRAM and NAND flash memory [9,17,21]. As listed in the table, the performance and the endurance characteristics of STT-MRAM are very similar to those of DRAM, but STT-MRAM consumes less power than DRAM. Similar to NAND flash, PCM has asymmetric read/write latencies. However, PCM does not have erase operations, and its access latency and endurance limit are all better than NAND flash.

Table 1. Characteristics of non-volatile memory (NVM) technologies in comparison with dynamic random access memory (DRAM) and NAND flash memory.

	DRAM	STT-MRAM	PCM	NAND Flash
Maturity	Product	Prototype	Product	Product
Read latency	10 ns	10 ns	20–50 ns	25 μ s
Write latency	10 ns	10 ns	80–500 ns	200 μ s
Erase latency	N/A	N/A	N/A	200 ms
Energy per bit access (r/w)	2 pJ	0.02 pJ	20pJ/100 pJ	10 nJ
Static power	Yes	No	No	No
Endurance (writes/bit)	10^{16}	10^{16}	10^6 – 10^8	10^5
Cell size	6–8 F ²	>6 F ²	5–10 F ²	4–5 F ²
MLC	N/A	4 bits/cell	4 bits/cell	4 bits/cell

3. Performance Implication of NVM Based Storage

We perform measurement experiments to investigate the effect of NVM based storage. We set the configuration of the CPU, memory, and storage to Intel Core i5-3570, 8 GB DDR3, and 1 TB SATA 3 Gb/s HDD, respectively. We install the Linux kernel 4.0.1 64 bits and Ext4 file system. We consider two types of NVM storage, PCM and STT-MRAM. For now, as it is difficult to perform an experiment with commercial PCM and STT-MRAM products, we make use of RAMDisk consisting of DRAM and set a certain time delays for emulating NVM performances. To do so, we implement an NVM device driver by making use of the RAMDisk device driver and the PCMSim that is publicly available [22].

As the optimistic performance of STT-MRAM is expected to be similar to that of DRAM, we set no time delays for emulating STT-MRAM, which we call RAMDisk throughout this article. For the

PCM case, we set the latency of read and write operations to $4.4\times$ and $10\times$ that of DRAM, respectively, in our NVM device driver, which is the default setting of PCMsimsim [22].

In our NVM experiments, we also install the Ext4 file system on the RAMDisk. For benchmarking, we use IOzone, which is the representative microbenchmark that measures the performance of storage systems by making a sequence of I/O operations [23]. In our experiment, sequential read, random read, sequential write, and random write of IOzone scenarios with the total footprint of 2 GB were performed. We also use Filebench benchmark for some additional experiments.

3.1. Effect of the Memory Size

In this section, we measure the performance of NVM when the memory size is varied. Figure 1 shows the benchmarking performance of HDD, RAMDisk, and PCMDisk as a function of the memory size when executing IOzone. As shown in Figure 1a,b, not only RAMDisk but also HDD exhibit surprisingly high performances when the memory size is 1 GB or more. This is because all requested data already reside in the buffer cache when the memory size is large enough. However, as the memory size is small, there is a wide performance gap between HDD and NVM in read operations. Specifically, the throughput of the RAMDisk is over 5 GB/s for all cases but that of HDD degrades significantly as the memory size becomes 512 MB or less. The performance of the PCMDisk sits in between those of HDD and RAMDisk.

Figure 1. Performance comparison of a hard disk drive (HDD), RAMDisk, and phase-change memory (PCM) disk varying the main memory size.

This provides significant implications of utilizing NVM in future memory and storage systems. Specifically, the memory size is difficult to be extended any longer when considering the ever-increasing footprint of big data applications as well as the scaling limitation of DRAM. To relieve this problem, NVM may provide an alternative solution. That is, a future computer system can be composed of limited memory capacities but has NVM storage devices, not to degrade performances. Such “small memory and fast storage” hierarchies can be efficient for memory-intensive applications such as big data and/or scientific applications.

Now, let us see the results for the write operations. As shown in Figure 1c,d, the average performance improvements of the PCMDisk and RAMDisk are 1204% and 697%, respectively, compared

to the HDD performances. The reason is that a write system call triggers flush operations to the storage even though the memory space is sufficient. Specifically, file systems perform flush or journaling operations, which periodically write the modified data to storage not to lose the modifications against power failure situations. When flush is triggered, the modifications are directly written to the original position in the storage, whereas journaling writes the modifications to the storage's journal area first and then reflects them to the original position later. The default period of journaling/flush is set to 5 s in the Ext4 file system, and thus write operations cannot be buffered although sufficient memory space is provided.

3.2. Effectiveness of Buffer Cache

In this section, we investigate the performance effect of buffer cache when NVM storage is used. Buffer cache stores requested file data in a certain part of the main memory, thereby servicing subsequent requests directly without accessing slow storage. The primary goal of buffer cache management is in minimizing the number of storage accesses. However, as the storage becomes sufficiently fast by adopting NVM, one may wonder if the traditional buffer cache will be still necessary. To investigate this, we first set the I/O mode to synchronous I/O and direct I/O, and measure the performance of the NVM storage.

In the synchronous I/O mode, a write system call is directly reflected to storage devices. We can set the synchronous I/O mode by opening a file with the `O_SYNC` flag. In the direct I/O mode, file I/O bypasses the buffer cache and transfers file data between storage and user space directly. A direct I/O mode can be set by opening a file with the `O_DIRECT` flag. Figures 2–4 compares the performance of the default I/O mode, which is denoted as “baseline”, and the synchronous I/O mode and the direct I/O mode for HDD, RAMDisk, and PCMDisk, respectively. Note that the default I/O mode is set when user applications do not specify specific I/O modes, which performs I/O in asynchronous write, buffered I/O, and prefetching turned on.

Figure 2. HDD performance under different I/O (input/output) modes.

Figure 3. RAMDisk performance under different I/O modes.

Figure 4. PCMDisk performance under different I/O modes.

Let us first see the effect of the synchronous I/O mode. As shown in Figure 2a,b, Figure 3a,b, and Figure 4a,b, the write throughput of synchronous I/O is degraded significantly for all cases including HDD, RAMDisk, and PCMDisk. When comparing NVM and HDD, the performance degradation in NVM is relatively minor than HDD. This is because writing to HDD is even slower than writing to NVM. Specifically, the performance degradation of HDD is the largest with small random writes when adopting synchronous I/O.

Now, let us see the effect of direct I/O. As the storage becomes sufficiently fast by adopting NVM, one may wonder if direct I/O will perform better than conventional buffered I/O. To answer this question, we investigate the performance of direct I/O that does not use the buffer cache.

In case of HDD, the performance is degraded significantly for all kinds of operations when the direct I/O is used. This is consistent with the expectations as HDD is excessively slower than DRAM. Now, let us see the direct I/O performance of NVM. When the storage is the RAMDisk, the effect of direct I/O contrasts among different operations. Specifically, the performance of the RAMDisk is degraded for read operations, but it is improved for write operations under direct I/O. Note that the physical access time of DRAM and RAMDisk is identical but using the buffered I/O still improves the

read performances because of the software overhead in I/O operations. That is, accessing data from the buffer cache is faster than accessing the same data from I/O devices although the buffer cache and the I/O device have identical performance characteristics. Unlike the read case, the write performance of RAMDisk is improved with the direct I/O. This implies that write performance is degraded although we use the buffer cache. In fact, writing data to the buffer cache incurs additional overhead as it should be written to both memory and storage. Note that additional writing to memory is not a burden when storage is excessively slow like HDD, but that is not case for the RAMDisk.

However, in the case of the PCMDisk, the buffer cache is also effective in write requests as shown in Figure 4. That is, the baseline (with buffer cache) performs 40% better than direct I/O in write operations with PCMDisk. Based on this study, we can conclude that the buffer cache is still effective in fast NVM storage. However, in the case of the RAMDisk, it degrades the storage performances for write operations. Thus, for adopting the buffer cache more efficiently, read requests may be cached but write requests may be set to bypass the buffer cache layer as storage access time approaches the memory access time.

To investigate the efficiency of the buffer cache in more realistic situations, we capture system call traces while executing a couple of representative I/O benchmarks, and perform replay experiments as the storage device changes. Specifically, file request traces were collected while executing two popular Filebench workloads: proxy and web server. Then, we evaluate the total storage access time with and without buffer cache for the given workloads. In this experiment, the cache replacement policy is set to LRU (Least Recently Used), which is most commonly used in buffer cache systems. Note that LRU removes the cached item whose access time is the oldest among all items in the buffer cache when free cache space is needed.

Figure 5 shows the normalized storage access time when the underlying storage is HDD and the buffer cache size is varied from 5% to 50% of the maximum cache usage of the workload. The cache size of 100% implies the configuration that complete storage accesses within the workload can be cached simultaneously, which is equivalent to the infinite cache capacity. This is an unrealistic condition and in practical aspects, the cache size less than 50% indicates most of the real system situations. As shown in the figure, the effect of the buffer cache is very large in HDD storage. In particular, buffer cache improves the storage access time by 70% on average. This significant improvement happens as HDD is excessively slower than buffer cache, and thus reducing the number of storage accesses by caching is very important.

Figure 5. Performance effects of the buffer cache when storage is the HDD.

Figure 6 shows the results with the RAMDisk as the storage medium. As shown in the figure, the effectiveness of the buffer cache is reduced significantly as the storage changes from HDD to RAMDisk. Specifically, buffer cache improves the storage access time by only 3% on average as RAMDisk is adopted. This result clearly shows that as the storage access latency becomes close to that of the buffer cache, the gain from buffer cache decreases largely in realistic situations. However, we perform some additional experiments, and show that the buffer cache is still efficient by managing the cache space appropriately.

Figure 6. Performance effects of the buffer cache when storage is the RAMDisk.

Figure 7 compares the storage access time of two configurations that adopt the same RAMDisk but differentiate the management policy of the buffer cache. Specifically, the LRU-cache inserts all requested file data in the cache and replaces the oldest data when free space is needed. Note that this is the most common configuration of buffer cache, which is also shown in Figure 4. The other configuration, AC-cache, is an admission-controlled buffer cache that allows the admission of file data into the buffer cache only after its second access happens. The rationale of this process is to filter out single-access data, thereby allowing only multiple accessing data to be cached. Although this is a very simple approach, the results in Figure 7 show that AC-cache improves the storage access time by 20–40%. This implies that buffer cache is still effective under fast storage like STT-MRAM, but more judicious management will be necessary to get the maximized performances.

Figure 7. Performance comparison of the buffer cache with Least Recently Used (LRU) and admission-controlled LRU when storage is the RAMDisk.

3.3. Effects of Prefetching

Prefetching is a widely used technique to improve storage system performances. Specifically, prefetching reads not only the currently requested page but also a certain number of adjacent pages in storage, and stores them in the memory. Note that Linux’s prefetching reads up to 32 adjacent pages while handling read I/Os. Prefetching is effective in slow storage systems like HDD as the access latency of HDD is not sensitive to the amount of data to be transferred but storage accesses can be decreased if prefetched data are actually used. Note that file accesses usually show sequential patterns, so prefetching is effective in the HDD storage. In this section, we analyze the effectiveness of prefetching when NVM storage is adopted. As the storage access time becomes close to the memory access time, the effectiveness of prefetching obviously decreases. Furthermore, prefetching may waste the effective memory space if the prefetched data are not actually used.

To review the effectiveness of prefetching in NVM storage, we execute the IOzone benchmark under HDD and NVM storage with/without prefetching and measure the throughput. Figures 8–10 depict the measured throughput of RAMDisk, HDD, and PCMDisk, respectively, with prefetching (denoted as “ra_on”) and without prefetching (denoted as “ra_off”). As shown in Figure 8, prefetching is effective in read performances when the HDD storage is adopted. However, it does not affect the performance of RAMDisk and PCMDisk as shown in Figures 9 and 10. This implies that prefetching is not effective any longer in RAMDisk or PCMDisk.

Figure 8. HDD performance when the prefetching option is turned on/off.

Figure 9. RAMDisk performance when the prefetching option is turned on/off.

Figure 10. PCMDisk performance when the prefetching option is turned on/off.

3.4. Effects of Concurrent Accesses

In modern computing systems, various applications access storage concurrently increasing the randomness of access patterns and aggravating contention for storage. For example, popular social media web servers service a large number of user requests simultaneously, possibly incurring severe contention due to large random accesses generated from concurrent requests. In virtualized systems with a multiple number of guest machines, I/O requests from each guest machine eventually get mixed up generating a series of random storage accesses [24]. Mobile systems like smartphones also face such situations as the mixture of metadata logging by Ext4 and data journaling by SQLite generates random accesses [25]. Database and distributed systems also have similar problems [26].

Thus, supporting concurrent accesses to storage must be an important function of modern storage systems. To explore the effectiveness of NVM under such concurrent accesses, we measure the throughput of HDD, RAMDisk, and PCMDisk as the number of IOzone threads increases. Figure 11 shows the throughput of each storage as the number of threads changes from 1 to 8. As shown in the figure, NVM performs well even though the number of concurrent threads increases exponentially. Specifically, performance improves until the number of threads becomes 8 for most cases. Note that the large number of threads increase access randomness and storage contention. However, we observe that this only has a minor effect on the performance of NVM storage. In contrast, the throughput of HDD degrades as the number of concurrent threads increases.

Figure 11. Performance comparison of HDD, RAMDisk, and PCMDisk varying the number of threads.

In summary, NVM can constitute a contention-tolerable storage system for applications with highly concurrent storage accesses in future computer systems.

3.5. Implications of Alternative PC Configurations

Based on the aforementioned experimental results, this section presents an alternative PC configuration by adopting NVM storage, and discusses the effectiveness of such configurations with respect to the cost, lifespan, and power consumption. Due to the recent advances in multi-core and many-core processor technologies, new types of applications such as interactive rendering and machine learning are also performed in desktop PCs. This is in line with the emerging era of the Fourth Industrial Revolution as well as the recent trend like personalized YouTube broadcasting, where creating high

quality contents or performing computing-intensive jobs is no longer the domain of the expert. Thus, the performance of desktop systems should be improved in accordance with such situations.

While the processor's computing power has improved significantly, it is difficult to increase the memory capacity of desktop PCs due to the scaling limitation of DRAM. Thus, instead of increasing the memory capacity, we argue that NVM storage can provide an alternative solution to supplement the limited DRAM size in emerging applications. That is, a future computer system can be composed of limited memory capacities but has NVM storage devices, not to degrade performances.

Figure 12 shows the architecture of an alternative PC consisting of small DRAM and fast NVM storage in comparison with a traditional PC consisting of large DRAM and slow HDD. Table 2 lists the detailed specifications and estimates of a "large DRAM and slow HDD PC" (hereinafter we will refer to it as "HDD PC") and a "small DRAM and fast NVM PC" (hereinafter we will refer to it as "NVM PC"). By considering emerging desktop workloads such as interactive rendering, big data processing, and PC virtualization, we use a large memory of 64 GB in HDD PC. In contrast, we use the memory size of 16 GB in NVM PC based on the observations in Section 3.1. For NVM storage, we use Intel's Optane SSD, which is the representative NVM product on the market at this time [27]. As shown in Table 2, the total price of NVM PC is similar to that of HDD PC. This is because not only the memory itself but also the mainboard becomes expensive when the memory capacity is very large. A more serious problem is that the memory size is difficult to be extended any longer due to the scaling limitation of DRAM. When the market of NVM storage fully opens, it can be anticipated that the price of NVM will become more competitive. Even though an auxiliary HDD is added to NVM PC, it will not significantly affect the total price.

Figure 12. An alternative PC consisting of small DRAM and fast NVM storage in comparison with the traditional PC consisting of large DRAM and slow HDD.

Now, let us discuss the lifespan issue of NVM. The technical specifications of Optane 900p indicate that the endurance rating is 8.76 PBW (peta bytes written), which implies that we can write 8.76 PB into this device before we need to replace it. As its warranty period is specified to be 5 years, if we evaluate the average amount of daily writing allowed, it is $8.76 \text{ PB} / (5 \text{ years} \times 365 \text{ days/year}) = 4.9152 \text{ TB per day}$. As the size of the Optane 900p product is 280 GB or 480 GB, this implies that writing more than 10 times to the full storage capacity every day is allowed for 5 years. When we compare it with flash-based SSDs, we do not need to consider the endurance problem in NVM storage devices.

Table 2. Specifications of a “large DRAM and slow HDD PC” and a “small DRAM and fast NVM PC”.

Component	16 GB DRAM + NVM PC	64 GB DRAM + HDD PC	Note
CPU	\$317 ⁽¹⁾	\$317 ⁽¹⁾	(1) Intel Core i7-7700 3.6 GHz 4-Core Processor
Mainboard	\$56 ⁽²⁾	\$114 ⁽³⁾	(2) ASRock H110M-HDS R3.0 Micro ATX LGA1151 Motherboard (3) ASRock B250 Pro4 ATX LGA1151 Motherboard
DRAM	\$70 ⁽⁴⁾	\$280 ⁽⁵⁾	(4) Corsair Vengeance LPX 16 GB (5) Corsair Vengeance LPX 16 GB (16 GB × 4)
Storage	\$370 ⁽⁶⁾	\$50 ⁽⁷⁾	(6) Intel Optane 900P (280 GB) (7) Seagate Barracuda Compute HDD 2 TB
Misc (CPU Cooler, Case, Power)	\$156	\$156	
Total Price	\$969	\$917	

Now, let us consider the power issue. We analyze the power consumptions of memory and storage separately and then accumulate them. The memory power consumption P_{DRAM} is calculated as

$$P_{DRAM} = P_{DRAM_static} + P_{DRAM_active} \quad (1)$$

where

$P_{DRAM_static} = Unit_static_power$ (W/GB) × $Memory_size$ (GB) and,

$P_{DRAM_active} = Read_energy_{DRAM}$ (J) × $Read_freq_{DRAM}$ (/s) + $Write_energy_{DRAM}$ (J) × $Write_freq_{DRAM}$ (/s).

$Unit_static_power$ is the static power per capacity including both leakage power and refresh power, and $Read_energy_{DRAM}$ and $Write_energy_{DRAM}$ refer to the energy required for read and write operations for the unit size, respectively. $Read_freq_{DRAM}$ and $Write_freq_{DRAM}$ are the frequencies of read and write operations for the unit time, respectively.

The NVM power consumption P_{NVM} is calculated as

$$P_{NVM} = P_{NVM_idle} + P_{NVM_active} \quad (2)$$

where

$P_{NVM_idle} = Idle_rate_{NVM} \times Idle_power_{NVM}$ (W) and,

$P_{NVM_active} = Read_energy_{NVM}$ (J) × $Read_freq_{NVM}$ (/s) + $Write_energy_{NVM}$ (J) × $Write_freq_{NVM}$ (/s).

$Idle_rate_{NVM}$ is the percentage of time that no read or write occurs on NVM and $Idle_power_{NVM}$ is the power consumed in NVM during idle time. $Read_energy_{NVM}$ and $Write_energy_{NVM}$ refer to the energy required for read and write operations for the unit size, respectively. $Read_freq_{NVM}$ and $Write_freq_{NVM}$ are the frequencies of read and write operations for the unit time, respectively.

The HDD power consumption P_{HDD} is calculated as

$$P_{HDD} = P_{HDD_idle} + P_{HDD_active} + P_{HDD_standby} \quad (3)$$

where

$P_{HDD_idle} = Idle_rate_{HDD} \times Idle_power_{HDD}$ (W),

$P_{HDD_active} = Active_rate_{HDD} \times Active_power_{HDD}$ (W), and

$P_{HDD_standby} = Transition_freq_{HDD}$ (/s) × ($Spin_down_energy_{HDD}$ (J) + $Spin_up_energy_{HDD}$ (J)).

$Idle_rate_{HDD}$ is the percentage of time that no read or write occurs on HDD and $Idle_power_{HDD}$ is the power consumed in HDD during idle states. $Active_rate_{HDD}$ and $Active_power_{HDD}$ refer to the percentage of time that read or write occurs on HDD and the power consumed in HDD during active states, respectively. $Transition_freq_{HDD}$ is the frequency of state transitions between idle and standby states for the unit time. $Spin_down_energy_{HDD}$ and $Spin_up_energy_{HDD}$ refer to the energy required for spin_down and spin_up operations, respectively.

The static power is consumed consistently in DRAM regardless of any operations as DRAM memory cells store data in small capacitors that lose their charge over time and must be refreshed. As the size of DRAM increases, this static power keeps increasing to sustain refresh cycles to retain its data. However, this is not required in NVM because of its non-volatile characteristics. Active power consumption, on the other hand, refers to the power dissipated when data is being read and written.

Figure 13 shows the normalized power consumption of the NVM PC in comparison with the HDD PC as the I/O rate is varied. As shown in the figure, the NVM PC, which uses small DRAM capacity along with NVM, consumes less power than the HDD PC. This is because the static power of DRAM is proportional to its capacity, and thus the reduced size of DRAM saves the static power of DRAM. Specifically, power-savings of the NVM PC becomes large as the I/O rate decreases. When the I/O rate decreases, the static power consumption required for DRAM refresh operations accounts for a large portion of the total power consumption compared to the active power consumption required for actual read/write operations. The average power consumption of memory and storage layers by adopting the NVM PC is 42.9% of the HDD PC in our experiments. This result indicates that NVM PC will be effective in reducing the power consumption of future computer systems. In summary, we showed that the new memory-storage configuration consisting of small memory and fast storage could be an alternative solution for ever-growing memory demands and the limited scalability problem of DRAM memory.

Figure 13. Comparison of power consumption.

4. Related Work

As NVM can be adopted in both memory and storage subsystems, a spectrum of studies have been conducted to explore the performance of systems making use of NVM as the main memory [6,13,20,21,28,29] and/or storage [30–35]. Mogul et al. propose a novel memory management technique for systems making use of DRAM and NVM hybrid memory. Their technique allocates pages to DRAM or NVM by utilizing the page's attributes, such that read-only pages are placed on PCM, whereas writable pages are placed on DRAM in order to decrease the write traffic to PCM [28]. Querishi et al. present a PCM based main memory system and adopt DRAM as PCM's write buffer to increase the lifespan of PCM as well as accelerating PCM's write performances [6]. Lee et al. propose a PCM based main memory system and try to enhance the PCM's write performance by judiciously managing the last level cache memory. Specifically, they present the partial writing and the buffer reorganizing schemes for tracking the change of data in the last level cache and flush only the changed part to PCM memory [21,29]. Lee et al. present a new page replacement algorithm for memory systems being composed of PCM and DRAM [13]. Their algorithm performs online estimation of page attributes and places write-intensive pages on DRAM and read-intensive pages on PCM. Kultursay

et al. propose an STT-MRAM based main memory system for reducing the power consumption of the main memory [20]. Their system considers the entire replacement of DRAM to STT-MRAM and exhibits that the performance of STT-MRAM based main memory is competitive to that of DRAM with the 60% energy-saving effect in the memory.

Now, let us discuss the studies focusing on NVM as storage devices. In early days, the size of NVM was very small, and thus NVM acts as a certain part of the storage systems. NEB [32], MRAMFS [33], and PRAMFS [34] are examples of such cases. They aim to maintain hot data or important data such as metadata on NVM [34]. As the size of NVM was small, they also focused on the efficiency of the NVM storage spaces. For example, Edel et al. [33] try to save NVM spaces by making use of compression, whereas Baek et al. [32] perform space saving by extent-based data management.

As the density of NVM increases, studies on NVM storage suggest the file system design that maintains the total file system in NVM. Baek et al. present a system layer that acts as the memory as well as the storage by making use of NVM for managing both memory and file objects [35]. Condit et al. present BPFS, which is a new file system for NVM exploiting the byte-addressable characteristics of NVM [30]. Specifically, BPFS overwrites data if the write size is small enough for atomic operations, thereby reducing the writing overhead of NVM without compromising reliability. Similarly, Wu et al. propose a byte-addressable file system for NVM, which does not need to pass through the buffer cache layer by allocating memory address spaces to the location of the NVM file system [31]. Coburn et al. propose a persistent in-memory data storage consisting of NVM, which can be created by users [36]. Volos et al. propose a new NVM interface that allows the creation and management of NVM without the risk of inconsistency from system crashes [37]. Specifically, it enables users to define persistent data with the given primitives, which is automatically managed by transaction, and thus users do not need to concern the inconsistency situations from system failures.

Recently, studies on NVM focus on the efficiency of I/O paths and software stacks that are designed originally for slow storage like HDD rather than NVM storage. Yang et al. argue that synchronous I/O performs better than asynchronous I/O if storage becomes very fast [38]. Caulfield et al. quantify the overhead of each software stack in I/O operations and propose efficient interfaces for fast NVM storage systems [39]. Our study is different from these studies in that we evaluate the NVM storage performance, in particular for PCM and STT-MRAM based storage systems, for a wide range of operating system settings and modes, while they focus on a detailed examination of a typical aspect of the storage access path.

5. Conclusions

In this article, we revisited the memory management issues specially focusing on the storage I/O mechanisms when the underlying storage is NVM. As the storage performance becomes sufficiently fast by adopting NVM, we found out that reassessment of the existing I/O mechanisms is needed. Specifically, we performed a broad range of measurement studies to observe the implications of using NVM based storage on existing I/O mechanisms. Based on our study, we made five observations. First, the performance gain of NVM based storage is limited due to the existence of the buffer cache, but it becomes large as we reduce the memory size. Second, buffer caching is still effective in fast NVM storage even though its effectiveness is limited. Third, prefetching is not effective when the underlying storage is NVM. Fourth, synchronous I/O and direct I/O do not affect the performance of storage significantly under NVM storage. Fifth, performance degradation due to the contention of multi-threads is less severe in NVM based storage than in HDD. Based on these observations, we discussed how to address rapidly growing memory demands in emerging applications by adopting NVM storage. Specifically, we analyzed the cost of a system consisting of a small memory and fast NVM storage in comparison with the traditional PC consisting of a large memory and slow storage. We showed that the memory-storage configuration consisting of a small memory and fast storage could be an alternative solution for ever-growing memory demands and the limited scalability of DRAM memory.

In the future, we will study some other issues related to memory and storage layers such as security and reliability when NVM is adopted as the storage of emerging computer systems. We anticipate that our findings will be effective in designing future computer systems with fast NVM based storage.

Author Contributions: K.C. implemented the architecture and algorithm, and performed the experiments. H.B. designed the work and provided expertise. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2019R1A2C1009275) and also by the ICT R&D program of MSIP/IITP (2019-0-00074, developing system software technologies for emerging new memory that adaptively learn workload characteristics).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ng, S. Advances in disk technology: Performance issues. *Computer* **1998**, *31*, 75–81. [[CrossRef](#)]
2. Evans, C. Flash vs 3D Xpoint vs storage-class memory: Which ones go where? *Comput. Wkly.* **2018**, *3*, 23–26.
3. Stanisavljevic, M.; Pozidis, H.; Athmanathan, A.; Papandreou, N.; Mittelholzer, T.; Eleftheriou, E. Demonstration of Reliable Triple-Level-Cell (TLC) Phase-Change Memory. In Proceedings of the of the 8th IEEE International Memory Workshop, Paris, France, 15–18 May 2016; pp. 1–4.
4. Zhang, W.; Li, T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In Proceedings of the of the 18th IEEE International Conference on Parallel Architectures and Compilation Techniques (PACT), Raleigh, NC, USA, 12–16 September 2009; pp. 101–112.
5. Lee, E.; Jang, J.; Kim, T.; Bahn, H. On-demand Snapshot: An Efficient Versioning File System for Phase-Change Memory. *IEEE Tran. Knowl. Data Eng.* **2012**, *25*, 2841–2853. [[CrossRef](#)]
6. Qureshi, M.K.; Srinivasan, V.; Rivers, J.A. Scalable high performance main memory system using phase-change memory technology. In Proceedings of the 36th International Symposium on Computer Architecture (ISCA), Austin, TX, USA, 20–24 June 2009; pp. 24–33.
7. Nale, B.; Ramanujan, R.; Swaminathan, M.; Thomas, T. *Memory Channel that Supports near Memory and Far Memory Access*; PCT/US2011/054421; Intel Corporation: Santa Clara, CA, USA, 2013.
8. Ramanujan, R.K.; Agarwal, R.; Hinton, G.J. *Apparatus and Method for Implementing a Multi-level Memory Hierarchy Having Different Operating Modes*; US 20130268728 A1; Intel Corporation: Santa Clara, CA, USA, 2013.
9. Eilert, S.; Leinwander, M.; Crisenza, G. Phase Change Memory: A new memory technology to enable new memory usage models. In Proceedings of the 1st IEEE International Memory Workshop (IMW), Monterey, CA, USA, 10–14 May 2009; pp. 1–2.
10. Lee, E.; Yoo, S.; Bahn, H. Design and implementation of a journaling file system for phase-change memory. *IEEE Trans. Comput.* **2015**, *64*, 1349–1360. [[CrossRef](#)]
11. Lee, E.; Kim, J.; Bahn, H.; Lee, S.; Noh, S. Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM. *ACM Trans. Storage* **2017**, *13*, 1–13. [[CrossRef](#)]
12. Lee, E.; Bahn, H.; Yoo, S.; Noh, S.H. Empirical study of NVM storage: An operating system's perspective and implications. In Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Paris, France, 9–11 September 2014; pp. 405–410.
13. Lee, S.; Bahn, H.; Noh, S.H. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. *IEEE Trans. Comput.* **2014**, *63*, 2187–2200. [[CrossRef](#)]
14. Kwon, O.; Koh, K.; Lee, J.; Bahn, H. FeGC: An Efficient Garbage Collection Scheme for Flash Memory Based Storage Systems. *J. Syst. Softw.* **2011**, *84*, 1507–1523. [[CrossRef](#)]
15. Weis, N.; Wehn, L.; Igor, I.; Benini, L. Design space exploration for 3d-stacked drams. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Grenoble, France, 14–18 March 2011; pp. 1–6.
16. Elliot, J.; Jung, E.S. Ushering in the 3D Memory Era with V-NAND. In Proceedings of the Flash Memory Summit, Santa Clara, CA, USA, 12–15 August 2013.

17. Wright, C.D.; Aziz, M.M.; Armand, M.; Senkader, S.; Yu, W. Can We Reach Tbit/sq.in. Storage Densities with Phase-Change Media? In Proceedings of the European Phase Change and Ovonic Symposium (EPCOS), Grenoble, France, 29–31 May 2006.
18. Bedeschi, F.; Fackenthal, R.; Resta, C.; Donz, E.M.; Jagasivamani, M.; Buda, E.; Pellizzer, F.; Chow, D.W.; Cabrini, A.; Calvi, G.M.A.; et al. A multi-level-cell bipolar-selected phase-change memory. In Proceedings of the 2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 3–7 February 2008.
19. Zhang, Y.; Wen, W.; Chen, Y. STT-RAM Cell Design Considering MTJ Asymmetric Switching. *SPIN* **2012**, *2*, 1240007. [[CrossRef](#)]
20. Kultursay, E.; Kandemir, M.; Sivasubramaniam, A.; Mutlu, O. Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative. In Proceedings of the IEEE International Symposium Performance Analysis of Systems and Software (ISPASS), Austin, TX, USA, 21–23 April 2013; pp. 256–267.
21. Lee, B.C.; Ipek, E.; Mutlu, O.; Burger, D. Architecting phase change memory as a scalable DRAM alternative. In Proceedings of the 36th ACM/IEEE International Symposium on Computer Architecture (ISCA), Austin, TX, USA, 20–24 June 2009.
22. PCMSim. Available online: <http://code.google.com/p/pcmsim> (accessed on 18 January 2020).
23. Norcott, W. IOzone Filesystem Benchmark. Available online: <http://www.iozone.org/> (accessed on 1 October 2019).
24. Tarasov, V.; Hildebrand, D.; Kuenning, G.; Zadok, E. Virtual Machine Workloads: The Case for New NAS Benchmarks. In Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST), 12–15 February 2013; pp. 307–320.
25. Lee, K.; Won, Y. Smart layers and dumb result: IO characterization of an Android-based smartphone. In Proceedings of the International Conference on Embedded Software (EMSOFT), Tampere, Finland, 7–12 October 2012; pp. 23–32.
26. Stonebraker, M.; Madden, S.; Abadi, D. The End of an Architectural Era (It's Time for a Complete Rewrite). In Proceedings of the 33rd Very Large Data Bases Conference (VLDB), Viena, Austria, 23–27 September 2007.
27. Intel® SSD Client Family. Available online: <https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/consumer-ssds.html> (accessed on 18 January 2020).
28. Mogul, J.C.; Argollo, E.; Shah, M.; Faraboschi, P. Operating system support for NVM+DRAM hybrid main memory. In Proceedings of the 12th USENIX Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland, 18–20 May 2009; pp. 4–14.
29. Lee, B.C.; Ipek, E.; Mutlu, O.; Burger, D. Phase change memory architecture and the quest for scalability. *Commun. ACM* **2010**, *53*, 99–106. [[CrossRef](#)]
30. Condit, J.; Nightingale, E.B.; Frost, C.; Ipek, E.; Lee, B.; Burger, D.; Coetzee, D. Better I/O through byte-addressable, persistent memory. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Big Sky, MT, USA, 11–14 October 2009; pp. 133–146.
31. Wu, X.; Reddy, A.L.N. SCMFS: A File System for Storage Class Memory. In Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, WA, USA, 12–18 November 2011.
32. Baek, S.; Hyun, C.; Choi, J.; Lee, D.; Noh, S.H. Design and analysis of a space conscious nonvolatile-RAM file system. In Proceedings of the IEEE Region 10 Conference (TENCON), Hong Kong, China, 14–17 November 2006.
33. Edel, N.K.; Tuteja, D.; Miller, E.L.; Brandt, S.A. MRAMFS: A compressing file system for non-volatile RAM. In Proceedings of the 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Volendam, The Netherlands, 4–8 October 2004; pp. 569–603.
34. PRAMFS. Available online: <http://pramfs.sourceforge.net> (accessed on 1 October 2019).
35. Baek, S.; Sun, K.; Choi, J.; Kim, E.; Lee, D.; Noh, S.H. Taking advantage of storage class memory technology through system software support. In Proceedings of the Workshop on Interaction between Operating Systems and Computer Architecture (WIOSCA), Beijing, China, June 2009.
36. Coburn, J.; Caulfield, A.; Akel, A.; Grupp, L.; Gupta, R.; Swanson, S. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Newport Beach, CA, USA, 5–11 March 2011; pp. 105–118.

37. Volos, H.; Tack, A.J.; Swift, M.M. Mnemosyne: Lightweight persistent memory. In Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Newport Beach, CA, USA, 5–11 March 2011; pp. 91–104.
38. Yang, J.; Minturn, D.B.; Hady, F. When poll is better than interrupt. In Proceedings of the USENIX Conference File and Storage Technologies (FAST), San Jose, CA, USA, 14–17 February 2012.
39. Caulfield, A.M.; De, A.; Coburn, J.; Mollov, T.I.; Gupta, R.K.; Swanson, S. Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories. In Proceedings of the IEEE/ACM International Symposium on Microarchitecture, Northwest, Washington, DC, USA, 4–8 December 2010; pp. 385–395.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).