A New Memory-Disk Integrated System with HW Optimizer

DO-HEON LEE, SU-KYUNG YOON, and JUNG-GEUN KIM, Yonsei University
CHARLES C. WEEMS, University of Massachusetts
SHIN-DUG KIM, Yonsei University

11

Current high-performance computer systems utilize a memory hierarchy of on-chip cache, main memory, and secondary storage due to differences in device characteristics. Limiting the amount of main memory causes page swap operations and duplicates data between the main memory and the storage device. The characteristics of next-generation memory, such as nonvolatility, byte addressability, and scaling to greater capacity, can be used to solve these problems. Simple replacement of secondary storage with new forms of nonvolatile memory in a traditional memory hierarchy still causes typical problems, such as memory bottleneck, page swaps, and write overhead. Thus, we suggest a single architecture that merges the main memory and secondary storage into a system called a Memory-Disk Integrated System (MDIS). The MDIS architecture is composed of a virtually decoupled NVRAM and a nonvolatile memory performance optimizer combining hardware and software to support this system. The virtually decoupled NVRAM module can support conventional main memory and disk storage operations logically without data duplication and can reduce write operations to the NVRAM. To increase the lifetime and optimize the performance of this NVRAM, another hardware module called a Nonvolatile Performance Optimizer (NVPO) is used that is composed of four small buffers. The NVPO exploits spatial and temporal characteristics of static/dynamic data based on program execution characteristics. Enhanced virtual memory management and address translation modules in the operating system can support these hardware components to achieve a seamless memory-storage environment. Our experimental results show that the proposed architecture can improve execution time by about 89% over a conventional DRAM main memory/HDD storage system, and 77% over a state-of-the-art PRAM main memory/HDD disk system with DRAM buffer. Also, the lifetime of the virtually decoupled NVRAM is estimated to be 40% longer than that of a traditional hierarchy based on the same device technology.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General

General Terms: Design, Performance

Additional Key Words and Phrases: Emerging technologies, mass storage, memory control and access, virtual memory

ACM Reference Format:
Do-Heon Lee, Su-Kyung Yoon, Jung-Geun Kim, Charles C. Weems, and Shin-Dug Kim. 2015. A new memory-disk integrated system with HW optimizer. ACM Trans. Architec. Code Optim. 12, 2, Article 11 (May 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2738053

This work was supported by an Industry-Academy joint research program between Samsung Electronics and Yonsei University.
Authors' addresses: D.-H. Lee, S.-K. Yoon, J.-G. Kim, and S.-D. Kim, Department of Computer Science, Yonsei University, Seoul, Korea; emails: {intovortex, sk.yoon, junggeun, sdkim}@yonsei.ac.kr; C. C. Weems, School of Computer Science, University of Massachusetts, Amherst, MA 01003-4610; email: weems@cs.umass.edu.

DOI: http://dx.doi.org/10.1145/2738053

1. INTRODUCTION

Traditional high-performance computer memory systems are hierarchically composed of, for example, independent units of on-chip Static RAM (SRAM), Dynamic RAM (DRAM) main memory, and secondary storage, such as a Hard Disk Drive (HDD) and/or a NAND flash-based Solid State Drive (SSD). Conventional virtual memory logically extends main memory to secondary storage in a manner that causes data duplication and page swap operations. Power consumption and capacity demands on conventional limited main memory are steadily increasing. Moreover, the volatility of DRAM necessitates a boot-up process to restore its contents after various levels of power-down. Conventional secondary storage devices are also too slow to act as a virtual extension of DRAM main memory, given frequent accesses.

To overcome these drawbacks of the basic components of a conventional memory hierarchy system, the next generation of nonvolatile memory, such as Phase-change RAM (PRAM), Ferroelectric RAM (FeRAM), and Magnetic RAM (MRAM), have some beneficial characteristics compared with DRAM, such as nonvolatility, byte-addressability, and greater capacity for scaling. According to recent studies [Zhou et al. 2009; Lee et al. 2009; Park et al. 2010; Yoon et al. 2014; Jang et al. 2014], PRAM is a good candidate for mass production and has the potential to at least partially replace DRAM main memory and NAND flash SSD. Most next-generation nonvolatile memories also consume less power than conventional volatile memories. For these reasons, this study investigates a new architecture based on PRAM device technology.

However, because of the slow read/write latency of PRAM, it is not effective to use it directly as main memory. Also, its limited write endurance can be quickly exhausted at main memory access rates. Furthermore, using PRAM as a replacement for either main memory or secondary storage does not solve the problems of data duplication and swapping overhead. Thus, we present a new memory architecture to exploit the characteristics of nonvolatile memory, particularly PRAM. Our study merges conventional main memory and secondary storage layers into a single memory layer using PRAM device technology.

The new memory hierarchy system is called a Memory-Disk Integrated System (MDIS). In this approach, data are stored in the nonvolatile MDIS and accessed in place as if they were in a conventional DRAM main memory without duplications occurring between layers as before. Thus, conventional file access and memory access mechanisms must be changed at the operating system (OS) layer to be transparent to the new MDIS. The overall MDIS architecture is designed around a fetch-block-based M-D Nonvolatile RAM performance optimizer (NVPO), a virtually decoupled Nonvolatile RAM (NVRAM) module, and its associated virtual memory management module in the OS. In this article, we discuss the two hardware components, the virtually decoupled NVRAM module, and the NVPO, along with the software mechanism for seamless address translation. The proposed virtually decoupled NVRAM module is designed as two virtual spaces, static and dynamic. Static space can be read directly, whereas dynamic space operates like conventional main memory and contains, for example, the heap segment and stack segment. To overcome the characteristic asymmetry of read/write access latency and increase the NVRAM's lifetime, the fetch-block-based NVPO consists of four decoupled buffers located between the on-chip cache and the virtually decoupled NVRAM module. These buffers are designed to aggressively utilize spatial and temporal locality to support virtualization of the static and dynamic regions by adapting to the characteristics of the executing code.

The proposed structure is evaluated by using a trace-based simulator with SPEC 2006 [Henning 2006; SPEC CPU 2006; Phansalkar et al. 2007], SPLASH-2 [Woo et al. 1995], grep [Singer et al. 2011], and PostMark [Katcher 1997] traces. According to

our simulation results, execution time can be reduced by about 89% compared to a conventional hierarchy of DRAM main memory and HDD secondary storage and 77% over a set of DRAM buffers/PRAM main memory/HDD disk pair proposed as state-of-the-art proposals [Qureshi et al. 2009]. Moreover, eliminating page swap operations can increase the lifetime of the nonvolatile memory by about 40%. Overall, the execution speed of our proposed system is 4.5× faster than that of a PRAM-based main memory and storage system, and it is 1.1× faster than a DRAM-based main memory system with PRAM secondary storage.

The rest of this article is organized as follows. Section 2 presents other work related to adapting nonvolatile memory for main memory using buffer structures to reduce access latency and exploit NVRAM efficiently. Section 3 presents the proposed memory-disk integrated system and its management algorithm. Section 4 analyzes the performance of this system. Finally, we provide conclusions in Section 5.

2. RELATED WORK

Many studies have investigated the use of nonvolatile memory in an existing memory hierarchy, including disk cache and write buffers over NAND flash memory, hybrid main memory systems, and homogeneous nonvolatile memory structures. In addition, several commercial storage models using nonvolatile memory also have been proposed such as Memory Channel Storage (MCS) architecture placing nonvolatile memory on memory bus directly [Diablo Technologies 2015]. In this section, a new memory and storage architecture using PRAM will be reviewed briefly along with work on using off-chip cache memory to reduce access and increase the performance and lifetime of nonvolatile memories.

PRAM, which is commercially available, is being variously applied to existing memory systems ranging from embedded systems to high-end computing systems [Bedeschi et al. 2008]. Ferreira et al. [2010] proposed a new memory system called PMMA to effectively use PRAM for main memory in next-generation embedded systems. In this architecture, PRAM is used both as a storage medium and as a main memory component. Because the density of PRAM is considerably higher than that of DRAM, it can be used as a backing store in many embedded computing applications.

Qureshi et al. [2009] proposes a PRAM-based hybrid main memory system coupled with a small DRAM buffer. In this architecture, the DRAM buffer acts as a cache with respect to the PRAM to hide the read latency. A write buffer is used to hide the write latency. In addition to the write buffer, fine-grained wear-leveling is proposed to extend the endurance limits of PRAM, which employs a line-size write unit. The PRAM-based hybrid main memory architecture is composed of a 32GB PRAM with a 1GB DRAM buffer. The hybrid main memory shows results similar to those of a 32GB DRAM.

Jiang et al. [2012] presents architectural innovations to overcome long write latency in MLC PRAM. MLC PRAM commonly uses an iterative program-and-verify (P&V) write scheme for program multibits per cell, which causes large write latency. In this system, the authors propose Write Truncation (WT) for identifying difficult-to-write cells, which need more iterations to write, and truncating last iterations. Because WT terminates the write operations of difficult-to-write cells, those cells remain in a temporal error state. To correct this soft error, the authors also present an extra Error Correction Code (ECC). This architecture reduces the read/write latencies by 57% and 28%, respectively, and improves the performance by 26%. Sun et al. [2011] propose a frequent-value-based PRAM memory architecture to mitigate the write intensity to PRAM main memory and improve the endurance and write energy. The authors explore frequent-value locality in data written on PRAM main memory. In this system, frequent-values are identified, and these data are stored in a compressed pattern with an FV bit, indicating whether the data are compressed or original, in order to reduce the

write intensity to PRAM. Original data are stored in data blocks in PRAM main memory without any encoding. By using this scheme, PRAM endurance of the frequent-value storage architecture is enhanced by 4× on average, and the write energy is improved by 27%. Chang et al. [2014] improves the capability of PRAM for use as both main memory and storage and enhances the performance of the whole system. The authors propose a PRAM translation layer to jointly manage main memory space and storage space in the PRAM device. Moreover, using three device commands, FREE, LINK, and UNLINK, the proposed system reduces I/O operations between main memory and storage. Additionally, the authors show the possibility of improving system performance by eXecute-In-Place (XIP).

Beyond architectures using PRAM as the basic component of main memory, there is relevant research to reduce the access latency of main memory with an off-chip cache and to integrate the main memory and storage layers with nonvolatile memories. Zhang et al. [2004] proposed a new memory hierarchy that uses cached DRAM to construct a large, low-overhead, off-chip cache. The cached DRAM is composed of a large DRAM space to hold large working sets and a small SRAM space to exploit the spatial locality in L2 miss streams in order to reduce access latency. Their cached DRAM consisted of 64MB of DRAM with 128KB of on-chip cache, and their performance results indicate better execution time than an 8MB off-chip cache alone.

Jung and Cho [2013] conceptually integrated main memory and storage with a Single-Level Cell (SLC) and a Multilevel Cell (MLC) of nonvolatile memory. SLC nonvolatile memory can be utilized as a main memory, and MLC nonvolatile memory can be used as secondary storage. They use a memory resource controller, located between the OS virtual memory management layer and the physical nonvolatile memory, to dynamically adjust the space of the main memory and secondary storage. Memory can be extended by the resource controller when a process requests a large memory allocation. Their approach avoids many, but not all, of the page swap operations in the memory.

Other approches related to NVRAM application are to exploit various nonvolatile memory technologies such as STT-RAM. Xue et al. [2011] introduces emerging NVRAM technologies, their potential challenges, and their new opportunities. The authors present PRAM as an attractive replacement of DRAM and STT-RAM as alternatives to L2/L3 cache in a chip multiprocessor. The much better scalability of PRAM can overcome the scaling challenge DRAM is facing. Thus, several studies have been recently under way to construct hybrid main memory systems using both PRAM and DRAM. However, limited endurance and asymmetric read/write latencies of PRAM must still be overcome. As another approach, the authors introduce a L2/L3 cache system using STT-RAM. By using the advantages of STT-RAM, such as low leakage and fast read speed, it can replace the L2/L3 cache in a CMP. However, because STT-RAM also has long write latency, the system must use a write buffer scheme to overcome this drawback.

Last, OS-level studies have been conducted to exploit NVRAM as both main memory and storage. Rudoff [2013] introduces four NVM programming models: NVM block mode, NVM file mode, PM volume mode, and PM file mode. And Wu and Reddy [2011] propose a Storage Class Memory (SCM) file system, called SCMFS, for storage class memory. An SCM device is directly attached to the memory bus and shares critical system resources with the main memory, such as memory bus bandwidth, CPU cache, and TLBs. SCMFS is built on top of the virtual memory space, and the Memory Management Unit (MMU) is utilized for translating file system addresses to physical addresses in the SCM. By reusing memory management infrastructure and simplifying the complexity of the file system, SCMFS minimizes the CPU overhead with respect to file system operation and improves overall system performance. Oikawa [2013] demonstrated memory management for a file system with an integrated memory-disk system.

He changed the entire file system by using the block size of the file system to exploit secondary storage as an extended memory space that can be mapped into a virtual address space. Also, when the OS terminates, all nonvolatile memory pages are returned to the file system to keep the file system consistent for the subsequent OS boot. In effect, the system converts all blocks in the secondary storage layer into memory pages, and the virtual memory system is responsible for managing both main memory space and nonvolatile space. Oikawa [2014] also presents another file system for a byte-addressable memory storage system that could be used as both main memory and storage. Through the library implementation of the file system, the user-level file system can interact with memory storage directly without any intervention by the kernel. This file system shows 1.9× and 6.5× faster read/write performance than current file systems.

3. MAIN ARCHITECTURE

3.1. Overall Architecture

In this research, we introduce the MDIS approach, which integrates the conventional main memory and disk storage layers into a single layer using nonvolatile memory. MDIS storage is classified into two logical spaces, static and dynamic, from the perspective of the OS. The two spaces are mapped and manipulated over the nonvolatile memory. Program files and static data are accessed directly in the static space via the virtual memory management layer, taking advantage of the byte-addressable characteristic of NVRAM. Thus, redundant data duplication between main memory and secondary storage can be avoided. Conventional dynamic allocation and deallocation operations are done in the dynamic space of the NVRAM.

To support this mechanism, virtual memory management and file management modules of the OS need to be modified as follows. First, virtual memory management should allocate virtual addresses in the static space to support a RAM-based file system, such as PRAMFS [SourceForge 2013] or PMFS [GitHub 2013]. Thus, we can support XIP on the static space. If we utilize such a file system, the program will be executed by linking to its associated pages in the static space. Program execution using dynamic libraries doesn't need to move pages into the dynamic space since they are simply linked when the program is executed.

Furthermore, when handling a data file, we can read its contents directly from the static space instead of loading it into the dynamic space. Thus, we can remove data duplication and improve performance on program execution and file accesses. Using a RAM-based file system provides compatibility with conventional OSs by maintaining the conventional file system interface.

Our MDIS approach can avoid the page swapping that occurs in a memory hierarchy because of the elimination of data duplication between main memory and secondary storage. The MDIS consists of two major parts connected by an NVRAM translation layer, as shown in Figure 1. Those two parts are the virtually decoupled NVRAM module and an NVRAM performance optimizer (NVPO). The latter effectively minimizes the performance gap between the lowest cache level and the NVRAM. In this article, we focus on designing these hardware modules and the corresponding virtual memory management module for the OS. The NVRAM translation layer is conceptually described in this article in terms of the basic operational flow.

The NVPO module is composed of four decoupled DRAM buffers intended to hide asymmetric read/write access latencies: an active fetch-block buffer, a dynamic data centric buffer, a static data-centric buffer, and a dynamic data write buffer. The NVPO enhances NVRAM read latency, which is about 5× slower than DRAM, and hides NVRAM write latency that is about 10× slower than DRAM.

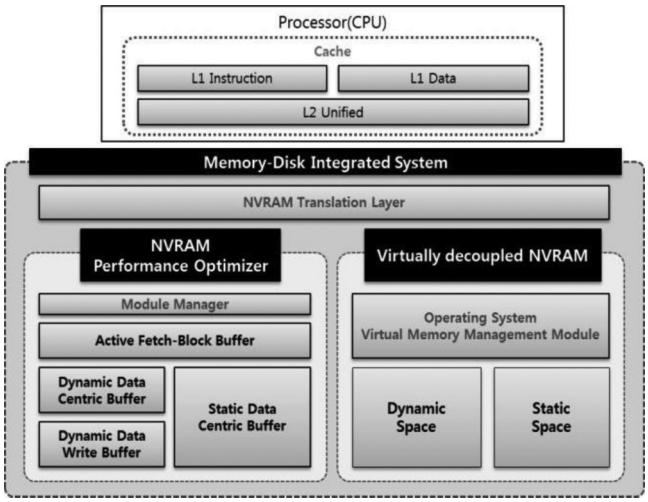

Fig. 1. Overall architecture.

Additionally, extending the lifetime of the NVRAM module is addressed by the combination of a fetch-block-based performance optimizer with new virtual memory and file management modules in the OS. The proposed architecture's two major components are described in the following subsections.

3.2. Virtually Decoupled NVRAM

Virtually decoupled NVRAM is divided into two spaces: dynamic space and static space. Each space can be used like traditional main memory space and file system space. The virtually decoupled NVRAM also supports an NVRAM translation layer to provide an address translation mechanism and basic wear-leveling operations.

A conventional memory hierarchy is constructed of multiple levels of cache, main memory, and secondary storage. Program and data files are typically kept in HDD or SSD and copied to main memory for use by the processor, resulting in duplication. The conventional Executable and Linkable Format (ELF)-based program structure is composed of two major parts, readable static space, including text and data segments, and dynamic space, including stack segments and heap segments.

Figure 2 shows the structure of our NVRAM composed of virtual dynamic and static spaces. The static space keeps track of static data, such as executable and readable data files, as conventional file storage, and it can be accessed as text segments and data segments based on the ELF memory map structure. The dynamic space can be accessed as a conventional main memory in which the OS allocates and deallocates pages associated with heap and stack spaces in the virtual memory map.

The static space can be used to read executable files in place without duplicating data, and the dynamic space can be used to access dynamic and temporal data via conventional memory allocation. Conventional data migration between main memory and storage can thus be avoided. Stack and heap are kept in dynamic space when the process is running. The NVRAM translation layer manages mode bits that indicate the status of each physical NVRAM block and can change the status from dynamic to static. Write-back of dynamic data to the static space is thus handled by changing the mode bit, without any physical migration (write and erase) operation. The translation layer can also manage the capacities of the static and dynamic spaces by changing

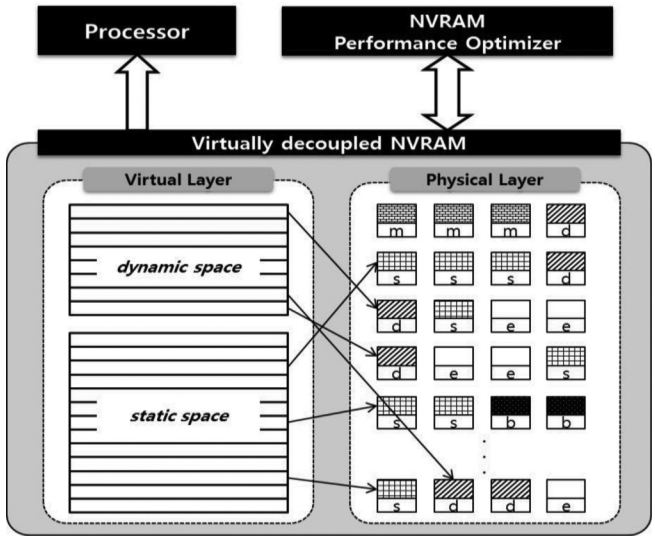

Fig. 2. Structure of virtually decoupled NVRAM.

mode bits. For example, file creation can be handled by changing the mode bit of a given block in the virtually decoupled NVRAM.

As shown in Figure 2, our virtual space and mode bits represent various states of the NVRAM blocks, such as dynamic (D), static (S), empty (E), or bad (B). The OS exploits this information for file and memory management. The file system manages only blocks whose mode bit is marked with the static value. Dynamic space management is performed on blocks with the dynamic value. In other words, the NVRAM translation layer manages the dynamic and static space of the virtually decoupled NVRAM via the mode bits of every block, and it expands and shrinks their capacities dynamically without the need for page swap operations.

3.2.1. Improved Virtual Memory Management Module. The conventional memory hierarchy supports a virtual memory environment that maps to a physical memory (e.g., about 4GB in a 32-bit system). It also can hide the limited capacity of the physical main memory, but doing so may cause frequent page swap operations to storage swap space. In the MDIS architecture, virtual memory mapping of each process can be managed by static and dynamic page linking instead of using only main memory mapping to provide an extended memory space. Thus, the program execution flow changes from a main memory load and execute paradigm to linking up pages in the virtual static space of the NVRAM and using XIP, as shown in Figure 3. Furthermore, the absence of swap space increases the lifetime of the NVRAM.

3.2.2. NVRAM Translation Layer. We designed a memory address translation mechanism for the MDIS translation layer. Figure 3 shows the NVRAM translation layer, consisting of address mapping tables with conceptual relationships for virtual address mapping, logical dynamic and static space mapping, a wear-leveling engine, and NVRAM mapping. The address table is used to manage the status and data types of each block address and to process address translations before accessing the NVRAM. When the program starts execution, a virtual memory map of the running process is created, and the translation layer is updated by modifying the metadata for the virtually decoupled NVRAM. The virtual memory map of the running process can be transformed and mapped into static or dynamic space. Thus, the translation layer guarantees fast

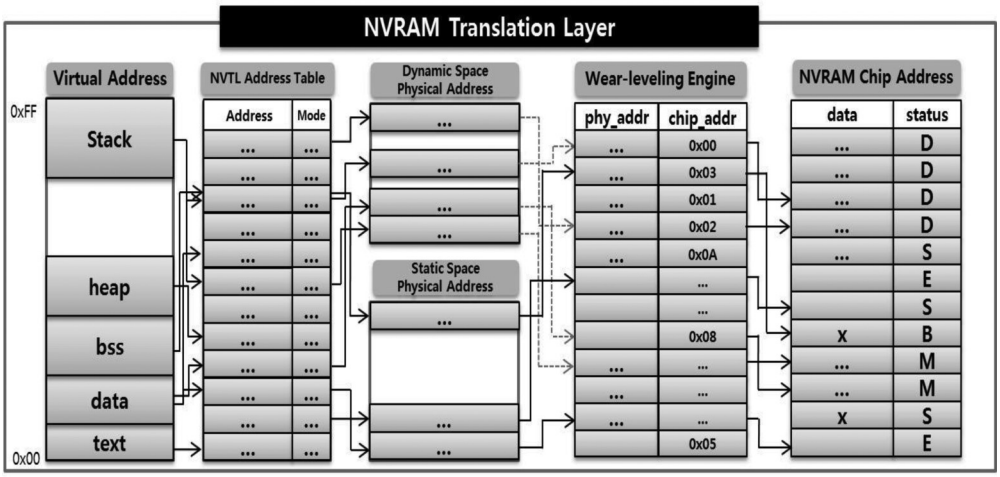

Fig. 3. Memory address translation via improved virtual memory management.

address translation with the support of the address translation table, as it translates the virtual address requested from the processor into the virtually decoupled NVRAM physical address. The virtual memory management module references this information to create a seamless virtual memory map of the processes.

3.3. NVRAM Performance Optimizer

As mentioned before, the NVRAM performance optimizer (NVPO) is composed of four buffers to improve asymmetric read/write latencies, increase lifetime of the NVRAM, and optimize the structure of our proposed system. To demonstrate the effectiveness of the MDIS compared with the conventional memory and disk hierarchy, access latency should be less than or equal to that of a conventional DRAM-disk pair, and long write latency should be hidden or reduced through an effective buffer design. An Active Fetch-Block Buffer (AFBB) is used to fetch blocks by fetch-block unit (8KB), where the fetch-block unit has the size of two pages. The AFBB exploits spatial locality in the program to increase data availability and minimizes the long access latency of the virtually decoupled NVRAM by hiding and reducing relatively slow read/write operations. Also, the Static Data Centric Buffer (SDCB) and Dynamic Data Centric Buffer (DDCB) are used to store evicted fetch-blocks from the AFBB, thereby enhancing temporal locality. Finally, to hide and delay write latency to the NVRAM, a Dynamic Data Write Buffer (DDWB) is designed to store evicted reusable fetch-blocks from the DDCB and delay any write operation to hide slow write latency and maximize the lifetime of the NVRAM. As shown in Figure 4, we have designed an effective structure for the NVPO, composed of 2MB of DRAM, to minimize cost and reduce power consumption.

3.3.1. Active Fetch-Block Buffer. The AFBB provides 1MB of space to hold the 8KB fetch-blocks. It is designed to support high fetch accuracy and accessing performance. As shown in Table I, the AFBB management table maintains the status information required for each fetch-block and is used to select a victim fetch-block when loading a new fetch-block by using a first-in, first-out (FIFO) algorithm. Each fetch-block is divided into two reusable 4KB block units, corresponding to the unit of management in the SDCB or DDCB. Depending on its status, a victim fetch-block is migrated into SDCB or DDCB without verifying the dirty flag, to provide additional temporal locality.

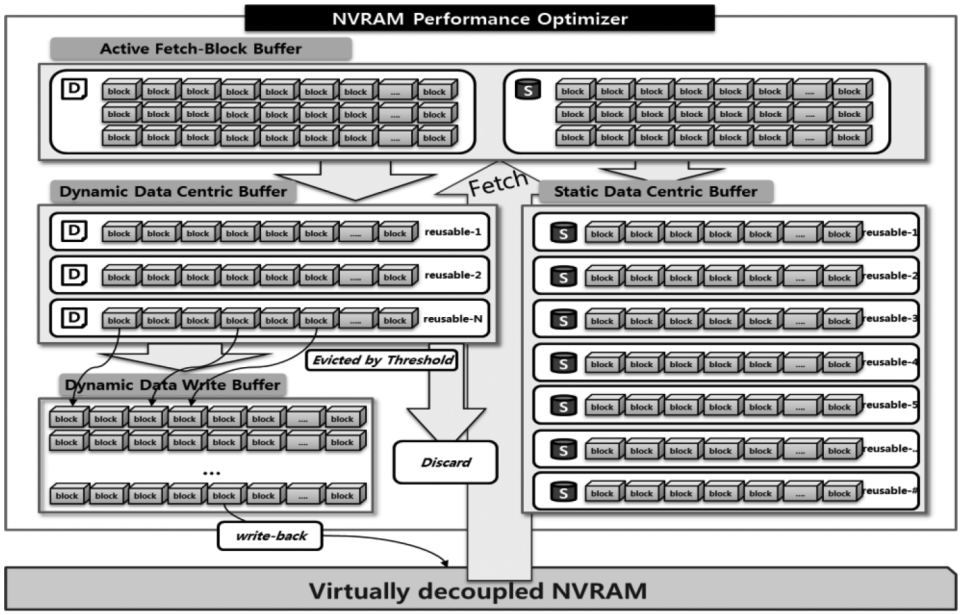

Fig. 4. The NVRAM performance optimizer structure.

Table I. AFBB Management Table

Valid (1bit)	Tag (20/52bit)	Fetch-Block Type (1bit)	Timestamp (8bit)
1	0x1023a	static	36
1	0x2e2a2	dynamic	40
...	...	...	...
0	0x20187	dynamic	120

Figure 5 shows the static and dynamic average access counts of benchmarks for various potential fetch-block sizes ranging from a basic cache block size of 64Bytes to one of 8KB. The x-axis shows the overall percentage of benchmark execution in a 10% unit scale, and the y-axis shows the average access counts of benchmarks. Figure 5(a) shows the number of static accesses by fetch-block sizes, and the 8KB size has the highest reuse rate. For dynamic accessing, as shown in Figure 5(b), an 8KB fetch-block size again shows an outstanding reuse rate compared with the other cases.

Figure 6(a) and 6(b) shows the performance impact on benchmarks of various fetch-block sizes, ranging from 4KB to 64KB. In Figure 6(a), the x-axis shows the fetch-block size at the granularities appropriate for managing the AFBB, and the y-axis shows the hit rate. In Figure 6(b), the x-axis shows the fetch-block sizes, and the y-axis shows the corresponding write-back traffic. Figure 6(a) shows that 16KB gives the best hit rate for the AFBB. However, Figure 6(b) shows that an 8KB fetch-block case is preferable because a bigger fetch-block size induces significantly higher write-back traffic from AFBB to either DDCB or SDCB, causing performance degradation.

3.3.2. Static Data Centric Buffer and Dynamic Data Centric Buffer. The SDCB and DDCB are managed in units of 4KB to exploit both spatial and temporal locality. Basically, the two buffers are designed to have the same structure but different sizes: the SDCB consists of a 512KB space and the DDCB consists of a 256KB space, resulting in a total capacity of 768KB. As shown in Table II, each buffer has an associated management table that stores dirty status and access counts for each reusable-block. If any particular block in

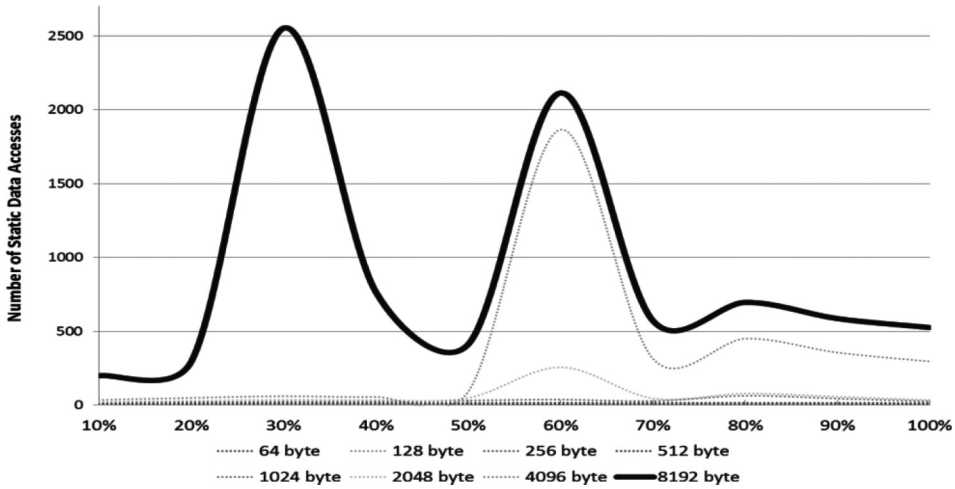

Fig. 5(a). Average access count of static data fetch-block size.

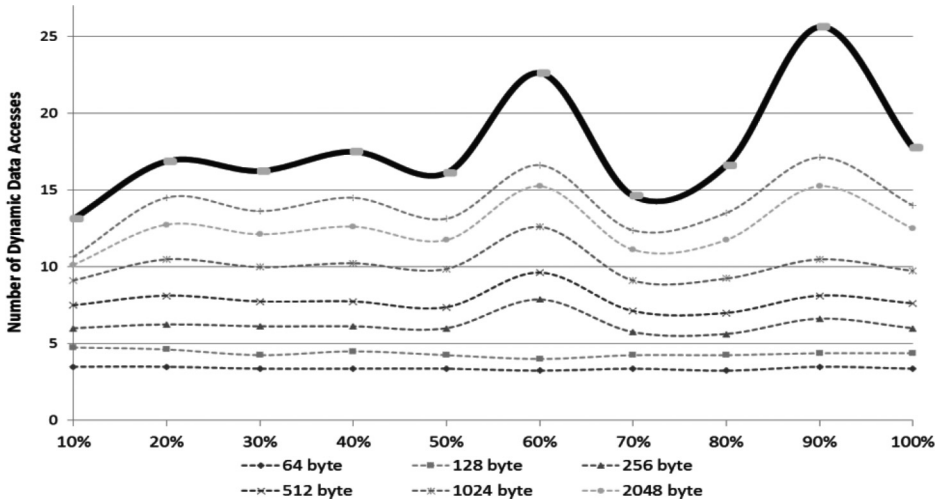

Fig. 5(b). Average access count of dynamic data by fetch-block sizes.

the 4KB reusable-block is accessed by a request, it renews the threshold value of the reusable-block. Victim blocks for each buffer are selected based on a minimum threshold value to maximize temporal locality. The SDCB management table has a 64KB space, and the DDCB management table has a 37KB space for buffer management, thus the total space is less than 128KB.

We simulated only two cases, which are 4KB and 8KB managing granularity, and determined to use 4KB based on write-back traffic and hit rate. We don't consider sizes greater than 8KB because one fetch-block from the AFBB can be divided into a set of sub-blocks to check reusability. Thus, only recently accessed sub-blocks can be collected to move into the next level DDCB buffer. This is because the DDCB buffer can only receive evicted data from the AFBB, and only reusable sub-blocks need to be maintained for optimal reusability.

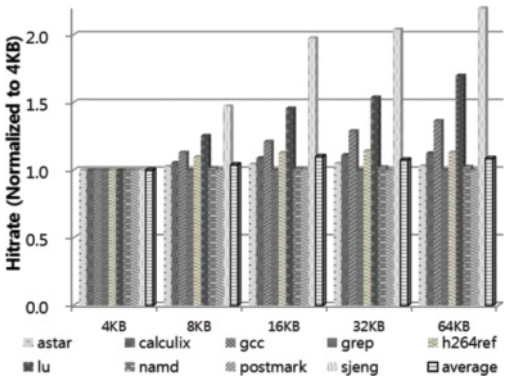

Fig. 6(a). Hit ratio of AFBB by fetch-block size.

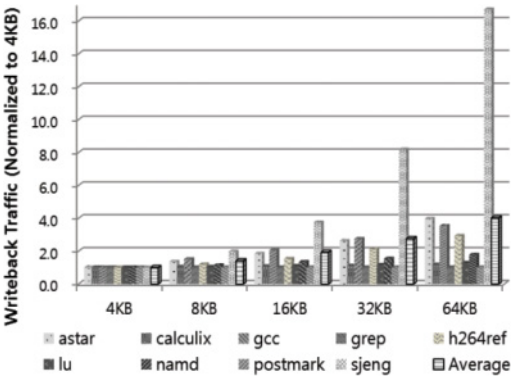

Fig. 6(b). Write-back traffic of AFBB by fetch-block size.

Table II. Static and Dynamic Data Centric Buffer Management Table

		Block 1 (1+8bit)		Block 2 (1+8bit)		Block n (1+8bit)		
Valid (1 bit)	Tag (20/52 bit)	D	H	D	H	D	H	Threshold
1	0x1023a	0	3	1	11	0	2	5
1	0x29129	1	15	1	94	0	53	54
0	0x81f1d	0	68	0	21	0	2	30
...	...	...	...	...	...	...	...	...
1	0x18acc	1	5	0	85	0	1	31

Likewise, we determined that SDCB managing granularity should be 4KB for reasons similar to those for the DDCB. Note that it has only static data types and thus there is no write-back traffic from the SDCB to the MDIS.

3.3.3. Dynamic Data Write Buffer. The DDWB consists of a 256KB space used to hide the write latency to the NVRAM. The DDWB stores only the dirty cache blocks among the reusable-blocks evicted from the DDCB, and so the data can be selectively written in cache block size units (64Bytes). The DDWB is managed according to FIFO protocol. Thus, dirty cache blocks are only written back to the NVRAM when they are evicted from the DDWB. Because of the DDWB, dynamic data can pass through up to three levels of buffer hierarchy, allowing greater delays for write operations. The DDWB includes a write queue and so can provide requested data concurrently.

3.3.4. Operation Flow of Nonvolatile RAM Performance Optimizer. Data requests occurring from the last-level cache are translated into a physical address for the virtually decoupled NVRAM through the NV translation layer, as shown in Figure 7:

—Step 1: Any request from the last-level cache is started in the virtually decoupled NVRAM layer.
—Step 2: The NVPO translates the virtual address into the relevant physical address. If a hit occurs in the AFBB, then the process goes to step 6; otherwise, it goes to the next step.
—Step 3: If the data type is static, the SDCB is accessed. If a hit occurs in the SDCB, the process goes to step 6. Otherwise, it goes to step 7.
—Step 4: If the data type is dynamic, the DDCB is accessed for a match. If a hit occurs in the DDCB, the process goes to step 6. Otherwise, it goes to the next step, when a miss occurs in the DDCB.

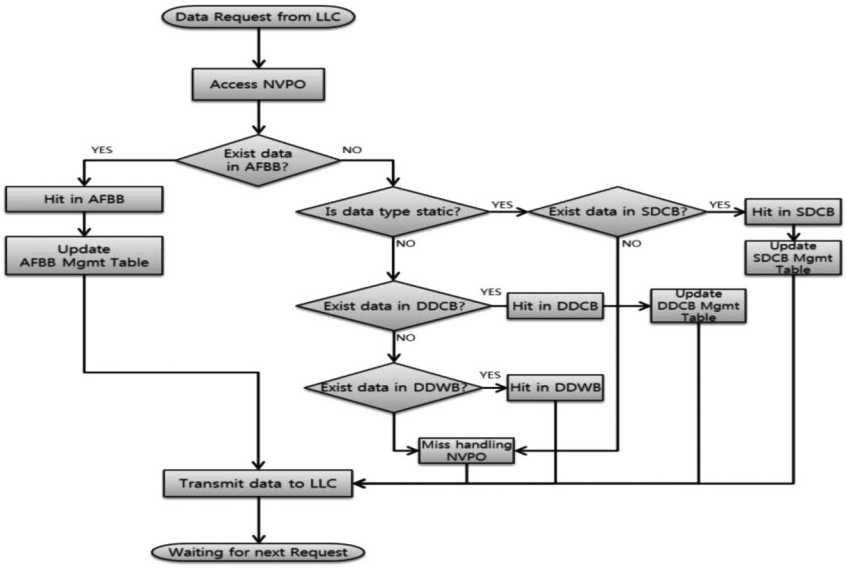

Fig. 7. Operation flow of nonvolatile RAM performance optimizer.

—Step 5: If the data matches in the DDWB, the process goes to step 6; otherwise, it goes to step 7.
—Step 6: The matched data are provided to the last-level cache, and the buffer management table is updated to maintain buffer status. The process then waits for the next request.
—Step 7: Arriving at this step means that the NVPO could not find any matched data; thus, the virtually decoupled NVRAM layer is accessed, the requested fetch-block is transmitted to the NVPO, and its associated cache block is transferred to the last-level cache, simultaneously.

3.3.5. Handling Misses of the NVPO. When a miss occurs in the NVPO, we determine a fetch-block to be evicted from the AFBB and then replace it. The evicted fetch-block from the AFBB is migrated to the DDCB or SDCB, depending on its type. In this process, as shown in Figure 8, threshold-based FIFO algorithms have been designed for the AFBB, SDCB, and DDCB to determine which fetch-block or reusable-block to evict. To enhance temporal locality, evicted fetch-blocks or reusable-blocks are migrated into the lower level buffers, as shown in Figure 9.

Finally, only dirty cache blocks are written to the virtually decoupled NVRAM layer. The details of the miss handling operations are as follows:

—Step 1: The AFBB selects a FIFO fetch-block to evict.
—Step 2: The evicted fetch-block is migrated to the DDCB or SDCB based on its data type. If the data type is static, the process goes to step 3; otherwise, to step 5.
—Step 3: If the SDCB has no space for the evicted fetch-block, the reusable-block with the lowest threshold value is selected by checking the threshold management table.
—Step 4: The evicted reusable-block from the SDCB does not need to be written to the NVRAM, and the NVPO discards the evicted reusable-block. The process then goes to step 9.
—Step 5: If the DDCB has no space for the evicted fetch-block, the two reusable-blocks are selected for replacement based on the lowest threshold values by checking the threshold management table.

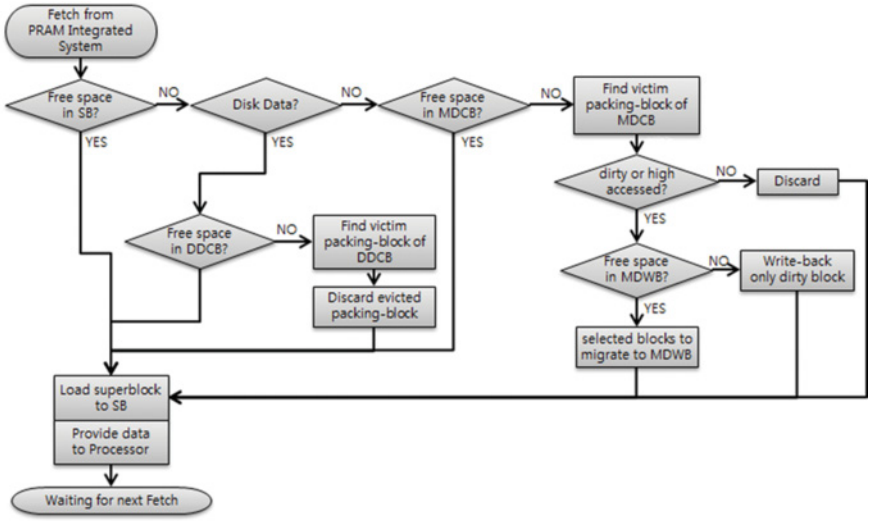

Fig. 8. Operation flow for handling misses of the NVPO.

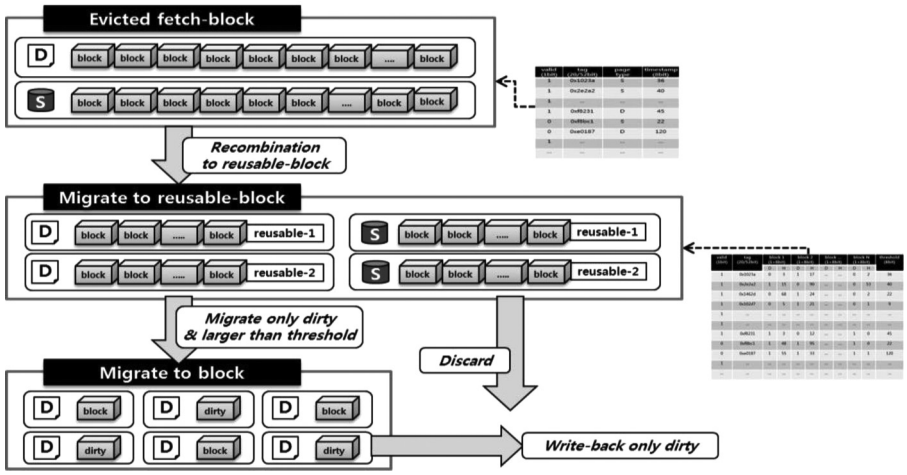

Fig. 9. Handling misses of the NVPO.

—Step 6: When evicted from the DDCB, dirty cache blocks and those clean cache blocks that have higher access counts than the threshold of the evicted reusable-block are migrated to the DDWB.
—Step 7: The process checks for any space available in the DDWB. If there is space available, the evicted cache blocks are migrated to the DDWB, and the process goes to step 9. If there is no space available, it goes to step 8.
—Step 8: The DDWB evicts the oldest cache block. If the oldest cache block in DDWB is dirty, it can be written back to the NVRAM; otherwise, it is discarded.
—Step 9: The evicted fetch-block of the AFBB is replaced with a new one.

4. EXPERIMENTAL RESULTS

In this section, we describe the simulation environment used to evaluate the impact of the proposed MDIS. We use eight benchmarks from the SPEC 2006 suite widely used

Table III. Simulation Configurations

Trace extractor		Pintool
Benchmark suites		SPEC 2006 (astar, bzip2, calculix, gcc, h264ref, namd, sjeng, soplex) SPLASH-2 (lu) grep PostMark
L1 Instruction, Data Cache		32KB, 8-way set associativity, 64Byte block size
L2 Unified Cache		256KB, 8-way set associativity, 64Byte block size
L3 Unified Cache		8MB, 16-way set associativity, 64Byte block size
NVPO	AFBB	1MB, 8KB fetch-block size
	DDCB	256KB, 4KB reusable-block size
	DDWB	256KB, 64Byte cache block size
	SDCB	512KB, 4KB reusable-block size
Dual Buffer	Buffer 1	1MB, 4KB page size
	Buffer 2	1MB, 128Byte block size

in both industry and academia, the lu benchmark belonging to the SPLASH-2 suite, and the grep [Singer et al. 2011] and PostMark [Katcher 1997] benchmarks. Only six out of the eight benchmarks from SPEC 2006, plus lu, grep, and PostMark, are used to evaluate the performance and hit-ratio to cover different computing characteristics in a normal computing environment. The other two SPEC benchmarks, bzip2 and soplex, are used in other experiments such as power consumption. From the SPEC 2006 suite, we use four integer benchmarks (astar, gcc, h264ref, and sjeng) and two floating point (calculix and namd). The SPLASH-2 suite consists of benchmarks for science, engineering, and graphics. We only use lu because it is widely used to represent dense linear algebra computations and reflects the distinct communication patterns among multiprocessors via shared memory. Grep is a read-intensive application over string-based data that has output relatively much smaller than its input data, and it also leads to much higher miss rates compared with other benchmarks. PostMark is for simulating heavy small-file system loads with a minimal amount of software to measure the transaction rates for a workload approximating a large internet electronic mail server.

For experiments related to power consumption, we exploit eight benchmarks from SPEC 2006, which are astar, bzip2, calculix, gcc, h264ref, namd, sjeng, and soplex. These help evaluate the work done by only the CPU and memory subsystem.

As summarized in Table III, we used the pin tool [Intel 2012] as a program instrumentation tool and a subset of the SPEC 2006 (astar, bzip2, calculix, gcc, h264ref, namd, sjeng, and soplex) benchmark suite [Henning 2006; SPEC CPU 2006; Phansalkar et al. 2007], SPLASH-2 (lu) [Woo et al. 1995], grep, and PostMark [Katcher 1997]. Table III also shows that the system configuration used in the our simulator is a 32KB L1 instruction cache and a 32KB L1 data cache, formed as an 8-way set associative with a 64 byte block size. The L2 unified cache is 256KB configured as an 8-way set associative with a 64 byte block size. The L3 unified cache is 8MB in size, a 16-way set associative with a 64 byte block size, and it also operates as a last-level cache. These parameters are based on recent processor configurations. Execution traces including the memory footprints of the benchmarks are obtained from the processor requests, memory management routines (e.g., allocation and deallocation), and read and write requests using the Pin tool while workloads are executed. The trace files are used as the input to our simulator for the proposed architecture. Detailed simulation parameters are listed in Table IV [Shin et al. 2012; Dhiman et al. 2009; Jung et al. 2012; Chung et al. 2011].

Table IV. Simulation Parameters

Parameter	DRAM	SLC PRAM
Read Latency	10ns	20ns
Write Latency	10ns	100ns
Endurance	10^{16}	10^{8}
Row Read Power	210mW	78mW
Row Write Power	195mW	773mW
Active Power	75mW	25mW
Standby Power	90mW	45mW
Refresh Power	4mW	0mW

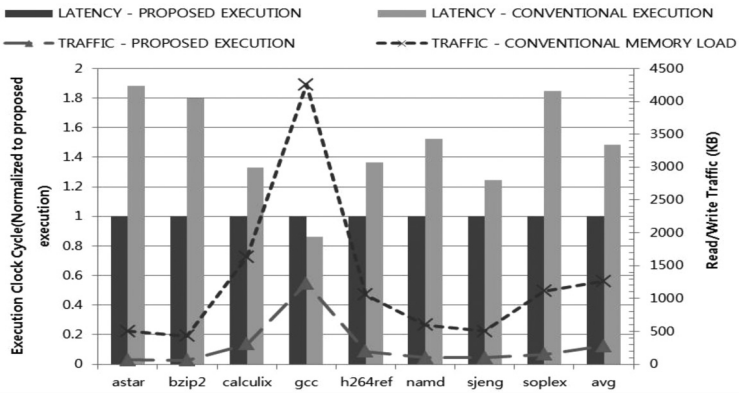

Fig. 10. Load latency and traffic.

Our trace-driven simulator assumes two ports (one read port and one write port) and is composed of processor cache memory, the NVPO buffer modules, and a memory-disk integrated module. For the experiment, the simulator for a conventional memory hierarchy is composed of a processor cache, main memory, disk buffer, and disk module. Extracted trace information for the benchmark suites is based on the processor requests that reach each level of the hierarchy. All benchmark suites are compiled as a single threading static binary for in-order execution. In each model configuration, we show the performance results in terms of hit rate, read/write traffic, access latency cycles, and power consumption. The configuration of our proposed architecture is composed of 1MB for AFBB, 256KB for DDCB, 256KB for DDWB, 512KB for SDCB, and 1GB for the nonvolatile memory device. We also implemented a simple dual-buffer simulator with the same buffer space as the proposed architecture to obtain spatial and temporal locality characteristics as a baseline to compare against the proposed system. The dual-buffer simulator fetches 4KB pages, which are the same size as those in the conventional memory and includes data threshold management [Choi et al. 2013].

4.1. Memory Load Traffic

In this section, we measure total traffic between the main memory and storage. The conventional memory hierarchy system loads data into the main memory by page size. In contrast, our proposed system does not load any static data from the text-segment or read-only data segment of the process. Figure 10 shows the average time to access instructions or data during program execution for the conventional hierarchy and the MDIS, as well as their read/write traffic results. The x-axis indicates the benchmark programs and the average value, and the y-axis on the left side shows the relative access latency normalized to that of direct execution by the proposed MDIS structure.

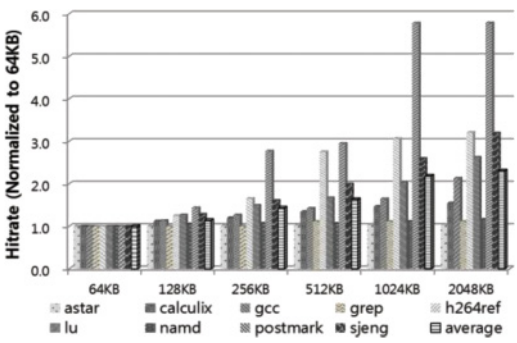

Fig. 11. Hit rate of AFBB by capacity.

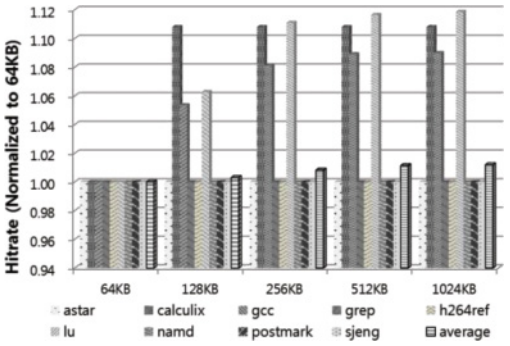

Fig. 12. Hit rate of SDCB by capacity.

Figure 10 also shows the average length of the main memory response, including associated disk access. The right side of the y-axis shows the total traffic between the processor and the main memory, where direct execution by the MDIS requires a total transfer of about 300KB to the processor, and the conventional main memory loads transfer about 1,300KB. According to our results, direct execution via our proposed architecture is about 33% faster on average than the conventional method after the main memory pages are loaded.

4.2. Impact of NVPO Buffer Size

The NVPO buffer is meant to minimize the access time for the NVRAM MDIS while guaranteeing cost-effectiveness and simple management. Thus, our experiment tries to limit the amount of buffer space used in our proposal to a maximum of 2MB, with acceptable latency. Based on this condition, the space range in the experiment varies from 64KB to 2MB. In Figures 11 to 14, the x-axis indicates the size of each buffer, and the y-axis shows the hit rate depending on those sizes.

As shown in Figure 11, for the hit rate by AFBB size, it shows the best result with the 1MB (1,024KB) space. Thus, we determined that a 1MB space for the AFBB is most suitable.

After the 1MB space was fixed for the AFBB, we tried to find best combination of SDCB, DDCB, and DDWB with the remaining 1MB. Both the DDCB and DDWB are for dynamic data, whereas the SDCB is to handle the evicted blocks of static data from the AFBB. Thus, the optimal SDCB space was determined first, to minimize the NVRAM access latency. Figure 12 shows a significant increase in hit rate going from 256KB to 512KB space.

The DDCB provides a second chance for reusing the evicted dynamic blocks from AFBB, so we next experimented with the DDCB space ranging from 64KB to 512KB. Although its best result is at 512KB, we must reserve some space for the DDWB, which serves as a write buffer for dynamic data. Thus, we provide 256KB for the DDCB, and the remaining space is assigned to the DDWB. Even though a miss occurs in the DDCB, the possibility of a hit in the DDWB still exists.

Last, in adjusting the size of the DDWB, we found that 256KB not only provides the highest increase in hit rate, but also keeps the total size within the limitation imposed to maintain cost-effectiveness, as shown in Figure 14.

Based on this experimentation, we conclude that 1MB for AFBB, 512KB for SDCB, 256KB for DDCB, and 256KB for DDWB are the best balance between cost- effectiveness and performance.

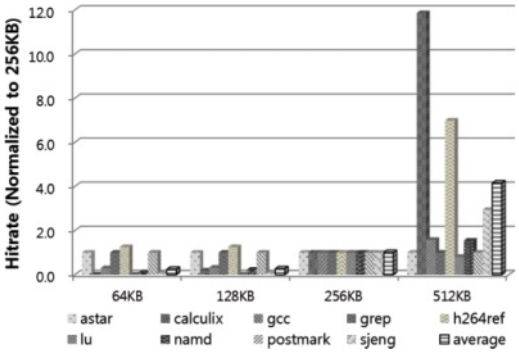

Fig. 13. Hit rate of DDCB by capacity.

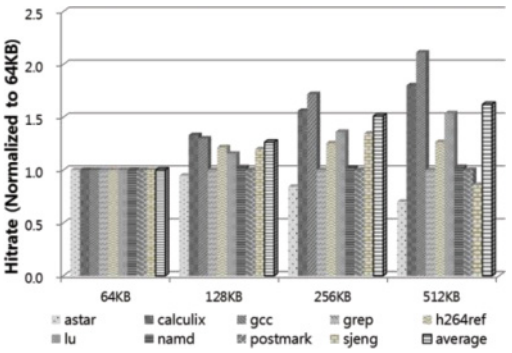

Fig. 14. Hit rate of DDWB by capacity.

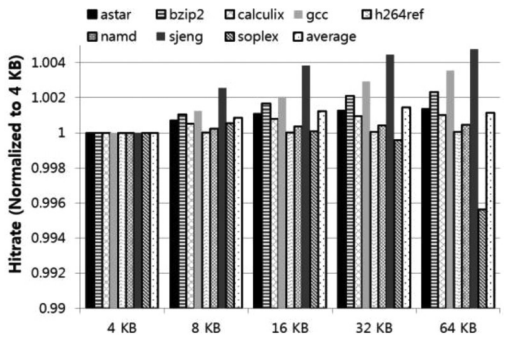

Fig. 15. Hit rate of NVPO.

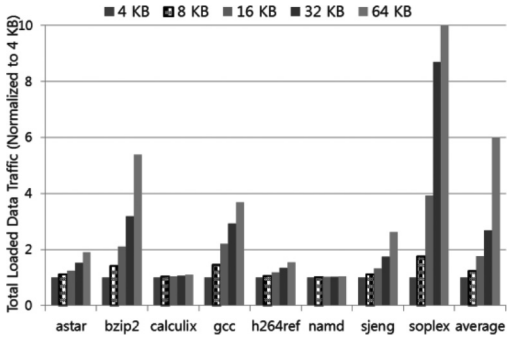

Fig. 16. Total loaded data traffic.

4.3. The Fetch-block Sizes

As explained in Section 3, our proposed system fetches a fetch-block unit from the virtually decoupled NVRAM, but the number of pages to be fetched as a unit cannot be determined by theoretical calculation, and the overall performance of our proposed system is significantly influenced by the fetch-block size of the AFBB. In this section, we measure the impact on overall performance by AFBB fetch-block size, proceeding with experiments step by step. The fetch-block can maximize the spatial locality of the AFBB in the NVPO, so we tried candidate sizes ranging from 4KB to 64KB. As shown in Figure 15, we measured the hit rate of the buffer, normalized to that of a 4KB fetch-block. The x-axis shows the fetch-block sizes from 4KB to 64KB, and the y-axis shows relative normalized hit rates. The results demonstrate that increasing the size of a fetch-block increases the spatial locality of the NVPO, but tradeoffs exist due to the limited NVPO capacity.

The fetch-block results in a long fetch-latency between the NVPO and the NVRAM. Figure 16 shows the total loaded data traffic for the benchmarks, and the y-axis shows data traffic normalized to a 4KB fetch-block size. Here, we see that the amount of fetch traffic tends to increase exponentially with fetch-block size.

The number of write-backs to the virtually decoupled NVRAM, based on evictions from the DDWB in the NVPO, was also measured. We analyzed how many write backs occur to the NVRAM, normalized to a 4KB fetch-block size. Except for the bzip2 benchmark, the write-back traffic makes no difference, but 32KB and 64KB fetch-block sizes tend to increase the write back traffic, as shown in Figure 17. The y-axis indicates

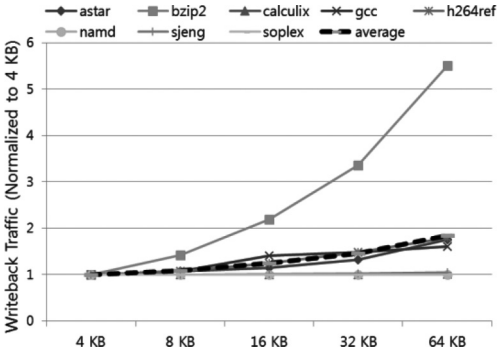

Fig. 17. Total write-back traffic.

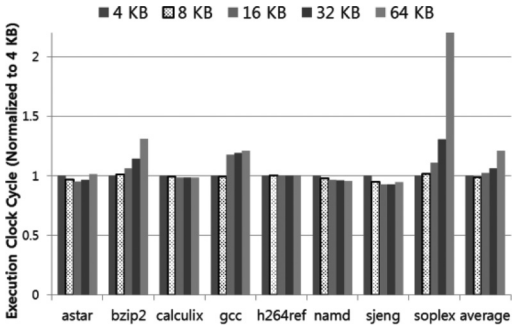

Fig. 18. Execution clock cycles with respect to fetch-block size.

the relative traffic normalized to 4KB, showing that the write traffic is similar across fetch-block sizes.

To identify exactly how effective each fetch-block size is, we experimented to determine the total execution cycles required for write-back operations for the bench-marks. Figure 18 shows the execution clock cycles across the benchmarks. The x-axis indicates the benchmarks, with the size of the fetch-block ranging from 4KB to 64KB, and the y-axis is the relative value normalized to the 4KB fetch-block. The 8KB fetch-block size is the most effective, and the overhead of write-back latency is an influence in the case of the 64KB fetch-block size for the soplex benchmark. We determined that the 8KB fetch-block size maximizes the spatial locality of the NVPO.

4.4. Effect of Memory-Disk Integration

An NVRAM write operation can be initiated by a write operation from the processor, an NVPO eviction operation, and memory allocation. Of course, the conventional hierarchy system has no buffer to reduce write operations and has page swap operations. We measured synthetic write operations to the main memory for the conventional system, and we also checked our proposed system's operations between the NVPO and the NVRAM, as shown in Figure 19. The model is a PRAM main memory and a main memory with a dual buffer that allows page swap operations. We constrained the model to 1GB of main memory, and the y-axis shows the amount of data written for each benchmark, ranging from 0MB to 60MB, where about 40MB of data are written back to storage in the conventional main memory system. The dual-buffer model reduces write operations considerably, to about an eighth that of the conventional system. Our proposed system is able to remove page swap operations and memory allocation for the static data, thereby reducing write traffic to half that of the dual-buffer system.

We analyzed memory writes in detail, finding that, on average, the PRAM main memory writes about 80MB to the memory side, but half of the write operations are memory allocation operations, as shown in Figure 20. The dual-buffer model reduces write operations of dynamic data to the main memory. Our proposed model not only reduced write counts of dynamic data, but also changed the method of execution of static data, thus resulting in decreased write operations to the main memory.

4.5. Lifetime and Performance Evaluation

The lifetime of the NVRAM is compared for three possible cases: the conventional main memory with and without the buffer system and our proposed system. Most NVRAM has a limited lifetime, which is a significant problem. Figure 21 shows the relative lifetime normalized to the conventional main memory system without a buffer, where it

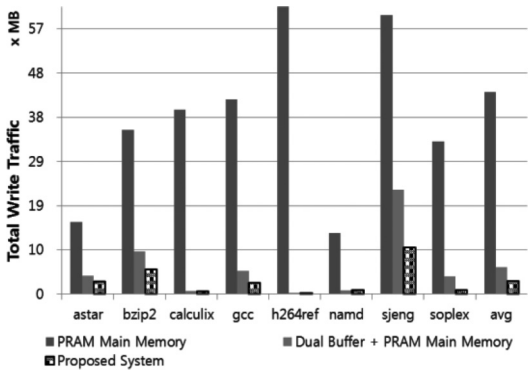

Fig. 19. Total write traffic.

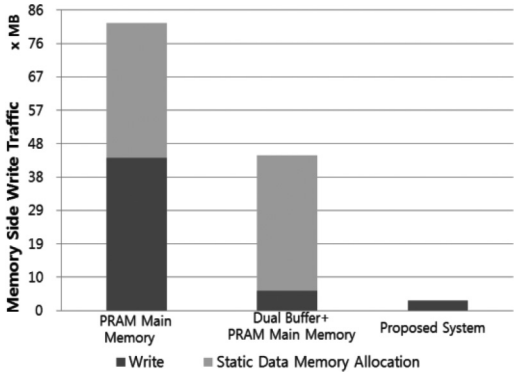

Fig. 20. Memory-side write traffic.

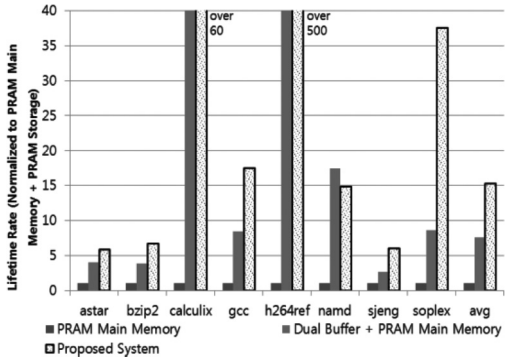

Fig. 21. PRAM lifetime.

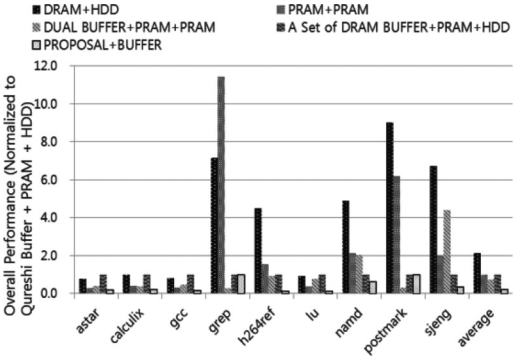

Fig. 22. Overall performance.

is clear that the dual-buffer model and our proposed system are able to eliminate many write operations to the NVRAM storage layer, thus enhancing lifetime. The DDWB in our proposed system is able to impede write operations significantly, and so it hides considerable numbers of write operations. Thus, our architecture is able to increase the lifetime by about 15× compared with the conventional memory hierarchy system without a buffer.

In the experiment related to overall performance, the size of all main memory designs is fixed at 1GB. The work in Qureshi et al. [2009] uses the total size of both buffers as 1GB to secure comparable performance with a conventional system and the size of PRAM main memory as 32GB. To compare fairly, we conducted an experiment with the buffer size reduced to match the work in Qureshi et al. [2009] using a 2MB buffer and 1GB main memory. The overall performance was compared for possible configurations, such as PRAM main memory with a set of DRAM buffers and HDD disk pair by Qureshi et al. [2009], the conventional system with DRAM main memory and HDD storage, the conventional system with PRAM (PRAM secondary storage, and conventional PRAM main memory), and the PRAM storage hierarchy system with buffers. These four systems are compared by counting execution clock cycles occurring between the last-level cache memory and the MDIS via the NVPO (1MB for AFBB, 512KB for SDCB, 256KB for DDCB, and 256KB for DDWB), as shown in Figure 22. Generally, the results show that the execution clock cycles of our proposed structure are much less than those of other systems. The conventional system with DRAM main

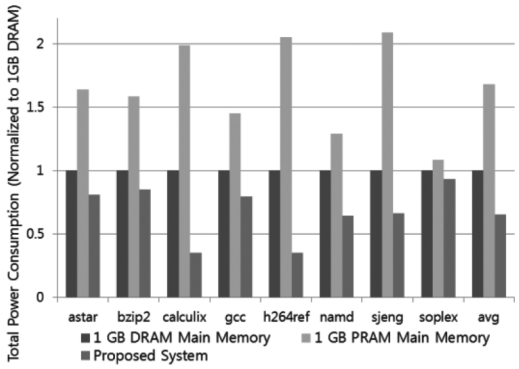

Fig. 23. Total power consumption.

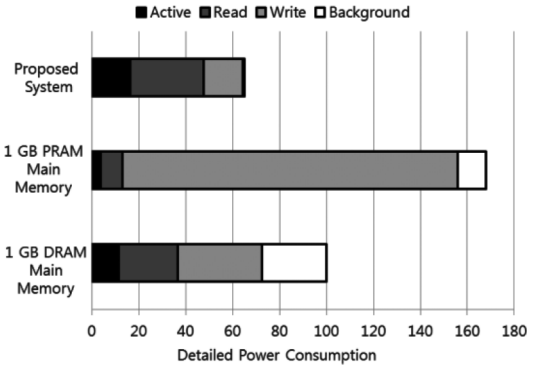

Fig. 24. Detailed power consumption.

memory and a disk pair shows the slowest performance, which is about 9.6× slower than our proposed system. The PRAM main memory with HDD storage by Qureshi et al. [2009] shows comparable performance with the combination model of PRAM main memory and PRAM storage, but our proposed system provides a speedup of 4.5× compared with them.

The PRAM main memory and PRAM secondary storage with a dual-buffer system consisting of a total of 2MB of space improves the performance compared to the conventional system with PRAM main memory and PRAM storage, but our proposed system provides 3.4× faster performance. Moreover, our system is able to increase lifetime and reduce power consumption more than the other cases. For cost effectiveness, an optimal combination of several nonvolatile memory components can be chosen.

4.6. Power Consumption

Figure 23 shows total power consumption of each benchmark workload from SPEC 2006, normalized to a 1GB DRAM main memory. On average, the 1GB PRAM main memory consumes 68% more power than the 1GB DRAM main memory. From Table IV, the read power consumption of PRAM is about 2.7× lower than that of the DRAM; on the other hand the write power consumption is significantly greater. Although PRAM has advantages of low standby/read power consumption and no refresh power compared with DRAM, because PRAM write power consumption is much greater than that of DRAM, PRAM can consume more energy. Specifically, h264ref and sjeng have higher levels of write requests, including write-back counts evicted from last-level cache, than any other benchmark in our sample workload. Soplex has fewer write requests, and thus power consumption for the 1GB PRAM main memory is similar to the 1GB DRAM main memory. Overall, however, compared to the DRAM-based main memory, our proposed system can save 35% of power consumption on average. The reason is that our proposed model reduces standby/refresh of DRAM and a large fraction of the write power consumption of the PRAM by using PRAM with a small amount DRAM.

To clarify the impact on power consumption of our proposed architecture, we show a power consumption break-down for each different model in Figure 24. For this, we assumed 1GB DDR3 SDRAM running at 800MHz and used the Micron system power calculator [Micron 2011]. The background power, including refresh/standby power, consumed about 25% of the total of the 1GB DRAM main memory power. Our proposed architecture reduces both the background power of the DRAM and the write power

consumption of the PRAM by using smaller amount of DRAM as a buffer and avoiding many unnecessary writes.

5. CONCLUSION

In this article, we proposed a memory-disk integrated system consisting of two hardware components: a virtually decoupled NVRAM module and a fetch-block-based nonvolatile performance optimizer. The virtually decoupled NVRAM consists of two virtual spaces: a static space and a dynamic space. The dynamic space can be accessed as a conventional main memory and can expand the capacity dynamically. The virtually decoupled NVRAM module can eliminate memory page swap operations that occur between the conventional main memory and the storage layer, thereby increasing the lifetime of the nonvolatile memory. The fetch-block-based nonvolatile performance optimizer consists of four decoupled buffers and is located between the lowest level of the cache and the virtually decoupled NVRAM module. The dynamic data-write buffer can hide and reduce write operations to the virtually decoupled NVRAM module, increase write endurance, and improve performance. The experimental results show that the proposed MDIS is able to increase the lifetime of the nonvolatile memory to about 15× that of a conventional main memory and storage system. Moreover, the power consumption is reduced by about 35% compared with a DRAM-based main memory system having the same capacity. Furthermore, our proposed architecture can operate 4.5× faster than a PRAM-based main memory system with DRAM buffer proposed by Qureshi et al. [2009] that is comparable to conventional DRAM-based memory architectures. Based on these findings, our proposed memory system offers the possibility of replacing systems using DRAM and HDD disk with a memory-disk integrated NVRAM-based system.

REFERENCES

Ferdinando Bedeschi, Rich Fackenthal, Claudio Resta, Enzo Michele Donze, Meenatchi Jagasivamani, Egidio Buda, et al. 2008. A multi-level-cell bipolar-selected phase-change memory. In *Proceedings of the Solid-State Circuits Conference (ISSCC'08)*. IEEE, 428–625.

Bing-Jing Chang, Yuan-Hao Chang, Hung-Sheng Chang, Tei-Wei Kuo, and Hsiang-Pang Li. 2014. A PCM translation layer for integrated memory and storage management. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis (ESWEEK'14)*. ACM, New York, NY, Article 6.

In-Sung Choi, Sung-In Jang, Chang-Hoon Oh, Charles C. Weems, and Shin-Dug Kim. 2013. A dynamic adaptive converter and management for PRAM-based main memory. *Microprocessors and Microsystems* 37, 6 (Jul. 2013), 554–561.

Hoeju Chung, Byung Hoon Jeong, ByungJun Min, Youngdon Choi, Beak-Hyung Cho, Junho Shin, et al. 2011. A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW. In *Proceedings of Solid-State Circuits Conference Digest of Technical Papers (ISSCC'11)*. IEEE, 500–502.

Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. 2009. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th ACM /IEEE Design Automation Conference (DAC'09)*. IEEE, 664–669.

Diablo Technologies. 2015. Memory channel storage. Retrieved February 23, 2015 from http://www.diablo-technologies.com/memory-channel-storage.

Alexandre Peixoto Ferreira, Rami Melhem, Bruce Childers, Daniel Mossé, and Mazin Yousif. 2010. Using PCM in next-generation embedded space applications. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*. IEEE, 153–162.

GitHub. 2013. Persistent memory file system. Retrieved February 23, 2015 from https://github.com/linux-pmfs/pmfs.

John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (Sept. 2006), 1–17.

Intel. 2012. Pin—a dynamic binary instrumentation tool. Retrieved February 23, 2015 from https://software.intel.com/en-us/articles/pintool#Tutorials/.

Sung-In Jang, Su-Kyung Yoon, Kihyun Park, Gi-Ho Park, and Shin-Dug Kim. 2014. Data classification management with its interfacing structure for hybrid SLC/MLC PRAM main memory. *The Computer Journal* (Nov. 2014). DOI: http://dx.doi.org/10.1093/comjnl/bxu133

Lei Jiang, Bo Zhao, Youtao Zhang, Jun Yang, and Bruce R. Childers. 2012. Improving write operations in MLC phase change memory. In *Proceedings of the 2012 IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1–10.

Ju-Young Jung and Sangyeun Cho. 2013. Memorage: Emerging persistent RAM based malleable main memory and storage architecture. In *Proceedings of the 27th ACM Conference on International Conference on Supercomputing (ICS'13)*. ACM, New York, NY, 115–126.

Jeffrey Katcher. 1997. *Postmark: A New File System Benchmark*. Technical Report TR3022. Network Appliance. 1–8 pages. file:///E:/Desktop/Katcher97-postmark-netapp-tr3022.pdf.

Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 2–13.

Micron. 2011. System power calculator information. Retrieved February 23, 2015 from http://www.micron.com/support/power-calc/.

Shuichi Oikawa. 2013. Integrating memory management with a file system on a non-volatile main memory system. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC'13)*. ACM, New York, NY, 1589–1594.

Shuichi Oikawa. 2014. Adapting byte addressable memory storage to user-level file system services. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems (RACS'14)*. ACM, New York, NY, 338–343.

Jung Sik Park, Ki-Seok Chung, Hi-Seok Kim, and Tae Hee Han. 2010. PRAM and NAND flash hybrid architecture based on hot data detection. In *Proceedings of the 2nd International Conference on Mechanical and Electronics Engineering (ICMEE'10)*. IEEE, 1, 93–97.

Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, 412–423.

Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. 2009. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 24–33.

Andy Rudoff. 2013. Programming models for emerging non-volatile memory technologies. Retrieved February 23, 2015 from https://www.usenix.org/system/files/login/articles/08_rudoff_040-045_final.pdf.

Dong-Jae Shin, Sung Kyu Park, Seong Min Kim, and Kyu Ho Park. 2012. Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory. In *Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS'12)*. ACM, New York, NY, 395–402.

Jeremy Singer, George Kovoor, Gavin Brown, Mikel Luján. 2011. Garbage collection auto-tuning for java mapreduce on multi-cores. In *Proceedings of the International Symposium on Memory Management (ISMM'11)*. ACM, New York, NY, 109–118.

SourceForge. 2013. Protected and persistent RAM filesystem. Retrieved February 23, 2015 from http://pramfs.sourceforge.net.

SPEC CPU. 2006. SPEC CPU2006. Retrieved February 23, 2015 from http://www.spec.org/cpu2006/.

Guangyu Sun, Dimin Niu, Jin Ouyang, and Yuan Xie. 2011. A frequent-value based PRAM memory architecture. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference (ASPDAC'11)*. IEEE, 211–216.

Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA'05)*. ACM, New York, NY, 24–36.

Xiaojian Wu and A. L. Narasimha Reddy. 2011. SCMFS: A file system for storage class memory. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, NY, Article 39.

Chun Jason Xue, Youtao Zhang, Yiran Chen, Guangyu Sun, J. Joshua Yang, and Hai Li. 2011. Emerging non-volatile memories: Opportunities and challenges. In *Proceedings of the 7th IEEE /ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'11)*. ACM, New York, NY, 325–334.

Su-Kyung Yoon, Mei-Ying Bian, and Shin-Dug Kim. 2014. An integrated memory-disk system with buffering adapter and non-volatile memory. *Design Automation for Embedded Systems* (Oct. 2014). DOI: http://dx.doi.org/10.1007/s10617-014-9152-7

Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2004. Design and optimization of large size and low overhead off-chip caches. *IEEE Transaction on Computers* 53, 7 (Jul. 2004), 843–855.

Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. 2009. Durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 14–23.

Received May 2014; revised February 2015; accepted February 2015