Liu HK, Chen D, Jin H *et al.* A survey of non-volatile main memory technologies: State-of-the-arts, practices, and future directions. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(1): 4–32 Jan. 2021. DOI 10.1007/s11390-020-0780-z

A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions

Hai-Kun Liu[1,2,3,4], *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*, Di Chen[1,2,3,4]
Hai Jin[1,2,3,4], *Fellow*, *CCF*, *IEEE*, *Member*, *ACM*, Xiao-Fei Liao[1,2,3,4], *Member*, *CCF*, *ACM*, *IEEE*
Binsheng He[5], *Member*, *ACM*, *IEEE*, Kan Hu[1,2,3,4], and Yu Zhang[1,2,3,4], *Member*, *CCF*, *ACM*, *IEEE*

[1] *National Engineering Research Center for Big Data Technology and System, Huazhong University of Science and Technology, Wuhan 430074, China*

[2] *Services Computing Technology and System Laboratory, Huazhong University of Science and Technology Wuhan 430074, China*

[3] *Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China*

[4] *School of Computing Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

[5] *School of Computing, National University of Singapore, Singapore 117418, Singapore*

E-mail: {hkliu, chendi, hjin, xfliao}@hust.edu.cn; hebs@comp.nus.edu.sg; {hukan, zhyu}@hust.edu.cn

Received July 5, 2020; accepted December 15, 2020.

Abstract Non-Volatile Main Memories (NVMMs) have recently emerged as a promising technology for future memory systems. Generally, NVMMs have many desirable properties such as high density, byte-addressability, non-volatility, low cost, and energy efficiency, at the expense of high write latency, high write power consumption, and limited write endurance. NVMMs have become a competitive alternative of Dynamic Random Access Memory (DRAM), and will fundamentally change the landscape of memory systems. They bring many research opportunities as well as challenges on system architectural designs, memory management in operating systems (OSes), and programming models for hybrid memory systems. In this article, we first revisit the landscape of emerging NVMM technologies, and then survey the state-of-the-art studies of NVMM technologies. We classify those studies with a taxonomy according to different dimensions such as memory architectures, data persistence, performance improvement, energy saving, and wear leveling. Second, to demonstrate the best practices in building NVMM systems, we introduce our recent work of hybrid memory system designs from the dimensions of architectures, systems, and applications. At last, we present our vision of future research directions of NVMMs and shed some light on design challenges and opportunities.

Keywords non-volatile memory, persistent memory, hybrid memory systems, memory hierarchy

1 Introduction

In-memory computing is becoming increasingly popular for data-intensive applications in the big data era. The memory subsystem has an ever-increasing impact on the functionality and performance of modern computing systems. Traditional big memory systems [1, 2] using DRAM (Dynamic Random Access Memory) are facing severe scalability challenges in terms of power and density [3]. Although DRAM scaling is continued from 28 nm in 2013 to 10+ nm in 2016 [4, 5], the scaling has slowed down and become more and more difficult. Moreover, recent studies [6–10] have showed that DRAM-based main memory accounts for about 30%–40% of the total energy consumption of a physical server.

Emerging Non-Volatile Main Memory (NVMM) technologies, such as Phase Change Memory (PCM), Spin-Transfer Torque RAM (STT-RAM), and 3D X-

Survey
Special Section on Memory-Centric System Research for High-Performance Computing
This work is supported jointly by the National Natural Science Foundation of China under Grants Nos. 61672251, 61732010, 61825202, and 61929103.

Point [11], generally offer much higher memory density, and much lower cost-per-bit and standby power consumption than DRAM. The advent of NVMM technologies has potential to bridge the gap between slow persistent storage (i.e., disk and SSD) and DRAM, and will fundamentally change the landscape of memory and storage systems.

Table 1 shows different memory features of Flash SSD, DRAM, PCM, STT-RAM, ReRAM, and Intel Optane DC Persistent Memory Modules (DCPMM)① including read/write latencies, write endurance, and standby power consumption [7, 12, 13]. Despite various advantages in density and energy consumption, NVMM exhibits about 6x–30x higher write latency and about 5x–10x higher write power consumption than DRAM. Moreover, the write endurance of NVMM is very limited (about 10^8 times) while DRAM is able to endure about 10^{16} times of write operations [14, 15]. These disadvantages make it hard to be a direct substitute for DRAM. A more practical way of using NVMM is hybrid memory architectures, composed of both DRAM and NVMM [15, 16].

In order to fully exploit the advantages of both DRAM and NVMMs in hybrid memory systems, there are many open research problems such as performance improvement, energy saving, cost reduction, wear leveling, and data persistence. To address those problems, there have been many studies on the design of memory hierarchy [15–18], memory management [19–21], and memory allocation schemes [22–24]. These research efforts lead to innovations in hybrid memory architecture, operation system (OS), and programming models. Although academic community and industry have proposed a substantial amount of work on integrating the emerging NVMMs in the memory hierarchy, there still remain many challenges to be addressed.

On the other hand, previous studies on NVMM technologies are mostly based on simulated/emulated NVMM devices. The promised performance of NVMM devices may have various deviations compared with real non-volatile (Dual In-line Memory Modules) DIMMs. Recently, the announced Intel Optane DCPMM has finally made NVMM DIMMs commercially available. The real Intel Optane DCPMM behaves significantly differently against the promised features that are expected by previous studies [18, 20, 26, 28]. For example, Intel Optane DCPMMs show 2x–3x higher read latency than DRAM, while its write latency is even lower than that of DRAM [25], as shown in Table 1. The maximal read and write bandwidths for a single Optane DCPMM DIMM are 6.6 GB/s and 2.3 GB/s, respectively, while the gap between read and write bandwidth of DRAM is much smaller (1.3x). Moreover, the read/write performance is non-monotonic with the increasing number of parallel threads in the system [25]. In their experiment, the peak performance is achieved between one and four threads and then tails off. Because of these key features of Optane DCPMM DIMMs, previous studies on persistent memory systems should be revisited and re-optimized to adapt to the real NVMM DIMMs.

Contributions. In this article, we first revisit the state-of-the-art studies on hybrid memory architectures, OS-level hybrid memory management, and hybrid memory programming models. Table 2 shows a classification of state-of-the-art studies about NVMM technologies. We classify these studies in a taxonomy according to different dimensions including memory architectures, persistent memory (PM) management, performance improvement, energy saving, wear leveling, programming models, and applications. We also discuss their similarities and differences to highlight the design challenges and opportunities. Second, to demonstrate the best practices in building NVMM systems, we present our efforts of hybrid memory system designs from the dimensions of architectures, systems, and applications. At last, we present our vision of future research directions of using NVMMs in real application scenarios, and shed some light on design challenges and opportunities in the research field.

Table 1. Different Features of NVMM Technologies

Memory Technology	Read Latency (ns)	Write Latency (ns)	Write Endurance (Times)	Standby Power
Flash SSD	25 000	200 000	10^5	Zero
DRAM	80	80	$>10^{16}$	Fresh power
PCM	50–80	150–1 000	10^8	Zero
STT-RAM	6	13	10^{15}	Zero
ReRAM	10	50	10^{11}	Zero
Intel Optane DCPMM	169 (sequential), 305 (random)	90	10^8	zero

①https://www.tomshardware.com/reviews/intel-cascade-lake-xeon-optane,6061-3.html, Dec. 2020.

Table 2. Classification of State-of-the-Art Studies About NVMM Technologies

Category	Sub-Category	State-of-the-Arts
Memory architectures	Simulators and emulators	PMEP [20], NVMain [26], ZSim [27], HSCC [18], HME [28], Quartz [29], LEEF [30]
	Hybrid memory architectures	Horizontal architectures [14, 15, 17, 31–37], hierarchical architectures [16, 18, 38–42]
OS-level hybrid memory management	Persistent memory management	Working memory: [14–17, 31, 32, 38, 39, 43, 44]; PM file systems: SCMFS [6], BPFS [19], PMFS [20], SIMFS [45], Contour [46], Dapper [47], NOVA [48], Orion [49], ZoFS [50]; persistent objects: Mnemosyne [12], CDDSs [13], NV-heap [51], NV-Duet [52], NVL-C [53], Pangolin [54], TimeStone [55], Pisces [56], Espresso [57]
	Performance improvement and energy saving	Page migration: [14, 15, 33, 34, 44, 58–61]; buffering NVMM writes: [7, 32, 62, 63]; NVMM energy saving: [64–68]; DRAM energy saving: [19, 69–72]
	Write endurance improvement	Write reduction: [8, 14–16, 64, 67, 73–76]; wear-leveling: [15, 77–79]
PM programming models and applications	Programming models and APIs	Mnemosyne [12], CDDSs [13], NV-heap [51], NV-Duet [52], NVL-C [53], Pangolin [54], TimeStone [55], Pisces [56], Espresso [57]
	Applications using NVMMs	Key-value stores: [80–90]; graph computing: [91–93]; machine learning: [94–96]

Although there are other surveys about NVMMs, this survey offers the unique perspective of NVMM and gives more recent review of this field given the rapid development of NVMM. In [97], the authors introduced architectural designs of PCM techniques to address the problems of limited write endurance, potential long latency, high energy writes, power dissipation, and some concerns for memory privacy. In [98], the authors presented a comprehensive survey and review of PCM device related computer architectures and software. Some other interesting surveys focus on architecturally integrating four NVM technologies (PCM, MRAM, FeRAM, and ReRAM) into the existing storage hierarchy [99], or the software optimizations [100] of using NVMMs for storage and main memory systems. Our survey is different from those surveys in three folds. First, previous studies [97, 98] put a focus on the PCM designs from the perspective of computer architecture. In contrast, our paper mainly focuses on system studies of using hybrid memories from the dimensions of memory hierarchy, system software, and applications. Second, our paper contains more reviews of newly-published journal/conference papers. Particularly, we have provided more studies on the new announced Intel Optane DCPMM device. Third, we introduce more about our recent experiences of hybrid memory systems to shed some light on design challenges and opportunities of future hybrid memory systems.

The rest of this paper is organized as follows. Section 2 describes the existing hybrid memory architectures composed of DRAM and NVMMs. Section 3 presents the challenges and current solutions of data persistence guarantees in NVMMs. Section 4 describes state-of-the-art studies on performance optimization and energy saving in hybrid memory systems. Section 5 introduces studies of NVMM write endurance. Section 6 presents our efforts and practices of NVMM techniques. In Section 7, we discuss the future research directions of NVMMs. We conclude the paper in Section 8.

2 Hybrid Memory Architectures

There have been a lot of studies on hybrid memory architectures. Generally, there are mainly two kinds of hybrid memory architectures, i.e., horizontal and hierarchical [18], as shown in Fig.1.

2.1 Horizontal Hybrid Memory Architectures

A number of DRAM/NVMM hybrid memory systems [14, 15, 31] manage DRAM and NVMM in a flat (single) memory address space by OSes [31, 32], and use both of them as main memory. To improve data access performance, these hybrid memory systems need to overcome the drawbacks of NVMM by migrating frequently accessed (hot) NVMM pages to DRAM, as shown in Fig.1(a). Memory access monitoring mechanisms need to be developed to guide the page migration.

Memory Access Monitoring. Zhang and Li [31] used a multi-queue algorithm to classify the hotness of pages, and place hot pages and cold pages in DRAM and NVMM, respectively. Park *et al.* [32] also advocated

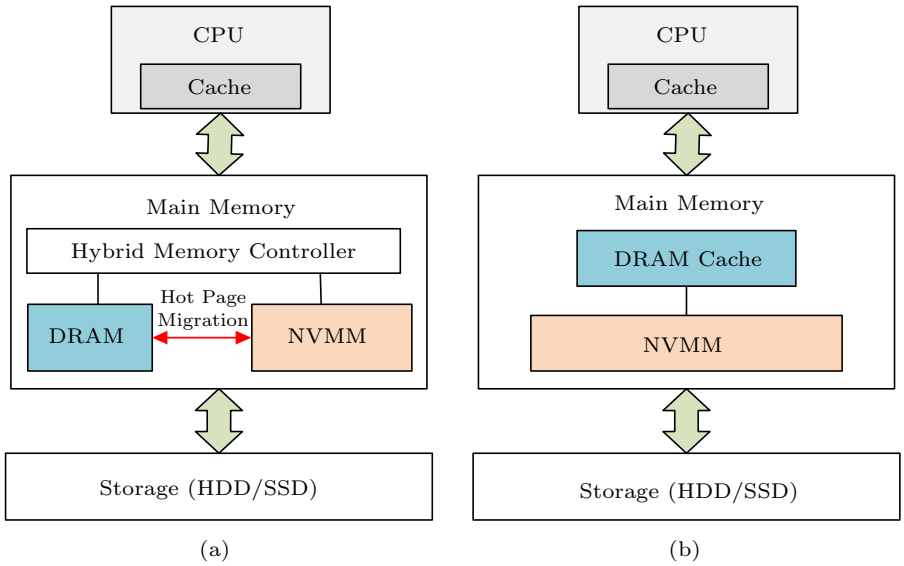


Fig.1. Hybrid memory architectures. (a) Horizontal flat-addressable hybrid memory architecture. (b) Hierarchical hybrid memory architecture.

a horizontal hybrid memory architecture to manage DRAM and NVMM. Moreover, they proposed three optimization strategies to reduce the energy consumption of hybrid memory systems. They monitored memory data in a very fine granularity of a DRAM row, and periodically checked the access counter of each DRAM row. According to counters, the data is written back to NVMM in order to reduce the energy consumption of DRAM refreshing. The data is not cached to DRAM from NVMM until it is accessed again. The dirty data is kept in DRAM as long as possible to reduce the overhead of the data swapping between DRAM and NVMM as well as the costly writes to NVMM.

Page Migration. There have been a number of page migration algorithms proposed for different optimization goals. Soyoon *et al.* [33] deemed that the frequency of NVMM writes is more important than the data access recency in identifying hot pages, and proposed a page replacement algorithm called CLOCK with Dirty bits and Write Frequency (CLOCK-DWF). For each NVMM write operation, CLOCK-DWF needs to first fetch the corresponding page to DRAM and then performs the write in DRAM. This approach may cause many unnecessary page migrations, and thus introduce more energy consumption and write-back operations to NVMM. Salkhordeh and Asadi [34] took both memory writes and reads into account to migrate the hot pages that are beneficial for performance and energy saving, and used two least recently used (LRU) queues to choose victim pages in DRAM and NVM individually. Yoon *et al.* [17] conducted page migrations based on row buffer locality, where pages with low row buffer hit rates are migrated to DRAM while pages with high row buffer hit rates are still kept in NVMM. Li *et al.* [101] proposed a utility model to guide page migrations based on an utility definition on many factors such as page hotness, memory-level parallelism, and row buffer locality. Khouzani *et al.* [35] considered memory layout of programs and memory-level parallelism to migrate pages in a hybrid memory system.

Architectural Limitations. There are several challenges to manage NVMM and DRAM in a horizontal hybrid memory architecture.

First, page-level memory monitoring is costly. On the one hand, as today's commodity x86 systems do not support memory access monitoring at the granularity of pages, hardware-supported page migration schemes require significant hardware modification to monitor memory access statistics [14, 15, 33]. On the other hand, memory access monitoring at the OS layer usually causes significant performance overhead. Many OSes maintain an "accessed" bit in the Page Table Entry (PTE) for each page to identify whether this page is accessed. However, this bit cannot truly reflect the recency and frequency of page accesses. Thus, some software-based approaches would disable Translation Lookaside Buffer (TLB) [102] to track each memory reference. Such page access monitoring mechanisms usually cause significant performance overhead and even offset the benefit of page migration in hybrid memory systems.

Second, page migration is also costly. One time

of page migration may induce many times of page read/write operations (costly). As a page may only contain a small fraction of hot data, the migration at the page granularity is relatively costly due to a waste of memory bandwidth and DRAM capacity.

Third, the hot page detection mechanism may take a long period of time to warm pages up, and thus degrades the gain of page migrations. Moreover, the hot page prediction may not be accurate for some irregular memory access patterns, causing unnecessary page migrations.

2.2 Hierarchical Hybrid Memory Architectures

A number of studies propose to organize DRAM and NVMM through a hierarchical cache/memory architecture [16, 38, 39]. They use DRAM as a cache of NVMM, as shown in Fig.1(b). The DRAM cache is invisible to operating systems and applications, and are managed completely by hardware.

Qureshi *et al.* [16] proposed a hierarchical hybrid memory system composed of a large size of PCM and a small size of DRAM. The DRAM cache contains most recently accessed data to reduce the most expensive NVMM accesses, while the large-capacity NVMM holds most of the required data to avoid costly I/O operations during the execution of applications. Similarly, Mladenov [38] designed a hybrid memory system with a small-capacity DRAM cache and a large-capacity NVMM, and managed them based on the spatial locality of application data. DRAM is managed as an on-demand cache and replaced through an LRU algorithm. Loh and Hill [39] managed DRAM in a granularity of cache lines to improve the efficiency of DRAM cache, and used a group-connected manner to map the NVMM data to the DRAM cache. They put the metadata (tag) and data in the same bank row so that the data can be quickly accessed for cache hits, and reduces the performance overhead of tag querying.

In this memory architecture, as DRAM is organized as N-way set-associative cache, additional hardware is required to manage the DRAM cache. For example, an SRAM storage is needed to store the metadata (i.e., tag) of data blocks in the DRAM cache, and hardware looking-up circuit is required to find the requested data in the DRAM cache. Thus, to access the data in the DRAM cache, two memory references are required, one for accessing the metadata and the other for the actual data. To accelerate metadata accesses, Qureshi *et al.* [16] used a high-speed SRAM to store the metadata. Meza *et al.* [40] reduced hardware cost for tag store by placing metadata alongside data blocks in the same DRAM row. They also proposed to use an on-chip metadata buffer to cache frequently accessed metadata in a small-size SRAM.

Architectural Limitations. Although hierarchical hybrid memory architectures usually deliver much better performance compared with the scenario of accessing the data in NVMM solely, it may cause significant performance degradation when running workloads with poor locality [103]. The reason is that most hardware-managed hierarchical DRAM/NVMM systems leverage an on-demand based data fetching policy for simplicity, and thus the DRAM cache is in the critical data path of memory hierarchy. If a data block does not hit in the DRAM cache, it has to be fetched from NVMM to DRAM regardless of the page hotness. This cache filling strategy may cause frequent data swapping between DRAM and NVMM (similar to the cache thrashing problem). On the other hand, hardware-managed cache architecture cannot fully utilize the DRAM capacity. Since the DRAM cache is designed to be set-associative, each NVMM data block is mapped to a fixed set. When a set is full, it must evict a data block before fetching a new NVMM data block into the DRAM, even though other cache sets are empty.

2.3 Architectures of Intel Optane DCPMM

The recently announced Intel Optane DCPMM supports both horizontal and hierarchical hybrid memory architectures when it is used combining with DRAM. There are currently two operating modes for Optane DCPMM DIMMs: memory mode and application direct mode [25]. Each of these modes has its advantages for specific use cases.

Memory Mode. In this mode, DCPMM acts as a large capacity of main memory. The operating system (OS) recognizes DCPMM as traditional DRAM and the persistence feature of DCPMM is disabled. If traditional DRAM is used combining with DCPMM, it is hidden from the OS and acts as a caching layer for DCPMM. Thus, DCPMM and DRAM are actually organized in a hierarchical hybrid memory architecture. The primary benefit of the memory mode is to provide superior memory capacity to be used on memory bus lanes. This mode strongly emphasizes building large storage capacity environments around the memory space without modifying the upper-level systems

and applications. Recommended use cases would be to expand the main memory capacity for better infrastructure scaling, such as parallel computing platforms for big data applications (MapReduce, graph computing).

Application Direct Mode. In this mode, DCPMM offers all persistence features to OS and applications. OS exposes both DRAM and DCPMM to the applications as main memory and persistent storage, respectively. The traditional DRAM mixed with DCPMM still acts as the standard DRAM for applications, while DCPMM is also assigned to the memory bus for faster memory access. DCPMM is used as one of two types of namespaces: direct access (DAX) and block storage. The former namespace offers byte-addressable persistent storage directly accessed by applications via special APIs. Thus, DCPMM and DRAM are logically organized in a horizontal hybrid memory architecture in this mode. The latter namespace presents DCPMM to applications as a block storage device, similar to an SSD, but can be accessed via a faster memory bus. The Application Direct Mode strongly emphasizes the advantage of latency reduction and bandwidth improvement up to 2.7x faster than NVMe. Recommended use cases would be for large in-memory databases which are subjected to the demand of data persistence.

There is also a mixed memory mode combining the Memory Mode and the Application Direct Mode. A portion of the capacity of DCPMM is used for the Memory Mode operations, and the remaining capacity of the DCPMM is used for the Application Direct Mode operations. This mixed memory mode provides a more flexible approach to manage the hybrid memory system for different application scenarios.

2.4 Summary

The above two kinds of hybrid memory architectures have their own pros and cons for different scenarios. Generally, the hierarchical architecture is more suitable for applications with good data locality, while the flat-addressable architecture is more applicable for latency-insensitive or large-footprint applications. There is not a conclusion on which architecture is better than another one. Actually, both hierarchical and flat-addressable hybrid memory architectures are supported by Intel Optane DCPMMs. One limitation of current DCPMM is that the system needs to restart after reconfiguring the modes of DCPMM. It could be beneficial and flexible for applications if a re-configurable hybrid memory system can dynamically fit different scenarios in a timely and efficient manner. This may be an interesting research direction of NVMM devices.

3 Persistent Memory Management

Data persistence is an important design consideration for NVMMs. In the following, we first present the technical challenges of persistent memory (PM) management, and then introduce the state-of-the-art studies on PM management, including the usage of PM, PM access modes, fault tolerance mechanisms, and persistent objects.

3.1 Technical Challenges

In hybrid memory systems, NVMM can act as main memory when running applications and serve as persistent storage when applications are completed. The byte-addressability and the non-volatility features of NVMM eliminate the distinction of memory and external storage. However, the data in NVMM should be reorganized and relocated when the data needs to be persisted in NVMM.

Fig.2 shows management operations of Persistent Memory (PM). The NVMM region is a physical PM device. The NVMM region can be used as working memory like DRAM, and can also work as persistent storage like disk. When the program is completed, the data in the working memory should be flushed into the persistent storage. Besides, to guarantee high reliability, a checkpointing mechanism is widely exploited to recover systems from a power failure or a system crash. Gao *et al.*[104] developed a novel method to leverage NVMM for real-time checkpointing in hybrid memory systems.

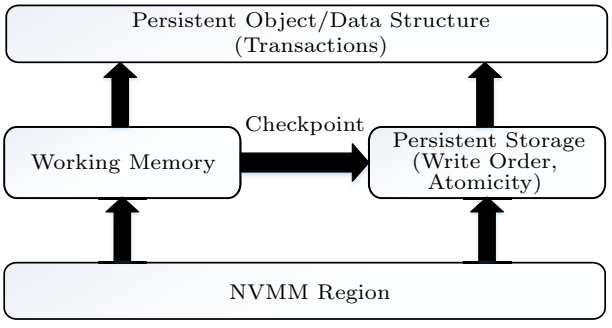


Fig.2. Data persistence in NVMM.

There are several challenges to manage PM efficiently. First, persistent storage is widely managed in the form of file systems. As the byte-addressable

NVMMs offer much better random access performance than traditional block devices, the performance bottleneck of PM-based file systems has shifted from the hardware to the system software stack. It is essential to shorten the data path in the software stack. Second, since many CPUs use write-back cache to achieve high performance for write operations, the last-level cache (LLC) may change the order of data written back to PM. In case of a power failure or system crash, it may cause a data inconsistency problem. Thus, to guarantee data consistency in PM, the order of write operations and a write atomicity model are required to guarantee the data consistency in PM. Third, persistent objects and data structures are more promising for PM programming compared with PM-based file systems, because they eliminate the complex data structures in file systems, including *i*-nodes, metadata, and data. However, these persistent objects and data structures still face the challenges of guaranteeing data consistency. In the following, we will review studies that have attempted to address those challenges.

3.2 Working Memory

A number of studies [14, 105, 106] use NVMM just as a replacement of DRAM, without concerning about the non-volatility property of NVMM. In this use case, both DRAM and NVM are allocated and reclaimed in pages. Application data is written back to external storage when programs complete.

Due to the performance gap between NVMM and DRAM, memory allocation should take the different features of DRAM/NVMM into account. Park *et al.* proposed to place different types of data in hybrid memory system according to application virtual memory layout [43]. Both the DRAM region and the NVM region are managed by the buddy system separately. Upon a page fault, the page allocator selects a type of pages for allocation based on the segment in which they are placed. Pages in heap and stack segments with intensive write operations are allocated in DRAM. Pages in other segments are allocated in NVMM, including read-only text segment and initialized data segment. Similarly, Wei *et al.* [44] also exploited application semantics to direct data placement in hybrid memory systems. However, they determined the placement of heap objects based on object read/write ratios. The above memory allocation policies are implemented in OSes and transparent to programmers. The page placement is also too coarse-grained to some extent since programmers often allocate small-size objects rather than pages.

3.3 Persistent Memory File System

Some work manages NVMM with traditional file systems to transparently support legacy applications. The file system managed NVMM region is called Persistent Memory File System (PMFS) [20]. In PMFS, applications can access the data in PM via read/write interfaces as traditional disk-based file systems. CPU can also directly access PM via load/store instructions based on Direct Access (DAX), which is implemented by the mmap interface, as shown in Fig.3.

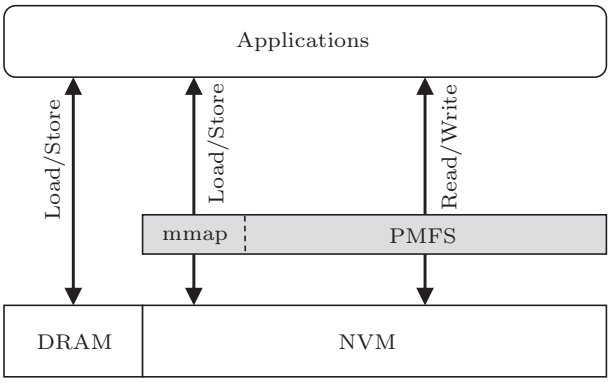


Fig.3. PM access modes in hybrid memory systems.

Although PM is able to significantly improve application performance compared with persistent storage, the direct access to byte-addressable PM still faces challenges of data consistency. As an update to a complex data structure usually contains multiple write operations on NVMM, a power failure or a system crash may incur data inconsistency problems if only a portion of critical data is being written. For example, there are two write operations to insert an item to a hash table in PM: one to write the data and the other to write the metadata. If the metadata is persisted before the data itself and a power failure occurs, the data and its metadata become inconsistent.

Current file systems or databases use atomic updates to tackle this problem, where the correlated write operations are grouped and are performed in a transaction manner, namely transaction updating. Also, in each transaction, multiple writes usually should be constrained in order.

3.3.1 *Write Order Guarantee*

For block-based file systems, the order of writes to persistent storage is usually guaranteed by software due to the huge performance gap between main memory and disk. The I/O operations are buffered sequentially in DRAM and flushed to persistent storage synchronously.

However, in hybrid memory systems, cache lines may be written back to NVMM in an order different from the order issued by CPUs. To guarantee the order of data writes to PM, there are generally three kinds of approaches in the following.

Hardware Primitives. PM file systems and programming frameworks can ensure the write order by explicitly evicting cache lines to NVMM. Modern CPUs provide *clflush* and *mfence* instructions to achieve this goal. The *clflush* instruction is used to evict a cache line to main memory explicitly. The *mfence* instruction is used to guarantee the order of all *load* and *store* operations before it. There have been various index access methods that optimize the usage of those instructions such as NV-tree [107] and Bztree [108].

Nevertheless, *clflush* invalidates a cache line in all cache levels and leads to performance degradation. Moreover, *clflush* only flushes a cache line to memory controller, and does not guarantee that the data is actually written in NVMM. To tackle these problems, Intel has developed two new instructions, i.e., *clwb* and *PCOMMIT*②. *clwb* writes back a cache line to memory controllers without invalidating it in the cache, and PCOMMIT ensures the data is finally written to the NVMM chips.

Write-Through Cache. Some previous studies adopt the write-through cache to guarantee data persistence in PM [12, 109]. Write operations can bypass CPU caches via instructions like *movntq*. It writes dirty data directly to memory rather than cache, offering a simple way to guarantee the write order to NVMM, without using the complicated barrier and costly flush operations. However, this strategy leads to significant performance degradation because write operations are manipulated in a stream manner. Mnemosyne [12] provides both the hardware primitives and write through policies to guarantee the order of writes.

Persistent Cache. Kiln [74] utilizes a non-volatile cache as the last-level cache (LLC) to guarantee data persistence at the cache level. In the non-volatile LLC, updates can be completed in-place. Kiln tracks the dirty lines that need to be updated but still retain them in the non-volatile cache. As most of the updated writes have existed in the non-volatile LLC, Kiln improves the system performance by reducing most writes to NVMM. Besides, the updated data is kept even when a system failure occurs.

3.3.2 Atomic Updating

Atomicity implies each update should be done in a "all-or-nothing" manner. It can avoid a data structure being partially updated upon power failures or system crashes. It is always implemented by a transaction operation. Modern processors can provide 8-byte atomic updates to DRAM or NVMM. An update to a simple variable up to 8 bytes can be done in-place. For more complex data structures, atomic updating operations become more complicated. There are three technologies to guarantee complex atomic operations, such as journaling, shadow updating, and logging structure.

Journaling. Journaling is commonly used in databases and file systems to guarantee atomic updating. All updates in a transaction are recorded to a journal file before the real object is updated. Thus, journaling always writes the same data twice, one to the journal file and the other for the actual data. To diminish the performance overhead due to duplicate writes, most systems only record metadata in journal files. For example, Ext4-DAX③ supports direct access to NVMM and uses the journaling mechanism to achieve metadata atomicity.

Shadow Paging. Shadow paging is a copy-on-write (COW) mechanism for tree-based file systems and databases. Each write operation triggers a memory copy. In the context of file systems, the COW operation needs to transfer from the root to the leaf in a cascade way. This cascade updating is costly. In BPFS [19], Condit *et al.* proposed a short-circuit shadow paging mechanism for atomic updating. Data updates are token in-place, including in-place updating and in-place appending. Fig.4 shows an example of in-place appending. The appended data is written to the end of the file in-place, and then the file size is updated in-place. In case of a system crash before the updating of file size, the appended data is invalid. In PMFS [20], DRAM pages are allocated and reclaimed by virtual memory manager and NVMM pages are managed by PMFS. Atomic updating is achieved through three approaches: in-place, logging and COW. In-place updates are used for 8-byte metadata atomic writes and metadata updates at 64-byte cache line size. Logging updates are used for more complicated metadata updates. The COW mechanism is used to update file data.

②https://software.intel.com/sites/default/files/managed/0d/53/319433-022.pdf, Dec. 2020.

③https://lwn.net/Articles/613384/, Dec. 2020.

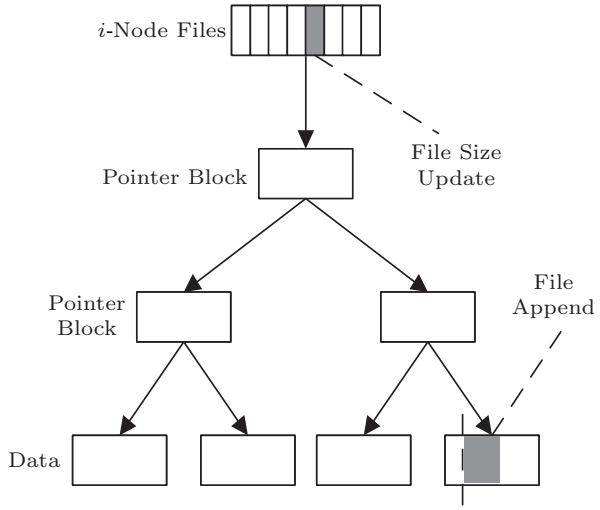


Fig.4. Shadow paging.

Logging Structure. Log-structured file systems are designed to exploit the relatively high performance of sequential writes to disk. Random writes are converted into sequential writes in a DRAM buffer, and then are synchronized to disk. However, for byte-addressable NVMM, the contiguous free memory region required by logging usually leads to difficulties in memory allocation and garbage collection [48, 110]. NOVA [48] redesigns a traditional logging-structure file system to improve the parallelism of data I/O and relax the constraints of contiguous memory allocation. NOVA maintains a log for each updated *i*-node rather than a uniform contiguous log file. Thus, multiple file updates and recovery can be token in parallel. To mitigate the pressure of contiguous memory allocation and garbage collection operations in log-structured file systems, NOVA adopts a linked list to store log pages. Besides, to accelerate the access to persistent files, NOVA maintains a directory tree in DRAM. Doshi *et al.* [111] exploited a backend cache controller to write the data to PM in an asynchronous way. A victim DRAM cache is used to store cache lines evicted from LLC, and then the persistent updates to NVMM are combined and written to NVMM in a streaming way. CCDS [13] guarantees data consistency by atomic updating and maintaining multiple versions of data. After an atomic update to a critical data structure, a new version of the data is created. To guarantee data consistency during updating, the most recent version is recorded and can be accessed by all threads. Upon a power failure, the most recent version is used for recovery while all in-progress updates are removed.

3.4 Persistent Objects

As PM-based file system interfaces still rely on complex software I/O stack and can introduce multiple times of data copying, a more attractive way is to store and access application data structures directly in PM, namely persistent data structures.

Persistent data structures have been widely explored in object-oriented databases. Berkeley DB [112] and Stasis [113] can define persistent data structures explicitly by application programming interfaces (APIs). However, all these systems store the persistent data structures in block-based disks. Recently, there have been a number of persistent object programming frameworks proposed for NVMM, such as NV-heaps [51], NV-Duet [52] and NVL-C [53]. Therefore, programmers can definitely allocate NVMM via *pmalloc* and DRAM via *malloc*, and further optimize data placement in hybrid memory systems according to application semantics.

A major challenge of using persistent objects is to guarantee referential integrity of objects, i.e., all references must point to a valid data. Otherwise, a memory leak or a memory error occurs because of dangling pointers or wild pointers. For example, if a pointer in PM refers to an object in DRAM, upon a power failure the pointer in PM becomes a dangling pointer. In a PM system, memory leaks and memory errors are usually more destructive since these exceptions may be permanent. The referential integrity may occur in three scenarios: memory allocation, pointer assignment operations and deallocation. In the following, we use NV-heaps as an example to illustrate these scenarios.

NV-heaps [51] is a lightweight and high-performance persistent object system using the emerging persistent memory. To guarantee data consistency and durability, NV-heaps provide a set of easy-to-use programming primitives, including persistent objects, specialized pointers, a memory allocator and atomic sections.

For memory allocation, NV-heaps are also subject to wild pointers as conventional programming models. Once an NV-heap is not pointed by a valid pointer, its memory space may be permanent unavailable until the NVMM device is reset. To prevent memory leak due to wild pointers, NV-heaps explore reference counters for garbage collection. A heap is reclaimed immediately once no other objects point to it.

For pointer assignment, four new pointer types may be generated in hybrid memory systems: pointers within an NV-heap (intra-heap NV-to-NV pointers), pointers from an NV-head to another NV-heap

(inter-heap NV-to-NV pointers), pointers from volatile memory to an NV-heap in NVMM (V-to-NV pointers), and pointers from an NV-heap in NVMM to volatile memory (NV-to-V pointers). To guarantee referential integrity, NV-to-V pointers and inter-heap NV-to-NV should not be assigned. The NV-to-V pointers are unsafe when a program ends. For instance, a pointer A in NVMM points to a data structure B in DRAM. When the program ends, the memory assigned to B in the DRAM is reclaimed while the persistent pointer A remains. An error may occur if the pointer A is accessed. The inter-heap NV-to-NV pointers are also dangerous. For example, a pointer P from NV-heap M points to another NV-heap N. If N becomes unavailable, the pointer P will point to invalid data. However, the inter-heap NV-to-NV pointers are still needed in some cases such as doubly-linked lists. Thus, weak pointers are proposed to implement inter-heap NV-to-NV pointers. Weak NV-to-NV pointers act like normal pointers but they do not affect the reference counts. When the reference count becomes zero, all weak pointers should be atomically released. When an NV-heap is closed, V-to-NV pointers may lead to a memory leak. To prevent this unsafe closing, NV-heaps are unmapped only when the program ends.

3.5 Studies on Intel Optane DCPMMs

The emergence of Intel Optane DCPMM arouses increasing interests in disclosing its performance features and the potential impact on data center applications [25, 114–117]. These experimental studies are essential to guide the design of hybrid memory systems and the application programming of DCPMM. Izraelevitz *et al.* offered an earliest, scholarly, and comprehensive performance measurement of DCPMM [114]. They explored its capabilities as a main memory device, as well as byte-addressable persistent memory exposed to user-space applications. This report enlightens the research community to understand the non-volatile memory devices, and to guide the future work on hybrid memory systems. They also explored the performance characteristics of Intel Optane DIMMs through both microbenchmarks and macrobenchmarks, and recommended a set of best practices to maximize the performance of this device [25]. Weiland *et al.* explored the performance features of Intel Optane DCPMM and the impact on high-performance scientific applications in the context of performance, efficiency and usability in both Memory and App Direct modes [116]. A similar work was presented by Patil *et al.* [115] They evaluated the performance characterization of DRAM-NVM hybrid memory systems for HPC applications using real DCPMM. They found that the NVMM-only executions are slower than DRAM-only and Memory-mode executions by a minimum of 2%, and a maximum of 6 times. Peng *et al.* [117] evaluated the impact of using DCPMM on in-memory graph processing workloads, and the experimental results suggest that the performance and power efficiency of applications can be optimized by properly distributing the data between NVMM and DRAM.

There have been a few hybrid memory systems designed from scratch, and also performance optimizations of existing systems using the new Intel Optane DCPMM device. Lersch *et al.* [86] conducted an extensive study of range indexes on DCPMM. They used a unified programming model for all trees to guarantee fair comparison and developed a benchmarking framework called PiBench. The empirical evaluation has recognized effective techniques, insights, and caveats to direct the design of the future PM-based index structures. Dash [87] is a holistic approach to building dynamic and scalable hash tables on DCPMM. The design takes scalability, load factor and recovery into consideration. The authors develop two popular dynamic hashing schemes, i.e., extensible hashing and linear hashing to demonstrate the efficiency of Dash. Gill *et al.* [93] presented the runtime and algorithmic principles of performing large-scale graph analytics on DCPMM and highlighted the principles of graph analytics on all large-memory platforms. Mahapatra *et al.* [88] argued that it is inefficient to persist all data structures such as Doubly Linked List, B+Tree and Hashmap in the persistent memory. They showcased that partial partly persistent implementations can also recreate the data structures along with the redundant data fields upon a system crash. Their solution can significantly improve the performance for a flush-dominated data structure. Ni *et al.* [89] presented performance studies on the interplay of DCPMM hardware and indexing data structures, and proposed group flushing and persistent optimized log-structuring techniques for improving the performance of indexing data structure on persistent memories. FlatStore [90] is an efficient PM-based key-value store particularly optimized for DCPMM. It decouples the data structure of a KV store into a persistent log structure for efficient storage and a volatile index for fast indexing. Due to the wider availability of DCPMM, more research studies of system design and implemen-

tation on real NVMMs emerge.

4 Performance Improvement and Energy Saving

As NVMMs show much higher access latency and write energy consumption, there have been a lot of studies on performance improvement and energy saving for NVMMs [32–34, 63, 65, 66]. These studies can be classified into three kinds: reducing the number of NVMM writes, reducing the energy consumption of NVM writes themselves, and reducing the energy consumption of DRAM through page migrations.

4.1 NVMM Write Reduction

To reduce NVMM writes, a hierarchical architecture is obviously more appropriate since the DRAM cache reduces abundant NVMM writes. Two major techniques namely page migration and bypassing NVMM writes have been developed for this purpose.

Page Migration. Page migration [14, 15, 33, 58, 59] policies choose the pages to be migrated mainly based on the number of writes and the recency of each page. Their main differences are in the conditions in which a page migration is triggered.

PDRAM [15] migrates PCM pages to DRAM according to the number of writes. In PDRAM, the memory controller maintains a table to record access counts of each PCM page. If the number of writes to a PCM page exceeds a given threshold, a page fault is triggered and then the page is migrated from the PCM page to DRAM.

CLOCK-DWF [33] integrates the write history of pages into the CLOCK algorithm. When a page fault occurs, the virtual page is fetched from the disk to the PCM. Otherwise, the page is allocated in DRAM as the page is likely to be a write-intensive one.

RaPP [14] migrates pages between DRAM and PCM based on the rank of pages. In RaPP, pages are ranked by the access frequency and recency. Top-ranked pages are migrated from PCM to DRAM. Thus, frequently written pages are placed in DRAM while seldom written pages are placed in PCM. Moreover, RaPP also places mission-critical pages in DRAM to improve application performance. By monitoring the number of writeback operations for each page in LLC, the memory controller is able to track the access frequency and recency of each page. RaPP ranks pages according to the Multi-Queue (MQ) algorithm [118]. A conventional MQ defines multiple least recently used (LRU) queues. Each LRU queue is a queue of page descriptors which include a reference counter and a logical expiration time. When a page is accessed at the first time, the page is moved to the tail of queue 0. If the reference count of the page reaches 2^{i+1}, the page is prompted to queue $i+1$. Once a PCM page is moved to queue 5, it is migrated to DRAM.

Buffering NVMM Writes. In a hybrid memory system, caches are able to reduce a large number of writes to NVMM. A proper cache replacement policy not only improves application performance, but also reduces the energy consumption of NVMM. Previous studies [7, 18] have found that many blocks in cache would not be reused again before they are evicted from the cache. These blocks are called dead blocks and consume precious cache capacity. DASCA [7] proposes a dead block prediction method to reduce the energy consumption of STT-RAM caches. Evicting these dead blocks will reduce the writes to STT-RAM caches and does not affect the cache hit rate. WADE [62] further exploits the asymmetry of energy consumption between NVMM read and NVMM write. As NVMM write operations consume much more energy than NVMM read operations, the frequently-written blocks should be kept in the cache. WADE divides the blocks in cache into two categories: frequently written-back blocks and non-frequently written-back blocks. Non-frequently written-back blocks are replaced to offer more opportunities for keeping other data blocks in the cache.

4.2 NVMM Energy Consumption Reduction

Since an NVMM write shows several times higher energy consumption than an NVMM read, there have been many efforts in reducing the energy consumption of NVMM writes. These approaches can be divided into two categories: differential write (only write dirty bits rather than a whole line), and parallel multiple writes during a single write.

Flip-N-Write [64] tries to reduce PCM write energy consumption by flipping the bits if the number of bits to be written exceeds half of the total bits in a cache line. During a single write, if more than half of bits in the line are written, each bit is flipped and thus the bit flips are no more than 50% of total bits. Meanwhile, a tag bit is set to identify whether the bits in a line are flipped. When the line is read, the tag bit is used to determine whether the bits in the line should be flipped.

Similar to Flip-N-Write, Andrew *et al.* [73] advocated fine-grained write. It only monitors dirty bits rather than all bits in a line. A new term called PCM

power token was introduced to indicate the power supply during a single write. Assume each chip is assigned P_{limit} Watts power and each bit-write requires P_{bit} Watts, $\lfloor P_{\text{limit}}/P_{\text{bit}} \rfloor$ bits can be written simultaneously. Within a chip, banks can be written concurrently. During a single write, if a number of write requests locate in different banks and the total power consumption does not exceed P_{limit}, these writes can be executed simultaneously. Thus, the fine-grained write not only reduces the NVMM writes, but also improves system performance by achieving higher bank parallelism.

A few studies [65, 66] improve the energy efficiency of NVMMs by separating the SET and RESET operations. As NVMMs consume more energy and time to write 1 than to write 0, both the write latency and the energy consumption can be reduced if these writes are performed in a proper manner. Three-stage-write [65] divides a write operation into a comparison stage, a write-zero stage and a write-one stage. In the comparison stage, the Flip-N-Write mechanism is exploited to reduce the number of writes. The zero bits and one bits are written separately in the write-zero stage and the write-one stage, respectively. Because write-zero operations account for a majority of write operations in most workloads, Tetris Write [66] further takes the asymmetry of SET and RESET operations into account, and schedules the costly write-one operations in parallel. The write-zero operations are inserted in the remaining interval of write-one operations under the power constraint.

CompEx [67] proposes a compression expansion encoding mechanism to reduce the energy consumption for MLC/TLC NVMMs. To improve the lifetime of MLC/TLC cells, the data is compressed first to reduce data redundancy. An expansion code is then applied to the compressed data and written to physical NVMM cells. For a TLC cell with 8 states, states 0, 1, 6 and 7 are called terminal energy state while states 2, 3, 4 and 5 are called central energy states. Central energy states consume more time and energy as they need more program and verify iterations. CompEx leverages the expansion code to use only terminal energy state for NVMM cells. This idea is motivated since the terminal energy state needs less energy and time than the central energy states when programming an MLC/TCL cell.

Hybrid on-chip caches were also proposed to reduce the power consumption of CPUs. RHC [68] constructs a hybrid cache, in which each way in SRAM and NVMM can be powered on or off independently. If a row has not been accessed for a long time, the row is powered off while its tag is still powered on to track the accesses of this row. When the accesses to the tag exceed a threshold, the row is powered on. To best utilize the high-performance SRAM and the low dynamic-power NVMM, RHC adopts different thresholds for SRAM and NVMM.

4.3 DRAM Energy Consumption Reduction

In a memory system with only DRAM, the static energy consumption can account for more than half of total energy consumption of memory systems [69–71]. In hybrid memory systems, page migration techniques are widely used to mitigate the energy consumption of DRAM. The inactive pages can be migrated from DRAM to NVMM so that the idle DRAM banks can be powered off. When the page becomes active later, it is migrated to DRAM again. However, if the page migration is not properly performed, the DRAM ranks may be frequently powered off and reactivate. The extra energy consumption is likely to offset the benefit gained by page migrations.

To reduce energy consumption in hybrid memory systems, RAMZzz [8] reveals two major roots of high energy cost. One is the sparse distribution of active pages, and the other is that page migrations may not be effective since the transfer among multi-energy states of DRAM introduces additional energy consumption. To solve the former problem, RAMZzz uses multiple queues to collect pages with a similar activity into the same DRAM rank, avoiding frequent energy state transfers. The multiple queues have L LRU queues to record the page descriptors. A page descriptor contains the page's ID and access (both read and write) counts in a period of time. To reduce the energy overhead of data migration, the pages with a similar memory access behavior are regrouped together. In this way, the pages need to be allocated to new banks. RAMZzz migrates these pages between banks in parallel.

Refree [72] further reduces the DRAM energy consumption in hybrid memory systems by avoiding DRAM refresh. When a DRAM row requires to be refreshed, it means the row has not been accessed for a long time. The data in the row is obsolete and it is not likely accessed again in the near further. Refree evicts these rows to PCM rather than refreshes them in DRAM. In Refree, all rows are monitored periodically. The interval of this period is equal to half of the retention time of a DRAM row since its last refresh. Therefore, rows are divided into active rows and non-active rows. The active rows are charged when they are

accessed. Non-active rows are evicted to PCM so that DRAM refreshes are eliminated.

5 Write Endurance Improvement

In a hybrid memory system, there are mainly two strategies to overcome the limited write endurance of NVMMs. One is to reduce NVMM writes, and the other is wear-leveling which spreads the write traffic evenly among all NVMM cells.

5.1 Write Reduction

There have been many write reduction strategies proposed for improving the lifetime of NVMMs, including data migration [8, 14, 15], caching or buffering [16], and inner-NVM write reduction [64, 73, 74].

A lazy write mechanism [16] was proposed to reduce the writes to PCM. In a hierarchical hybrid memory system, a DRAM buffer is used to hide the high-latency PCM accesses. When a page fault occurs, the data is fetched from the disk into the DRAM cache directly. The page is not written to PCM until the page is evicted from the DRAM cache. Line-level writes can also relieve write operations on NVMMs and thus reduce the wear of NVMMs [16]. For memory-intensive workloads, the write operations may be concentrated in a few lines. By tracking the cache line in DRAM, only the dirty lines are written back to PCM other than all lines of the page. Memory compression mechanisms [67, 75] were proposed to improve the lifetime of MLC/TLC NVMM. The data is compressed first before writing to NVMM cells. Therefore, only a small part of NVMM cells are written. However, the enhancement of endurance is at the expense of a moderate performance degradation. If an NVMM cell is written with a lower dissipated power, the cell can sustain more writes at the expense of higher write latency. Specifically, when the speed of writing an NVMM cell declines N times, the endurance of the cell can be improved by N to N^3 times. Mellow-Write [76] explores this feature to improve the lifetime of NVMMs. To mitigate the performance degradation, Mellow-Write only adopts slow writes to bank with only one write operation.

5.2 Wear-Leveling

Different from write reduction methods, wear-leveling spreads writes among all NVMM pages evenly. Although the total number of writes is not reduced, wear-leveling techniques can prevent some pages from being worn out by intensive writes quickly.

For NVMMs, we can record the write counts of each line to guide the wear-leveling policies. However, the external storage overhead cannot be ignored. Start-Gap [77] proposes a fine-grained wear-leveling scheme. The lines of a PCM page are stored in a rotating manner. A rotating value is generated randomly between 0 and 15 to indicate the shifted positions. For a PCM page with 16 lines, the rotating value can range from 0 to 15. When the rotating value is 0, the page is stored in its original address. If the rotating value is 1, line 0 is stored in line 1's physical address, and each line's address is shifted by the rotated value.

In PDRAM [15], wear-leveling is triggered by a threshold of write counts. When the write counts of a page exceed the given threshold, a page swapping interrupt is triggered to migrate the page to DRAM. The swapped PCM page is added to a list in which these pages will be relocated again.

Zombie [78] offers another direction to achieve weal-leveling, and further extends the overall lifetime of PCM. Other than Start-Gap that distributes writes among PCM cells evenly, Zombie leverages spare blocks in disabled pages to provide more error correction resource for working memory. When a PCM cell is worn out, it becomes unavailable. As memory footprint is organized in pages from the perspective of software, the whole page which contains the failure cell is disabled. However, if some spare cells are provided to replace the failed cells, the page can be used again. These spare cells are called error correction resource. When all spare cells are exhausted, the page with failed cells is abandoned finally. Usually, there are about 99% bits available when a page is disabled. Zombie utilizes the large number of good bits in disabled pages as the spare error correction resource, in which good bits are organized in fine-grained blocks. By pairing the working page with error correction resources, Zombie can extend the lifetime of NVMMs much longer.

DRM [79] adds an intermediate mapping layer between the virtual address space and the physical NVMM address space. In the intermediate address space, a page may map to a good page in PCM or two compatible PCM pages with faults. The compatible page means a pair of pages with fault bytes, but none of these fault bytes locate in the same place of the two pages. Thus, two compatible pages can be combined to form a new good page. In this way, DRM significantly improves PCM lifetime by 40x.

6 Practices of Hybrid Memory System Designs

In this section, we introduce our recent efforts and practices of system designs and optimizations on NVMMs from the perspective of memory architecture, OS-supported hybrid memory management, and NVMM-supported applications, as shown in Fig.5. In the following, we will present our practices briefly.

6.1 Memory Architectural Designs

In this subsection, we present our studies on hybrid memory simulation and emulation, hardware/software cooperative hybrid memory architecture, fine-grained NVM compression and wear leveling, and hybrid memory aware on-chip cache management.

6.1.1 Hybrid Memory Architectural Simulation

A hybrid memory architectural simulation is a prerequisite for studying hybrid memory systems. We integrate zsim [27] with NVMain [26] to build a full-system architectural simulator. Zsim is a fast processor simulator for x86-64 multi-core architectures. It is able to model multi-cores, on-chip cache hierarchy, cache coherence protocols such as MESI, on-chip interconnect topology network, and physical memory interfaces. Zsim collects the memory trace of processes using Intel Pin toolkit, and then replays the memory trace to characterize the memory access behaviors. NVMain is an architectural-level main memory simulator for NVMMs. It is able to simulate different profiles of memories such as read/write latency, bandwidth, power consumption, and so on. It also supports subarray-level memory parallelism and different memory address encoding schemes. Moreover, NVMain can also model hybrid memories such as DRAM and different NVMMs in the memory hierarchy. Since OS-level memory management is not simulated by zsim, we extend zsim by adding Translation Lookaside Buffer (TLB) and memory management modules (such as buddy memory allocator and page tables) to support a full-system simulation. Implementation details are refereed to our open-source software④. Our work provides a fast, and full-system architectural simulation framework to the research community. It can help researchers to understand different NVMM features, design hybrid memory systems, and evaluate the impact of various system designs on application performance in an easy and efficient manner.

6.1.2 Lightweight NVMM Performance Emulator

Current simulation-based approaches for studying NVMM technologies are too slow, or cannot run complex workloads such as parallel and distributed applications. We propose HME [28], a lightweight NVMM

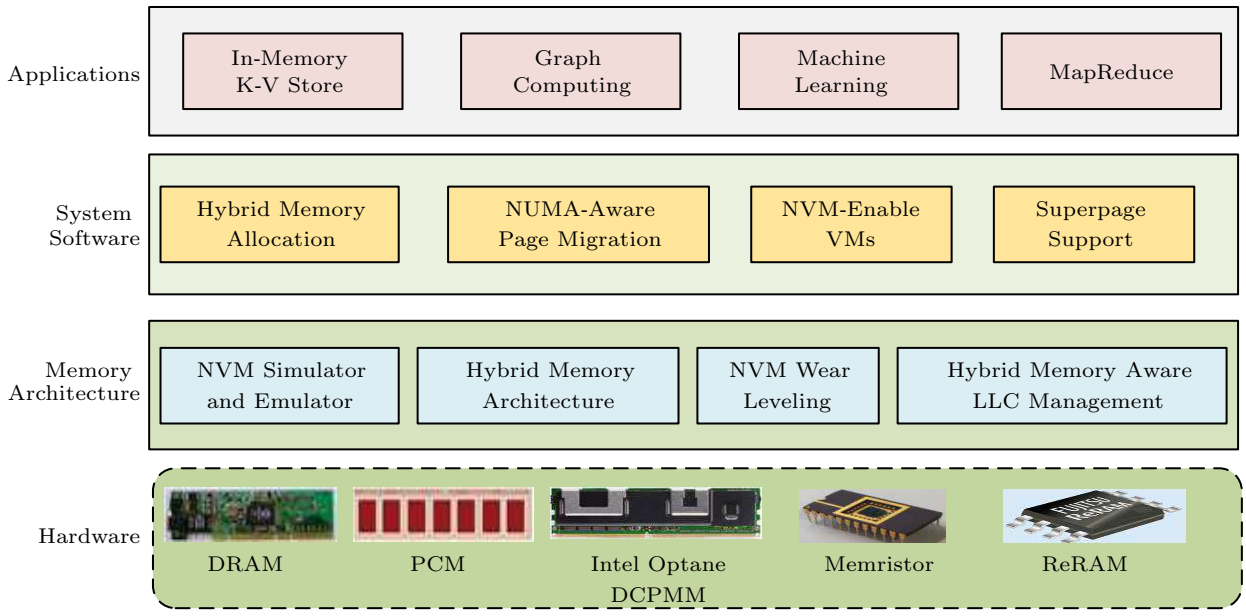


Fig.5. Our practices of system designs on hybrid memories.

④https://github.com/CGCL-codes/HSCC, Dec. 2020.

performance emulator using Non-Uniform Memory Access (NUMA) architectures. HME exploits hardware performance counters available in commodity Intel CPUs to emulate the performance features of slower NVMMs. To emulate the access latency of NVMMs, HME injects software-generated latency into DRAM accesses on the remote NUMA nodes periodically. To mimic the NVMM bandwidth, HME utilizes DRAM thermal control interfaces to throttle the amount of memory requests to a DRAM channel in a short period of time. Unlike another NVMM emulator Quartz [29] that does not emulate the write latency of NVMMs, HME identifies write-through and write-back cache eviction operations to emulate their latencies separately. In this way, HME is able to significantly reduce emulation errors of NVMM access latencies on average compared with Quartz [29]. Before the advent of real NVMM device — Intel Optane DCPMM, this work can help researchers and programmers to evaluate the impact of NVMM performance characteristics on applications, and guide the system designs and optimizations on hybrid memory systems.

6.1.3 *Hardware/Software Cooperative Caching*

Based on our hybrid memory simulator, we propose a hardware/software cooperative hybrid memory architecture called HSCC [18]. In HSCC, DRAM and NVMM are physically organized in a single memory address space and are all used as main memory. However, DRAM can be logically used as a cache of NVMM and also managed by OSes. Fig.6 shows the system architecture of HSCC. We extend page tables and TLB to maintain the NVMM-to-DRAM physical address mappings, and thus manage DRAM/NVMM in the form of a cache/memory hierarchy. In this way, HSCC is able to perform NVMM-to-DRAM address translation as efficient as virtual-to-NVMM address translation. Also, we add an access counter in each TLB entry and page table entry to monitor memory references. Unlike previous approaches monitoring memory accesses in the memory controller or OSes, our design can track all data accesses accurately with trivial storage (SRAM) and performance overhead. We identify frequently accessed (hot) pages through a dynamic threshold adjustment strategy to adapt to different applications, and then the hot pages in NVMM are migrated to DRAM cache for higher performance and energy efficiency. Moreover, we develop a utility-based DRAM cache filling scheme to balance the efficiency of DRAM cache and DRAM utilization. As the software-managed DRAM pages are able to map to any NVMM pages, DRAM is actually used as a fully associative cache. This approach can significantly improve the utilization of DRAM cache, and also offers opportunities to reconfigure the hybrid memory architecture according to dynamic memory access behaviors of applications. As CPUs can bypass the DRAM cache to directly access cold data in NVMM, DRAM can be used either as main memory in the flat-addressable hybrid memory architecture, or as a data filter cache of NVMM in the

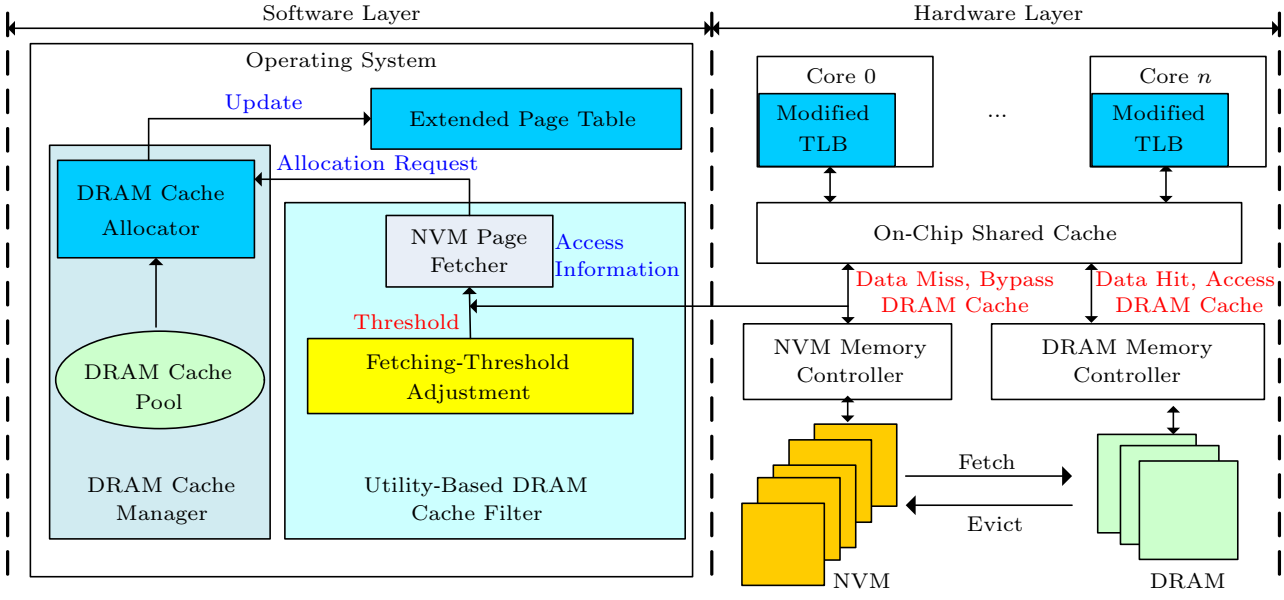


Fig.6. System architecture of HSCC [18].

hierarchical hybrid memory architecture. As a result, HSCC can significantly improve system performance by up to 9.6x and reduce energy consumption by 34.3% compared with the state-of-the-art work [16]. Our work offers the first architectural solution to achieve reconfigurable hybrid memory systems that can dynamically change the DRAM/NVMM management between horizontal and hierarchical memory architectures.

We further propose the following techniques on HSCC to improve the cache performance and improve the wear-leveling mechanisms.

As the cache miss penalty for an NVMM block is several times higher than that for a DRAM block, the cache hit rate is not the only one performance metric that should be improved in a flat-addressable hybrid memory architecture. To best utilize the expensive LLC, we propose a new metric, i.e., Average Memory Access Time (AMAT), to assess the overall performance of hybrid memory systems. We take the asymmetrical cache miss penalty of DRAM blocks and NVMM blocks into account, and propose an LLC miss penalty aware replacement algorithm called MALRU [36, 37] to improve AMAT in hybrid memory systems. MALRU partitions LLC into a reserved region and a normal replacement region dynamically. MALRU preferentially replaces dead and cold DRAM blocks in LLC so that NVMM blocks and hot DRAM blocks are kept in the reserved region. In this way, MALRU achieves the application performance improvement by up to 22.8% compared with the LRU algorithm. This work showcases how the hybrid memory system can effect the architectural design of on-chip cache.

To improve the write endurance of NVMM, we propose a new NVMM architecture to support space-oblivious data compression and wear-leveling [119]. As memory blocks of many applications usually contain a large amount of zero bytes and frequent values, we propose Zero Deduplication and Frequent Value Compression mechanisms (called ZD-FVC [119]) to reduce bit-writes on NVMM. ZD-FVC can be integrated into the NVMM module and implemented entirely by hardware, without any intervention of operating systems. We implement ZD-FVC [119] in Gem5 and NVMain simulators, and evaluate it with several programs from SPEC CPU2006. Experimental results show that ZD-FVC is much better than several state-of-the-art approaches. Particularly, DZ-FVC can improve data compression ratio by 1.5x compared with Frequent Value Compression. Compared with Data Comparison Write, ZD-FVC is able to reduce bit-writes on NVMM by 30%, and significantly improve the lifetime of NVMM by 5.8x on average. Correspondingly, ZD-FVC also reduces NVMM write latency by 43% and energy consumption by 21% on average. Our design provides a fine-grained data compression and wear-leveling solution for NVMMs in simple and efficient manner. It is complementary to other wear-leveling schemes to further improve NVMM lifetime.

6.2 System Software for Hybrid Memories

In this subsection, we present our practices of hybrid memory systems in the software layer, including object-level hybrid memory allocation and migration, NUMA-aware page migration, superpage supporting, and NVMM virtualization mechanisms.

6.2.1 Object Migration in Hybrid Memory Systems

Page migration techniques have been widely exploited to improve system performance and energy efficiency in hybrid memory systems. However, previous page migration schemes all rely on costly online page access monitoring schemes in the OS layer to track page access recency or frequency. Moreover, data migration at the page granularity often results in non-trivial performance overhead because of additional memory bandwidth consumption and cache/TLB consistency guarantee mechanism.

To mitigate the performance overhead of data migration in hybrid memory systems, we propose more lightweight object-oriented memory allocation and migration mechanisms called OAM [120]. The framework of OAM is shown in Fig.7. Unlike previous studies [44, 121] that only profile memory access behaviors in a global view for static object placement, we further analyze object access patterns in fine-grained time slots. OAM leverages a compile framework LLVM to profile application memory access patterns at the object granularity, and then divides the execution of applications into different phases. OAM exploits a performance/energy integrated model to guide the initial memory allocation and runtime object migration in different execution phases, without intrusive modifications of hardware and OSes for online page access monitoring. We develop new memory allocation and migration APIs by extending the Glibc library and Linux kernel. Based on these APIs, programmers are able to allocate DRAM or NVMM to different objects explicitly, and then migrate the objects whose access patterns are dynamically changed between

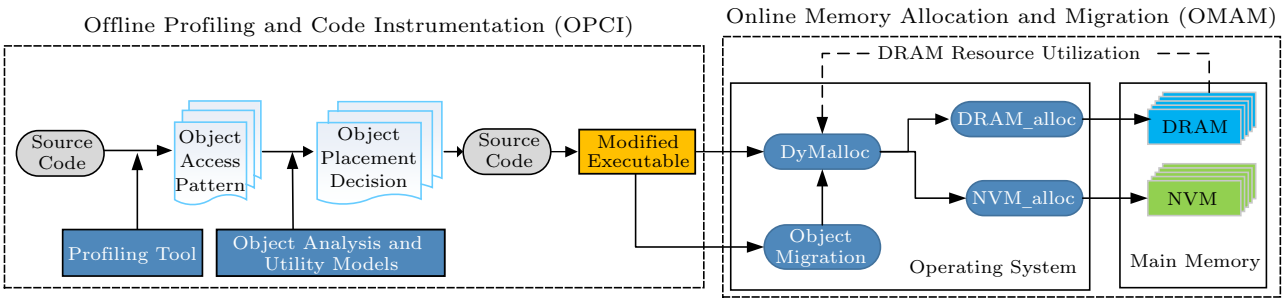


Fig.7. Overview of OAM framework [120].

DRAM and NVMM. We develop a static code instrumentation tool to automatically modify legacy applications' source codes, without re-engineering the applications by programmers. Compared with the state-of-the-art page migration approaches CLOCK-DWF [33] and 2PP [44], experimental results show that OAM can significantly reduce data migration cost by 83% and 69%, respectively, and achieve about 22% and 10% application performance improvement. Previous persistent memory management schemes often rely on memory access profiling to guide static data placement, and page migration (costly) techniques to adapt to dynamic memory access patterns at runtime. OAM provides a more lightweight hybrid memory management scheme which supports fine-grained object-level memory allocation and migration.

6.2.2 *NUMA-Aware Hybrid Memory Management*

In Non-Uniform Memory Access (NUMA) architectures, application-observed memory access latencies in different NUMA nodes are usually asymmetrical. Because NVMM is several times slower than DRAM, hybrid memory systems can further enlarge the performance gap among different NUMA nodes. Traditional memory management mechanisms for NUMA systems are no longer effective in hybrid memory systems and may even degrade application performance. For example, the automatic NUMA balancing (ANB) policy always migrates application data in a remote NUMA node to an NUMA node in which the application threads or processes are running. However, since the access performance of remote DRAM may be even higher than that of local NVMM, ANB may falsely move the application data to a slower place. To address this problem, we propose HiNUMA [60], a new NUMA abstraction for hybrid memory management. When application data is first placed in the hybrid memory system, HiNUMA places application data on both NVMM and DRAM to balance memory bandwidth utilization and total access latency for bandwidth-sensitive applications and latency-sensitive applications, respectively. The initial data placement is based on the NUMA topology and hybrid memory access performance. For runtime hybrid memory management, we propose a new NUMA balancing policy named HANB [60] for page migrations. HANB is able to reduce the total cost of hybrid memory accesses by taking both data access frequency and memory bandwidth utilization into account. We implement HiNUMA in Linux kernel, without any modifications of hardware and applications. Compared with traditional memory management policies in NUMA architectures and other state-of-the-art work, HiNUMA can significantly improve application performance by efficiently utilizing hybrid memories. The lessons learned from HiNUMA [60] are also applicable to hybrid memory systems equipped with real Intel Optane DCPMM device.

6.2.3 *Supporting Superpages in Hybrid Memory Systems*

With a rapid growth of application footprint and the corresponding memory capacity, virtual-to-physical address translation has become a new performance bottleneck for hybrid memory systems. Superpages have been widely exploited to mitigate address translation overhead in big-memory systems. However, the side effect of using superpages is that they often impede lightweight memory management such as page migration, which is widely exploited in hybrid memory systems to improve system performance and energy efficiency. Unfortunately, it is challenging to have both world of superpages and lightweight page migration.

To address this problem, we propose a novel hybrid memory management system called Rainbow [41] to bridge the fundamental conflict between superpages and lightweight page migration. As shown in Fig.8, Rainbow manages NVMM at the granularity of superpages (2 MB), and manages DRAM as a cache to

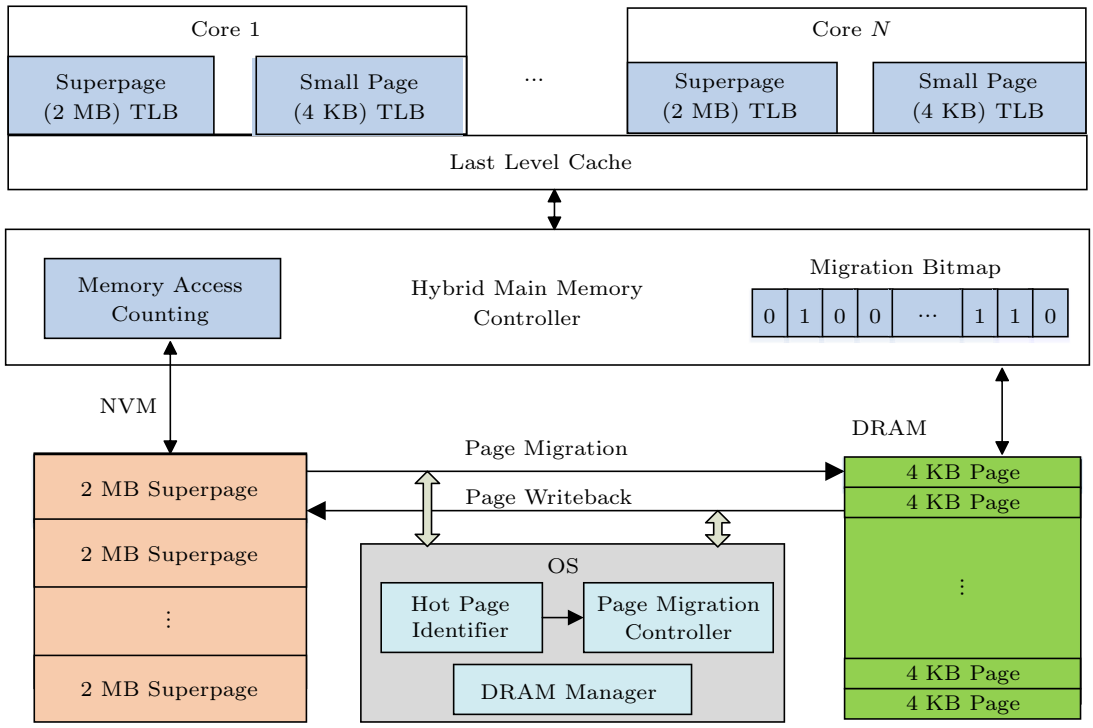


Fig.8. Architecture of Rainbow [41].

store hot data blocks in superpages at the granularity of base pages (4 KB). To speed up address translations, Rainbow employs the existing hardware feature of split TLBs to support superpages and normal pages.

We propose a two-stage page access monitor mechanism to identify hot base pages within superpages. In the first stage, Rainbow records the access counts of all superpages to identify top-N hot superpages. In the second stage, we logically split those hot superpages into base pages (4 KB) and further monitor them to recognize hot base pages. The schemes significantly diminish the SRAM storage overhead for page access counters and runtime performance overhead due to sorting the hot base pages. With a new NVMM-to-DRAM address remapping mechanism, Rainbow is able to migrate hot base pages to DRAM while still guaranteeing the integrity of superpage TLB. The split superpage TLBs and base page TLBs are consulted in parallel. Our address remapping mechanism logically uses superpage TLBs as a cache of the base page TLBs. Because the hit rate of superpage TLB is often very high, Rainbow is able to significantly accelerate base page address translation. To further improve TLB hit rate, we also extend Rainbow to support multiple page sizes and migrate contiguous hot base page together [42]. Compared with a state-of-the-art hybrid memory system [18] without superpage support, Rainbow can significantly improve application performance by at most 2.9x by having the benefit of using both superpages and lightweight page migration.

This work provides a hardware/software cooperative design to bridge the fundamental conflict between superpages and lightweight page migration techniques. This may be a promising solution to mitigate the ever-increasing virtual-to-physical address translation overhead in large-capacity hybrid memory systems.

6.2.4 *NVMM Management in Virtual Machines*

NVMMs are expected to be more popular in cloud and data center environments. However, there have been few studies on using NVMMs for virtual machines (VMs). We propose HMvisor [61], a hypervisor/VM cooperative hybrid memory management system to utilize DRAM and NVMM efficiently. As shown in Fig.9, HMvisor exploits a pseudo-NUMA mechanism to support hybrid memory allocation in VMs. Since virtual NUMA nodes in a VM can be mapped to different physical NUMA nodes, HMvisor can map different memory regions to a single VM and thus expose memory heterogeneity to VMs.

To support lightweight page migration in VMs, HMvisor monitors page access counts and classifies

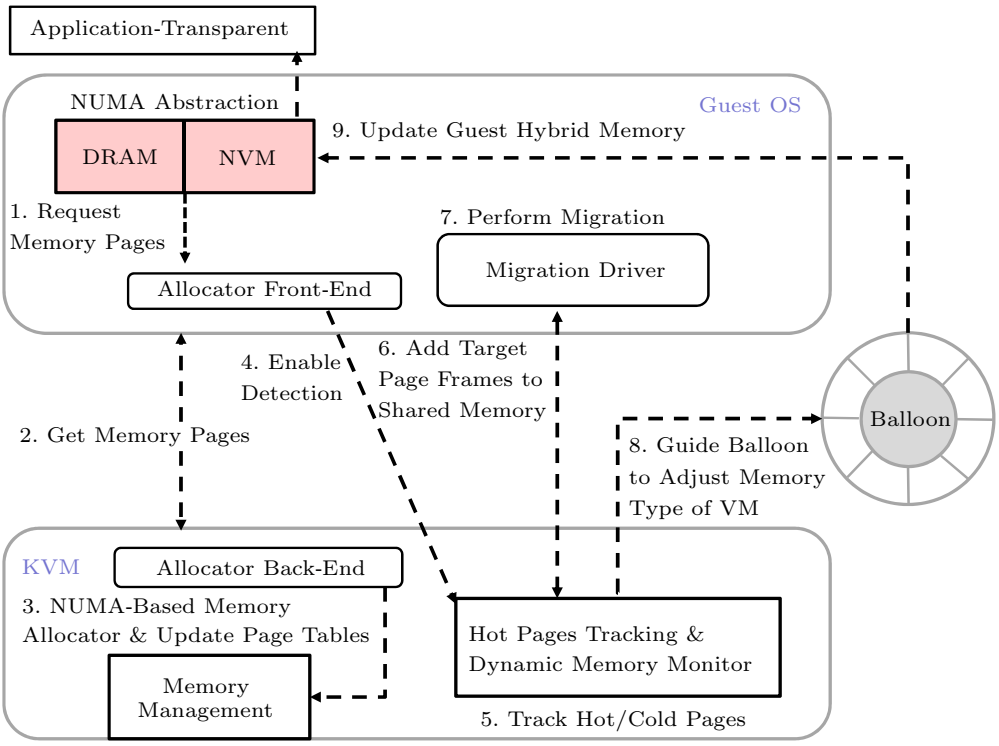


Fig.9. System overview of HMvisor [61].

hot pages and cold pages in the hypervisor, and then the VM periodically collects the information of hot pages through an inter-domain communicate mechanism. We implement a loadable driver in the VM to perform process-level page migrations between DRAM and NVMM. Since HMvisor performs page migrations by the VM itself, HMvisor does not need to suspend the VM for page migrations. HMvisor also advocates a hybrid memory resource trading policy to dynamically adjust the size of NVMM and DRAM in a VM. In this way, HMvisor can meet different memory requirements (capacity or performance) of diversifying applications while keeping the total monetary cost of the VM unchanged.

The prototype of HMvisor is implemented in the QEMU/KVM platform. Our evaluation shows that HMvisor is able to reduce NVMM write traffic by 50% at the expense of only 5% performance overhead. Furthermore, the dynamic memory adjustment policy can significantly reduce major page faults in a VM when it suffers high memory pressure, and thus can even improve application performance by 30 times.

This is an early system work to manage hybrid memory in a virtualization environment. The proposed schemes are completely implemented by software, and thus also applicable to hybrid memory systems comprising of the new Intel Optane DCPMM device.

6.3 NVMM-Supported Applications

Since hybrid memory systems can provide very large capacity of main memory, they have been widely explored for big data applications such as in-memory key-value (K-V) stores and graph computing. In this subsection, we present our practices of NVMM-supported system optimizations for those applications.

In-memory K-V stores with large-capacity memory can cache more hot data in main memory, and thus deliver higher performance to applications. However, there are several challenges to directly deploy traditional K-V stores such as memcached in hybrid memory systems. For example, how to identify hot K-V objects efficiently? How to redesign an NVMM-friendly K-V indexes to reduce NVMM writes? How to redesign the cache replacement algorithm to balance object access frequency and recency in hybrid memory systems? How to address the slab calcification problem [122] to best utilize the DRAM resource in hybrid memory systems?

To address the above problems, we propose HMCached [80], an extension of K-V cache (memcached) for hybrid DRAM/NVMM systems. Fig.10 shows the system architecture of HMCached. HMCached tracks K-V object accesses and records the ac-

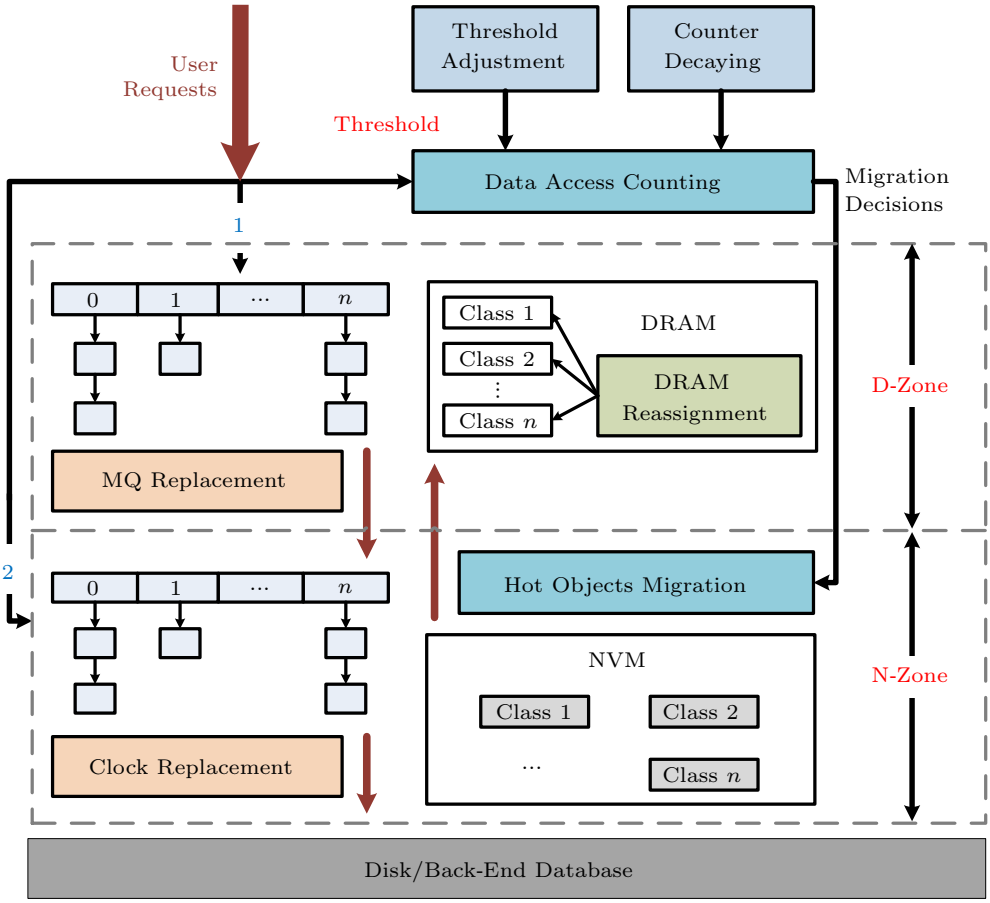


Fig.10. Architecture of HMCached [80].

cess counts in each K-V pair's metadata structure so that HMCached can easily identify frequently-accessed (hot) objects in NVMM and migrates them to DRAM. In this way, we logically use DRAM as an exclusive cache of NVMM to avoid more costly NVMM accesses. Moreover, we redesign an NVMM-friendly K-V data structure by splitting the hash-based K-V indexes to further reduce NVMM accesses. We put the frequently-updated metadata (e.g., reference counts, timestamp, and access counts) of K-V objects in DRAM, and the remaining portion (e.g., keys and values) in NVMM. We exploit a multi-queue algorithm [118] to take both object access frequency and recency into account for DRAM cache replacement. Moreover, we set up a utility-based performance model to evaluate the benefit of slab class reassignment. Our dynamic slab reallocation policy is able to address the slab calcification problem effectively, and significantly improve application performance when the data access pattern changes. Compared with the vanilla memcached, HMCached can significantly reduce NVMM accesses by 70% and achieves about 50% performance improvement. Moreover, HMCached is able to reduce 75% DRAM cost while the performance degradation is less than 10%.

To the best of our knowledge, we are the first to explore the object-level data management for K-V stores in hybrid memory systems. We implement HMCached based on Memcached and open the source codes⑤. We find that later studies such as flatstore [90] have a similar idea to decouple data structures of KV stores.

Today we have witnessed a number of in-memory graph processing systems in which application performance is highly bound to the capacity of main memory. High-density and low-cost NVMM technologies are essential to mitigate I/O cost for graph processing. As shown in Fig.11, a hybrid memory system can significantly improve application performance compared with an SSD-based storage system. Fig.12 shows the application performance gap between a hybrid memory system and a DRAM-only system. Although the gap is

⑤https://github.com/CGCL-codes/HMCached, Dec. 2020.

acceptable, it indicates that there are still opportunities to further exploit the advantages of NVMM and DRAM for in-memory graph processing systems.

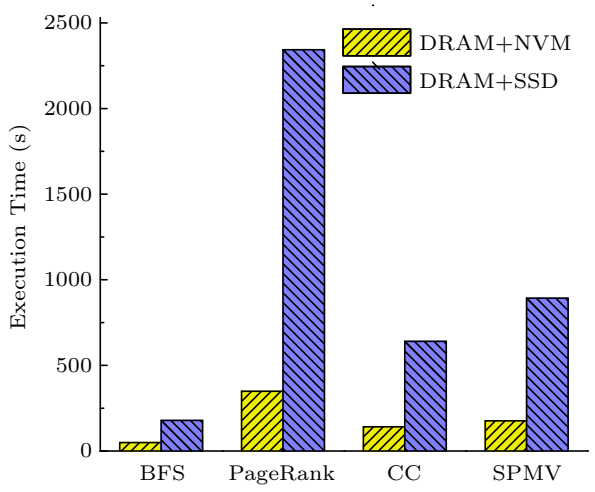


Fig.11. Difference of application execution time in "DRAM+NVM" and "DRAM+Disk" systems [91].

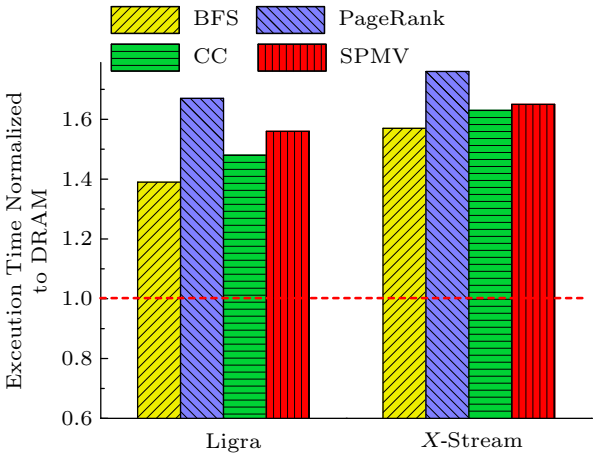


Fig.12. Normalized execution time of applications in a hybrid memory system and a DRAM-only system [91].

We propose NGraph [91], a new graph processing framework particularly designed to better utilize hybrid memories. We develop hybrid memory aware data placement policies based on access patterns of different graph data to mitigate random and frequent accesses to NVMM. Generally, graph structure data accounts for a majority of total graph data. NGraph partitions graph data according to destination vertices and employs a task decomposition mechanism to avoid the data contention between multiple processors. Moreover, NGraph adopts a work stealing mechanism to minimize the maximum time of parallel graph data processing on multicore systems. We implement NGraph based on a graph processing framework Ligra [123]. NGraph can improve application performance by up to 48% compared with the state-of-the-art Ligra. The lessons learned from this work [91] can be exploited to further improve the performance of large-scale graph analytics in a graph processing platform equipped with real PM device.

7 Research Directions

The advent of NVMM technologies has aroused many interesting research topics in the area of material, microelectronics, computer architecture, system software, programming model, and big data applications. As real NVMM devices such as Intel Optane DCPMM have been increasingly applied to data center environments, NVMMs may change the storage landscape of data centers. Our experiences and practices have demonstrated some preliminary and interesting studies on those dimensions. In the following, we share our vision of future research directions of NVMMs, and analyze the research challenges and new opportunities. Fig.13 illustrates the future trends of NVMM technologies in different dimensions.

1) *The Development of of 3D-Stacked NVMM Technologies Is Still Continuing.* NVMMs are expected to provide higher integration density for cost reduction. Currently, the high-end NVDIMMs are still too expensive for enterprise applications. The key challenge for NVMMs to compete with traditional DRAM and NAND flash is the storage density or the cost per byte. There are mainly two monolithic 3D integration mechanism for NVMM technologies [124]. One is to stack the horizontal cross-point array layer by layer, such as Intel/Micron 3D X-point. The other is the vertical 3D stacked structure that is referred to ReRAM technology. However, the 3D integrate technologies are not full-blown. There still remain many challenges such as fabrication cost, pillar electrode resistance, and sneak path problems.

2) *NVMMs Are Increasingly Used in Distributed Shared Memory Systems.* As the density of NVMMs is continuously increasing, the main memory capacity can approach hundreds of Terabytes in a single server. To improve the utilization of large-capacity NVMMs, it is essential to share them among multiple servers via remote direct memory access (RDMA) techniques. A typical approach of using NVMMs is to aggregate all shareable memories from multiple servers in a hybrid shared memory resource pool, such as

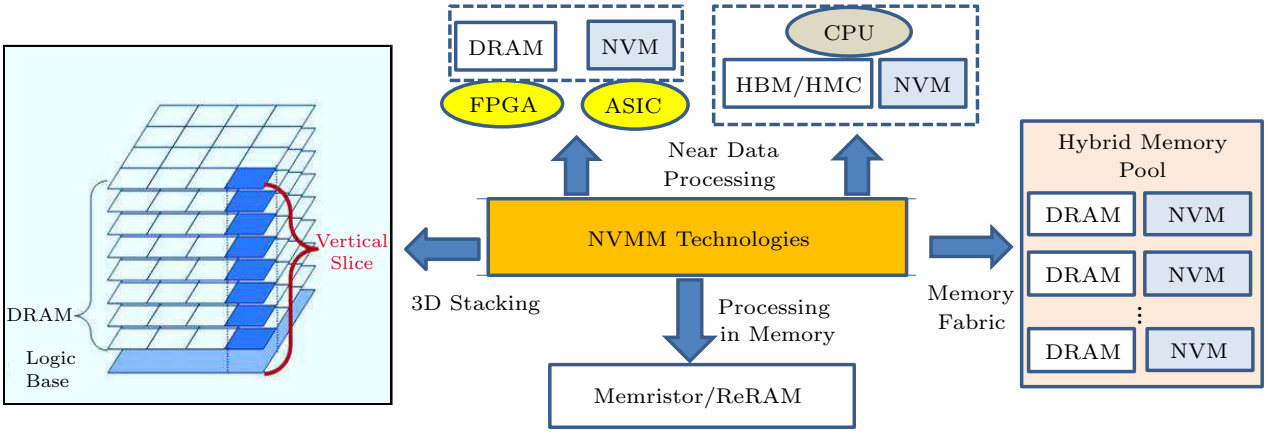


Fig.13. Future directions of NVMM technologies.

Hotpot [49, 125, 126]. All memory resources are shared in a global memory space. There have been a few preliminary studies on using NVMMs in datacenter and cloud environments [49, 126, 127]. A new trend of using PM is to manage it as disaggregated memory, like traditional disaggregated storage systems. This model is different from the previous shared PM systems in which the PM DIMMs are distributed in multiple servers and shared by user-level applications. These computation-memory tightly-coupled architectures have several drawbacks in terms of manageability, scalability, and resource utilization. In contrast, the disaggregated PM systems equipped with a large amount of PM in a few memory nodes can be connected by computation nodes via high-speed fabric. This computation/memory disaggregated architecture can mitigate the above challenges in data center environments more easily. However, there still remain a lot of challenges. For example, the persistent feature of NVMMs also should be guaranteed in distributed environments. Traditional PM management instructions such as *clflush* and *mfence* can only guarantee data persistence in a single server, but cannot guarantee data persisting to a remote server over RDMA networks. For each RDMA operation, once the data arrives at the network interface card (NIC) in the remote server, it issues an acknowledgment to the data sender. As there are data buffers in NICs, the data is not stored to the remote NVMM immediately. If a power failure occurs at this time, data persistence is not guaranteed. Thus, it is essential to redesign the RDMA protocol to support flushing primitives. Moreover, the computation nodes should support remote page swapping which should be transparent to user-level applications. To support this mechanism, the traditional virtual memory management policies should be redesigned. On the other hand, since PM shows memory-like performance and is byte-addressable, new designs on memory scheduling and management are required to adapt to disaggregated PM.

3) *NVMM-Based Computation/Memory Integrated Computer Architectures Are Arising.* For example, the use of emerging NVMMs in processing-in-memory (PIM) [95, 96] and near data processing (NDP) [128, 129] architectures is arising. PIM and NDP have emerged as new computing paradigms in recent years. NDP refers to the integration of a processor with memory on a single chip so that the computation can access the data in memory as close as possible. NDP is able to significantly reduce the cost of data movement. There are mainly two approaches to this goal. One is to integrate small computation logics such as (FPGA/ASIC) into memory chips so that the data can be pre-processed before it is finally fetched to CPUs. The other approach is to integrate memory units (HBM/HMC) into computations (CPUs/GPGPUs/FPGAs). This model is commonly used by many processor architectures such as Intel® Xeon Phi™ Knights Landing series, NVIDIA® tesla V100, and Google Tensor Processing Unit (TPU). PIM refers to processing data entirely in computer memory. It offers high bandwidth, massive parallelism, and high-energy efficiency by performing computations in main memory. PIM using NVMMs (such as ReRAM) usually can compute the bitwise logic of two or more memory rows in parallel, and support one-step multi-row operations. This paradigm is particular efficient for matrix-vector multiplication in an analog computing manner, and can achieve an extremely large degree of performance speedup and energy saving. As a result, PIM is widely explored in accelerating machine learning algorithms such as convolutional neural networks

(CNN) and deep neural network (DNN). Although there are growing interests in using NVMM technologies in PIM architectures [94–96, 130], current studies are mainly based on electrical simulations, and none of them are available for mid-scale prototyping.

4) *Beyond the Traditional Applications, Some Novel Applications Using NVMMs Are Emerging.* Although NVMM technologies have been preliminarily adopted in a lot of big data applications, such as K-V store, graph computing, and machine learning, most of those programming frameworks/models and runtime systems are designed for disk devices and DRAM-based main memory, and they are not effective and efficient in hybrid memory systems. For example, buffering and lazy-write mechanisms are widely utilized in those systems to hide the high latency of I/O operations. However, those mechanisms may not be needed in hybrid memory systems and may even hurt application performance. These big data processing platforms such as Hadoop/Spark/GraphChi/Tensorflow should be redesigned to adapt to the features of NVMM technologies. Beyond those traditional applications, some novel applications based on NVMMs are emerging. For example, there have been a few proposals to use NVMMs as hardware security primitives such as physical unclonable functions (PUFs) by exploiting the intrinsic variations of NVMM's switching processes [131]. PUFs are typically used in applications with high-security requirements, for example, cryptography. Recently, a number of logic circuits based on NVMM technologies have been proposed and prototyped [132–134]. For example, the ReRAM technology is proposed to use as reconfigurable switch for ReRAM-based FPGAs [133]. Moreover, the STT-RAM technology is proposed to design non-volatile cache or registers [135].

8 Conclusions

Emerging NVMM technologies have many good features compared with traditional DRAM technologies. They have a potential to fundamentally change the landscape of memory systems and even add new functionalities and features to the computer systems. There are vast opportunities to rethink the designs of todays' computer systems to achieve orders of magnitude improvement in system performance and energy consumption. This paper presents a comprehensive survey of the state-of-the-art work and our practices from the perspective of memory architecture, OS-level memory management, and application optimizations. We also shared our vision of future research directions about NVMM technologies. By taking advantage of the unique features of NVMMs, there are enormous opportunities to innovate the future's computing paradigm and develop a lot of diverse novel applications of NVMMs.

References

[1] Ousterhout J, Gopalan A, Gupta A, Kejriwal A, Lee C, Montazeri B, Ongaro D, Park S J, Qin H, Rosenblum M, Rumble S, Stutsman R, Yang S. The RAMCloud storage system. *ACM Transactions on Computer Systems*, 2015, 33(3): Article No. 7. DOI: 10.1145/2806887.

[2] Zhang H, Chen G, Ooi B C, Tan K L, Zhang M H. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2015, 27(7): 1920-1948. DOI: 10.1109/TKDE.2015.2427795.

[3] Malladi K T, Shaeffer I, Gopalakrishnan L, Lo D, Lee B C, Horowitz M. Rethinking DRAM power modes for energy proportionality. In *Proc. the 45th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2012, pp.131-142. DOI: 10.1109/MICRO.2012.21.

[4] Mutlu O. Memory scaling: A systems architecture perspective. In *Proc. the 5th IEEE International Memory Workshop*, May 2013, pp.21-25. DOI: 10.1109/IMW.2013.6582088.

[5] Nair P J, Kim D H, Qureshi M K. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In *Proc. the 40th Annual International Symposium on Computer Architecture*, June 2013, pp.72-83. DOI: 10.1145/2485922.2485929.

[6] Wu X J, Reddy A L N. SCMFS: A file system for storage class memory. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011, Article No. 39. DOI: 10.1145/2063384.2063436.

[7] Ahn J, Yoo S, Choi K. DASCA: Dead write prediction assisted STT-RAM cache architecture. In *Proc. the 20th International Symposium on High Performance Computer Architecture*, Feb. 2014, pp.25-36. DOI: 10.1109/HPCA.2014.6835944.

[8] Wu D, He B, Tang X, Xu J, Guo M. RAMZzz: Rank-aware DRAM power management with dynamic migrations and demotions. In *Proc. the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2012, Article No. 32. DOI: 10.1109/SC.2012.99.

[9] Lu Y, Wu D, He B, Tang X, Xu J, Guo M. Rank-aware dynamic migrations and adaptive demotions for DRAM power management. *IEEE Transactions on Computers*, 2015, 65(1): 187-202. DOI: 10.1109/TC.2015.2409847.

[10] Lu Y, He B, Tang X, Guo M. Synergy of dynamic frequency scaling and demotion on DRAM power management: Models and optimizations. *IEEE Transactions on Computers*, 2015, 64(8): 2367-2381. DOI: 10.1109/TC.2014.2360534.

[11] Foong A, Hady F. Storage as fast as rest of the system. In *Proc. the 8th IEEE International Memory Workshop*, May 2016. DOI: 10.1109/IMW.2016.7495289.

[12] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory. In *Proc. the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2011, pp.91-104. DOI: 10.1145/2248487.1950379.

[13] Venkataraman S, Tolia N, Ranganathan P, Campbell R H. Consistent and durable data structures for non-volatile byte-addressable memory. In *Proc. the 9th USENIX Conference on File and Storage Technologies*, February 2011, pp.61-75.

[14] Ramos L E, Gorbatov E, Bianchini R. Page placement in hybrid memory systems. In *Proc. the 25th International Conference on Supercomputing*, May 31–June 4, 2011, pp.85-95. DOI: 10.1145/1995896.1995911.

[15] Dhiman G, Ayoub R, Rosing T. PDRAM: A hybrid PRAM and DRAM main memory system. In *Proc. the 46th ACM/IEEE Design Automation Conference*, July 2009, pp.664-669. DOI: 10.1145/1629911.1630086.

[16] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology. In *Proc. the 36th Annual International Symposium on Computer Architecture*, June 2009, pp.24-33. DOI: 10.1145/1555754.1555760.

[17] Yoon H, Meza J, Ausavarungnirun R, Harding R A, Mutlu O. Row buffer locality aware caching policies for hybrid memories. In *Proc. the 30th IEEE International Conference on Computer Design*, Sept. 30–Oct. 3, 2012, pp.337-344. DOI: 10.1109/ICCD.2012.6378661.

[18] Liu H, Chen Y, Liao X, Jin H, He B, Zheng L, Guo R. Hardware/software cooperative caching for hybrid DRAM/NVM memory architectures. In *Proc. the International Conference on Supercomputing*, June 2017, Article No. 26. DOI: 10.1145/3079079.3079089.

[19] Condit J, Nightingale E B, Frost C, Ipek E, Lee B, Burger D, Coetzee D. Better I/O through byte-addressable, persistent memory. In *Proc. the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, October 2009, pp.133-146. DOI: 10.1145/1629575.1629589.

[20] Dulloor S R, Kumar S, Keshavamurthy A, Lantz P, Reddy D, Sankaran R, Jackson J. System software for persistent memory. In *Proc. the 9th European Conference on Computer Systems*, April 2014, Article No. 15. DOI: 10.1145/2592798.2592814.

[21] Agarwal N, Wenisch T F. Thermostat: Application-transparent page management for two-tiered main memory. In *Proc. the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, April 2017, pp.631-644. DOI: 10.1145/3093315.3037706.

[22] Coburn J, Caulfield A M, Akel A, Grupp L M, Gupta R K, Jhala R, Swanson S. NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *ACM SIGARCH Computer Architecture News*, 2011, 39(1): 105-118. DOI: 10.1145/1961295.1950380.

[23] Volos H, Tack A J, Swift M M. Mnemosyne: Lightweight persistent memory. *ACM SIGPLAN Not.*, 2011, 47(4): 91-104. DOI: 10.1145/1961296.1950379.

[24] Chakrabarti D R, Boehm H J, Bhandari K. Atlas: Leveraging locks for non-volatile memory consistency. *ACM SIGPLAN Notices*, 2014, 49(10): 433-452. DOI: 10.1145/2714064.2660224.

[25] Yang J, Kim J, Hoseinzadeh M, Izraelevitz J, Swanson S. An empirical guide to the behavior and use of scalable persistent memory. In *Proc. the 18th USENIX Conference on File and Storage Technologies*, February 2020, pp.169-182.

[26] Poremba M, Zhang T, Xie Y. NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. *IEEE Computer Architecture Letters*, 2015, 14(2): 140-143. DOI: 10.1109/LCA.2015.2402435.

[27] Sánchez D, Kozyrakis C. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In *Proc. the 40th Annual International Symposium on Computer Architecture*, June 2013, pp.475-486. DOI: 10.1145/2485922.2485963.

[28] Duan Z, Liu H, Liao X, Jin H. HME: A lightweight emulator for hybrid memory. In *Proc. the Design, Automation Test in Europe Conference and Exhibition*, March 2018, pp.1375-1380. DOI: 10.23919/DATE.2018.8342227.

[29] Volos H, Magalhaes G, Cherkasova L, Li J. Quartz: A lightweight performance emulator for persistent memory software. In *Proc. the 16th Annual Middleware Conference*, November 2015, pp.37-49. DOI: 10.1145/2814576.2814806.

[30] Zhu G, Lu K, Wang X, Zhou X, Shi Z. Building emulation framework for non-volatile memory. *IEEE Access*, 2017, 5: 21574-21584. DOI: 10.1109/ACCESS.2017.2715346.

[31] Zhang W, Li T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proc. the 18th International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2009, pp.101-112. DOI: 10.1109/PACT.2009.30.

[32] Park H, Yoo S, Lee S. Power management of hybrid DRAM/PRAM-based main memory. In *Proc. the 48th ACM/EDAC/IEEE Design Automation Conference*, June 2011, pp.59-64. DOI: 10.1145/2024724.2024738.

[33] Lee S, Bahn H, Noh S H. CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures. *IEEE Transactions on Computers*, 2014, 63(9): 2187-2200. DOI: 10.1109/TC.2013.98.

[34] Salkhordeh R, Asadi H. An operating system level data migration scheme in hybrid DRAM-NVM memory architecture. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, March 2016, pp.936-941. DOI: 10.3850/9783981537079_0605.

[35] Khouzani H A, Hosseini F S, Yang C. Segment and conflict aware page allocation and migration in DRAM-PCM hybrid main memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016, 36(9): 1458-1470. DOI: 10.1109/TCAD.2016.2615845.

[36] Chen D, Jin H, Liao X, Liu H, Guo R, Liu D. MALRU: Miss-penalty aware LRU-based cache replacement for hybrid memory systems. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, March 2017, pp.1086-1091. DOI: 10.23919/DATE.2017.7927151.

[37] Jin H, Chen D, Liu H, Liao X, Guo R, Zhang Y. Miss penalty aware cache replacement for hybrid memory systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020, 39(12): 4669-4682. DOI: 10.1109/TCAD.2020.2966482.

[38] Mladenov R. An efficient non-volatile main memory using phase change memory. In *Proc. the 13th International Conference on Computer Systems and Technologies*, June 2012, pp.45-51. DOI: 10.1145/2383276.2383284.

[39] Loh G H, Hill M D. Efficiently enabling conventional block sizes for very large die-stacked DRAM caches. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2011, pp.454-464. DOI: 10.1145/2155620.2155673.

[40] Meza J, Chang J, Yoon H, Mutlu O, Ranganathan P. Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. *IEEE Computer Architecture Letters*, 2012, 11(2): 61-64. DOI: 10.1109/L-CA.2012.2.

[41] Wang X, Liu H, Liao X, Chen J, Jin H, Zhang Y, Zheng L, He B, Jiang S. Supporting super-pages and lightweight page migration in hybrid memory systems. *ACM Transactions on Architecture and Code Optimization*, 2019, 16(2): 1-26. DOI: 10.1145/3310133.

[42] Wang X, Liu H, Liao X, Jin H, Zhang Y. TLB coalescing for multi-grained page migration in hybrid memory systems. *IEEE Access*, 2020, 8: 66304-66314. DOI: 10.1109/ACCESS.2020.2983065.

[43] Park Y, Park S K, Park K H. Linux kernel support to exploit phase change memory. In *Proc. the Linux Symposium*, July 2010, pp.217-224.

[44] Wei W, Jiang D, McKee S A, Xiong J, Chen M. Exploiting program semantics to place data in hybrid memory. In *Proc. the International Conference on Parallel Architecture and Compilation*, October 2015, pp.163-173. DOI: 10.1109/PACT.2015.10.

[45] Sha E H M, Chen X Z, Zhuge Q, Shi L, Jiang W. Designing an efficient persistent in-memory file system. In *Proc. the IEEE Non-Volatile Memory System and Applications Symposium*, Aug. 2015. DOI: 10.1109/NVMSA.2015.7304365.

[46] Chen X, Sha E H M, Wang X, Yang C, Jiang W, Zhuge Q. Contour: A process variation aware wear-leveling mechanism for inodes of persistent memory file systems. *IEEE Transactions on Computers*. DOI: 10.1109/TC.2020.3002537.

[47] Xue D, Huang L, Li C, Wu C. Dapper: An adaptive manager for large-capacity persistent memory. *IEEE Transactions on Computers*, 2019, 68(7): 1019-1034. DOI: 10.1109/MC.2020.2964916.

[48] Xu J, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *Proc. the 14th U SENIX Conference on File and Storage Technologies*, February 2016, pp.323-338.

[49] Yang J, Izraelevitz J, Swanson S. Orion: A distributed file system for non-volatile main memories and RDMA-capable networks. In *Proc. the 17th USENIX Conference on File and Storage Technologies*, February 2019, pp.221-234.

[50] Dong M, Bu H, Yi J, Dong B, Chen H. Performance and protection in the ZoFS user-space NVM file system. In *Proc. the 27th ACM Symposium on Operating Systems Principles*, October 2019, pp.478-493. DOI: 10.1145/3341301.3359637.

[51] Coburn J, Caulfield A M, Akel A, Grupp L M, Gupta R K, Jhala R, Swanson S. NV-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proc. the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2011, pp.105-118. DOI: 10.1145/1950365.1950380.

[52] Liu R S, Shen D Y, Yang C L, Yu S C, Wang C Y M. NVM Duet: Unified working memory and persistent store architecture. In *Proc. the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2014, pp.455-470. DOI: 10.1145/2541940.2541957.

[53] Denny J E, Lee S, Vetter J S. NVL-C: Static analysis techniques for efficient, correct programming of non-volatile main memory systems. In *Proc. the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, May 2016, pp.125-136. DOI: 10.1145/2907294.2907303.

[54] Zhang L, Swanson S. Pangolin: A fault-tolerant persistent memory programming library. In *Proc. the USENIX Annual Technical Conference*, July 2019, pp.897-912.

[55] Krishnan R M, Kim J, Mathew A, Fu X, Demeri A, Min C, Kannan S. Durable transactional memory can scale with timestone. In *Proc. the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2020, pp.335-349. DOI: 10.1145/3373376.3378483.

[56] Gu J, Yu Q, Wang X, Wang Z, Zang B, Guan H, Chen H. Pisces: A scalable and efficient persistent transactional memory. In *Proc. the USENIX Annual Technical Conference*, July 2019, pp.913-928.

[57] Wu M, Zhao Z, Li H, Li H, Chen H, Zang B, Guan H. Espresso: Brewing Java for more non-volatility with non-volatile memory. In *Proc. the 33rd International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2018, pp.70-83. DOI: 10.1145/3173162.3173201.

[58] Seok H, Park Y, Park K W, Park K H. Efficient page caching algorithm with prediction and migration for a hybrid main memory. *ACM SIGAPP Applied Computing Review*, 2011, 11(4): 38-48. DOI: 10.1145/2107756.2107760.

[59] Hirofuchi T, Takano R. RAMinate: Hypervisor-based virtualization for hybrid main memory systems. In *Proc. the 7th ACM Symposium on Cloud Computing*, October 2016, pp.112-125. DOI: 10.1145/2987550.2987570.

[60] Duan Z, Liu H, Liao X, Jin H, Jiang W, Zhang Y. HiNUMA: NUMA-aware data placement and migration in hybrid memory systems. In *Proc. the 37th IEEE International Conference on Computer Design*, November 2019, pp.367-375. DOI: 10.1109/ICCD46524.2019.00058.

[61] Yang D, Liu H, Jin H, Zhang Y. HMvisor: Dynamic hybrid memory management for virtual machines. *SCIENCE CHINA Information Sciences*. DOI: 10.1007/s11432-019-2729-5. (to be published)

[62] Wang Z, Shan S, Cao T, Gu J, Xu Y, Mu S, Xie Y, Jiménez D A. WADE: Writeback-aware dynamic cache management for NVM-based main memory system. *ACM Transactions on Architecture and Code Optimization*, 2013, 10(4): Article No. 51. DOI: 10.1145/2555289.2555307.

[63] Hu J, Xue C J, Zhuge Q, Tseng W, Sha E H M. Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, March 2011, pp.746-751. DOI: 10.1109/DATE.2011.5763127.

[64] Cho S, Lee H. Flip-*N*-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance. In *Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, December 2009, pp.347-357. DOI: 10.1145/1669112.1669157.

[65] Li Y, Li X, Ju L, Jia Z. A three-stage-write scheme with flip-bit for PCM main memory. In *Proc. the 20th Asia and South Pacific Design Automation Conference*, January 2015, pp.328-333. DOI: 10.1109/ASPDAC.2015.7059026.

[66] Li Z, Wang F, Feng D, Hua Y, Tong W, Liu J, Liu X. Tetris write: Exploring more write parallelism considering PCM asymmetries. In *Proc. the 45th International Conference on Parallel Processing*, August 2016, pp.159-168. DOI: 10.1109/ICPP.2016.25.

[67] Palangappa P M, Mohanram K. CompEx: Compression-Expansion coding for energy, latency, and lifetime improvements in MLC/TLC NVM. In *Proc. the 2016 IEEE International Symposium on High Performance Computer Architecture*, March 2016, pp.90-101. DOI: 10.1109/HPCA.2016.7446056.

[68] Chen Y T, Cong J, Huang H, Liu B, Liu C, Potkonjak M, Reinman G. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, March 2012, pp.45-50. DOI: 10.1109/DATE.2012.6176431.

[69] Liu J, Jaiyen B, Veras R, Mutlu O. RAIDR: Retention-aware intelligent DRAM refresh. In *Proc. the 39th Annual International Symposium on Computer Architecture*, June 2012, pp.1-12. DOI: 10.1109/ISCA.2012.6237001.

[70] Lee D, Kim Y, Seshadri V, Liu J, Subramanian L, Mutlu O. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *Proc. the 19th IEEE International Symposium on High Performance Computer Architecture*, Feb. 2013, pp.615-626. DOI: 10.1109/HPCA.2013.6522354.

[71] David H, Fallin C, Gorbatov E, Hanebutte U R, Mutlu O. Memory power management via dynamic voltage/frequency scaling. In *Proc. the 8th ACM International Conference on Autonomic Computing*, June 2011, pp.31-40. DOI: 10.1145/1998582.1998590.

[72] Pourshirazi B, Zhu Z. Refree: A refresh-free hybrid DRAM/PCM main memory system. In *Proc. the IEEE International Parallel and Distributed Processing Symposium*, May 2016, pp.566-575. DOI: 10.1109/IPDPS.2016.58.

[73] Hay A, Strauss K, Sherwood T, Loh G H, Burger D. Preventing PCM banks from seizing too much power. In *Proc. the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2011, pp.186-195.

[74] Zhao J, Li S, Yoon D H, Xie Y, Jouppi N P. Kiln: Closing the performance gap between systems with and without persistence support. In *Proc. the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2013, pp.421-432. DOI: 10.1145/2540708.2540744.

[75] Awad A, Blagodurov S, Solihin Y. Write-aware management of NVM-based memory extensions. In *Proc. the International Conference on Supercomputing*, June 2016, Article No. 9. DOI: 10.1145/2925426.2926284.

[76] Zhang L, Neely B, Franklin D, Strukov D, Xie Y, Chong F T. Mellow writes: Extending lifetime in resistive memories through selective slow write backs. In *Proc. the 43rd ACM/IEEE Annual International Symposium on Computer Architecture*, June 2016, pp.519-531. DOI: 10.1109/ISCA.2016.52.

[77] Qureshi M K, Karidis J, Franceschini M, Srinivasan V, Lastras L, Abali B. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proc. the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2009, pp.14-23. DOI: 10.1145/1669112.1669117.

[78] Azevedo R, Davis J D, Strauss K, Gopalan P, Manasse M, Yekhanin S. Zombie memory: Extending memory lifetime by reviving dead blocks. In *Proc. the 40th Annual International Symposium on Computer Architecture*, June 2013, pp.452-463. DOI: 10.1145/2485922.2485961.

[79] Ipek E, Condit J, Nightingale E B, Burger D, Moscibroda T. Dynamically replicated memory: Building reliable systems from nanoscale resistive memories. *ACM S IGPLAN Notices*, 2010, 45(3): 3-14. DOI: 10.1145/1735970.1736023.

[80] Jin H, Li Z, Liu H, Liao X, Zhang Y. Hotspot-aware hybrid memory management for in-memory key-value stores. *IEEE Transactions on Parallel and Distributed Systems*, 2020, 31(4): 779-792. DOI: 10.1109/TPDS.2019.2945315.

[81] Zhou J, Shen Y, Li S, Huang L. NVHT: An efficient key-value storage library for non-volatile memory. In *Proc. the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, December 2016, pp.227-236. DOI: 10.1145/3006299.3006318.

[82] Li W, Jiang D, Xiong J, Bao Y. HiLSM: An LSM-based key-value store for hybrid NVM-SSD storage systems. In *Proc. the 17th ACM International Conference on Computing Frontiers*, May 2020, pp.208-216. DOI: 10.1145/3387902.3392621.

[83] Eisenman A, Gardner D, AbdelRahman I, Axboe J, Dong S, Hazelwood K, Petersen C, Cidon A, Katti S. Reducing DRAM footprint with NVM in Facebook. In *Proc. the 13th EuroSys Conference*, April 2018, Article No. 42. DOI: 10.1145/3190508.3190524.

[84] Kim J, Lee S, Vetter J S. PapyrusKV: A high-performance parallel key-value store for distributed NVM architectures. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2017, Article No. 57. DOI: 10.1145/3126908.3126943.

[85] Arulraj J. Data management on non-volatile memory. In *Proc. the 2019 International Conference on Management of Data*, June 2019, p.1114. DOI: 10.1145/3299869.3328523.

[86] Lersch L, Hao X, Oukid I, Wang T, Willhalm T. Evaluating persistent memory range indexes. *Proc. VLDB Endowment*, 2019, 13(4): 574-587. DOI: 10.14778/3372716.3372728.

[87] Lu B, Hao X, Wang T, Lo E. Dash: Scalable hashing on persistent memory. *Proc. VLDB Endow.*, 2020, 13(10): 1147-1161. DOI: 10.14778/3389133.3389134.

[88] Mahapatra P, Hill M D, Swift M M. Don't persist all: Efficient persistent data structures. arXiv:1905.13011, 2019. https://arxiv.org/abs/1905.13011, Oct. 2020.

[89] Ni Y, Chen S, Lu Q, Litz H, Zhu P, Miller E L, Wu J. Closing the performance gap between DRAM and PM for in-memory index structures. Technical Report, University of California, 2020. https://www.ssrc.ucsc.edu/media/pubs/329b041d657e2c2225aa68fb33e72ecca157e6df.pdf, Oct. 2020.

[90] Chen Y, Lu Y, Yang F, Wang Q, Wang Y, Shu J. FlatStore: An efficient log-structured key-value storage engine for persistent memory. In *Proc. the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2020, pp.1077-1091. DOI: 10.1145/3373376.3378515.

[91] Liu W, Liu H, Liao X, Jin H, Zhang Y. NGraph: Parallel graph processing in hybrid memory systems. *IEEE Access*, 2019, 7: 103517-103529. DOI: 10.1109/ACCESS.2019.2931058.

[92] Malicevic J, Dulloor S, Sundaram N, Satish N, Jackson J, Zwaenepoel W. Exploiting NVM in large-scale graph analytics. In *Proc. the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*, October 2015, Article No. 2. DOI: 10.1145/2819001.2819005.

[93] Gill G, Dathathri R, Hoang L, Peri R, Pingali K. Single machine graph analytics on massive datasets using Intel Optane DC persistent memory. *Proc. VLDB Endow.*, 2020, 13(10): 1304-1318. DOI: 10.14778/3389133.3389145.

[94] Long Y, She X, Mukhopadhyay S. Design of reliable DNN accelerator with un-reliable ReRAM. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, March 2019, pp.1769-1774. DOI: 10.23919/DATE.2019.8715178.

[95] Li S, Xu C, Zou Q, Zhao J, Lu Y, Xie Y. Pinatubo: A processing-in-memory architecture for bulk bit-wise operations in emerging non-volatile memories. In *Proc. the 53rd Annual Design Automation Conference*, June 2016. DOI: 10.1145/2897937.2898064.

[96] Han L, Shen Z, Liu D, Shao Z, Huang H H, Li T. A novel ReRAM-based processing-in-memory architecture for graph traversal. *ACM Transactions on Storage*, 2018, 14(1): Article No. 9. DOI: 10.1145/3177916.

[97] Zilberberg O, Weiss S, Toledo S. Phase-change memory: An architectural perspective. *ACM Computing Surveys*, 2013, 45(3): Article No. 29. DOI: 10.1145/2480741.2480746.

[98] Wu C, Zhang G, Li K. Rethinking computer architectures and software systems for phase-change memory. *ACM Journal on Emerging Technologies in Computing Systems*, 2016, 12(4): Article No. 33. DOI: 10.1145/2893186.

[99] Boukhobza J, Rubini S, Chen R, Shao Z. Emerging NVM: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems*, 2017, 23(2): Article No. 14. DOI: 10.1145/3131848.

[100] Mittal S, Vetter J S. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 2016, 27(5): 1537-1550. DOI: 10.1109/TPDS.2015.2442980.

[101] Li Y, Ghose S, Choi J, Sun J, Wang H, Mutlu O. Utility-based hybrid memory management. In *Proc. the IEEE International Conference on Cluster Computing*, Sept. 2017, pp.152-165. DOI: 10.1109/CLUSTER.2017.130.

[102] Gandhi J, Basu A, Hill M D, Swift M M. BadgerTrap: A tool to instrument x86-64 TLB misses. *ACM SIGARCH Computer Architecture News*, 2014, 42(2): 20-23. DOI: 10.1145/2669594.2669599.

[103] Chen C, Hsiu P, Kuo T, Yang C, Wang C. Age-based PCM wear leveling with nearly zero search cost. In *Proc. the 49th Annual Design Automation Conference*, June 2012, pp.453-458. DOI: 10.1145/2228360.2228439.

[104] Gao S, He B, Xu J. Real-time in-memory checkpointing for future hybrid memory systems. In *Proc. the 29th ACM on International Conference on Supercomputing*, June 2015, pp.263-272. DOI: 10.1145/2751205.2751212.

[105] Islam N S, Rahman M, Lu X, Panda D K. High performance design for HDFS with byte-addressability of NVM and RDMA. In *Proc. the International Conference on Supercomputing*, June 2016, Article No. 8. DOI: 10.1145/2925426.2926290.

[106] Knyaginin D, Gaydadjiev G N, Stenström P. Crystal: A design-time resource partitioning method for hybrid main memory. In *Proc. the 43rd International Conference on Parallel Processing*, September 2014, pp.90-100. DOI: 10.1109/ICPP.2014.18.

[107] Yang J, Wei Q, Wang C, Chen C, Yong K, He B. NV-tree: A consistent and workload-adaptive tree structure for non-volatile memory. *IEEE Transactions on Computers*, 2016, 65(7): 2169-2183. DOI: 10.1109/TC.2015.2479621.

[108] Arulraj J, Levandoski J, Minhas U, Larson P. Bztree: A high-performance latch-free range index for non-volatile memory. *Proc. VLDB Endow.*, 2018, 11(5): 553-565. DOI: 10.1145/3164135.3164147.

[109] Ma K, Zheng Y, Li S, Swaminathan K, Li X, Liu Y, Sampson J, Xie Y, Narayanan V. Architecture exploration for ambient energy harvesting nonvolatile processors. In *Proc. the 21st IEEE International Symposium on High Performance Computer Architecture*, Feb. 2015, pp.526-537. DOI: 10.1109/HPCA.2015.7056060.

[110] Zhu T, Hu H, Qian W, Zhou H, Zhou A. Fault-tolerant precise data access on distributed log-structured merge-tree. *Frontiers of Computer Science*, 2019, 13(4): 760-777. DOI: 10.1007/s11704-018-7198-6.

[111] Doshi K, Giles E, Varman P. Atomic persistence for SCM with a non-intrusive backend controller. In *Proc. the IEEE International Symposium on High Performance Computer Architecture*, March 2016, pp.77-89. DOI: 10.1109/HPCA.2016.7446055.

[112] Olson M A, Bostic K, Seltzer M I. Berkeley DB. In *Proc. the USENIX Annual Technical Conference*, June 1999, pp.183-191.

[113] Sears R, Brewer E. Stasis: Flexible transactional storage. In *Proc. the 7th Symposium on Operating Systems Design and Implementation*, November 2006, pp.29-44.

[114] Izraelevitz J, Yang J, Zhang L *et al.* Basic performance measurements of the Intel Optane DC persistent memory module. arXiv:1903.05714, 2019. https://arxiv.org/abs/1903.05714, Oct. 2020.

[115] Patil O, Ionkov L, Lee J, Mueller F, Lang M. Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using Intel Optane DC persistent memory modules. In *Proc. the International Symposium on Memory Systems*, September 2019, pp.288-303. DOI: 10.1145/3357526.3357541.

[116] Weiland M, Brunst H, Quintino T *et al.* An early evaluation of Intel's Optane DC persistent memory module and its impact on high-performance scientific applications. In *Proc. the International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2019, Article No. 76. DOI: 10.1145/3295500.3356159.

[117] Peng I B, Gokhale M B, Green E W. System evaluation of the Intel Optane byte-addressable NVM. In *Proc. the International Symposium on Memory Systems*, September 2019, pp.304-315. DOI: 10.1145/3357526.3357568.

[118] Zhou Y, Philbin J, Li K. The multi-queue replacement algorithm for second level buffer caches. In *Proc. the 2001 USENIX Annual Technical Conference*, June 2001, pp.91-104.

[119] Liu H, Ye Y, Liao X, Jin H, Zhang Y, Jiang W, He B. Space-oblivious compression and wear leveling for non-volatile main memories. In *Proc. the 36th International Conference on Massive Storage Systems and Technology*, October 2020.

[120] Liu H, Liu R, Liao X, Jin H, He B, Zhang Y. Object-level memory allocation and migration in hybrid memory systems. *IEEE Transactions on Computers*, 2020, 69(9): 1401-1413. DOI: 10.1109/TC.2020.2973134.

[121] Hassan A, Vandierendonck H, Nikolopoulos D S. Software-managed energy-efficient hybrid DRAM/NVM main memory. In *Proc. the 12th ACM International Conference on Computing Frontiers*, May 2015, Article No. 23. DOI: 10.1145/2742854.2742886.

[122] Hu X, Wang X, Li Y, Zhou L, Luo Y, Ding C, Jiang S, Wang Z. LAMA: Optimized locality-aware memory allocation for key-value cache. In *Proc. the USENIX Annual Technical Conference*, July 2015, pp.57-69.

[123] Shun J, Blelloch G E. Ligra: A lightweight graph processing framework for shared memory. *ACM SIGPLAN Not.*, 2013, 48(8): 135-146. DOI: 10.1145/2442516.2442530.

[124] Yu S, Chen P. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Magazine*, 2016, 8(2): 43-56. DOI: 10.1109/MSSC.2016.2546199.

[125] Lu Y, Shu J, Chen Y, Li T. Octopus: An RDMA-enabled distributed persistent memory file system. In *Proc. the USENIX Annual Technical Conference*, July 2017, pp.773-785.

[126] Shan Y, Tsai S, Zhang Y. Distributed shared persistent memory. In *Proc. the 2017 Symposium on Cloud Computing*, September 2017, pp.323-337. DOI: 10.1145/3127479.3128610.

[127] Chen T, Liu H, Liao X, Ji H. Resource abstraction and data placement for distributed hybrid memory pool. *Frontiers of Computer Science*, 2020. (accepted)

[128] Barbalace A, Iliopoulos A, Rauchfuss H, Brasche G. It's time to think about an operating system for near data processing architectures. In *Proc. the 16th Workshop on Hot Topics in Operating Systems*, May 2017, pp.56-61. DOI: 10.1145/3102980.3102990.

[129] Gu B, Yoon A S, Bae D *et al.* Biscuit: A framework for near-data processing of big data workloads. In *Proc. the 43rd International Symposium on Computer Architecture*, June 2016, pp.153-165. DOI: 10.1109/ISCA.2016.23.

[130] Yan H, Ahn E C, Duan L. Enabling NVM-based deep learning acceleration using nonuniform data quantization: Work-in-progress. In *Proc. the 2017 International Conference on Compilers, Architectures and Synthesis for Embedded Systems Companion*, October 2017, Article No. 20. DOI: 10.1145/3125501.3125516.

[131] Chen A. Utilizing the variability of resistive random access memory to implement reconfigurable physical unclonable functions. *IEEE Electron Device Letters*, 2015, 36(2): 138-140. DOI: 10.1109/LED.2014.2385870.

[132] Torres L, Brum R, Cargnini L, Sassatelli G. Trends on the application of emerging non-volatile memory to processors and programmable devices. In *Proc. the IEEE International Symposium on Circuits and Systems*, May 2013, pp.101-104. DOI: 10.1109/ISCAS.2013.6571792.

[133] Liauw Y, Zhang Z, Kim W, Gamal A E, Wong S S. Nonvolatile 3D-FPGA with monolithically stacked RRAM-based configuration memory. In *Proc. the 2012 IEEE International Solid-State Circuits Conference*, Feb. 2012, pp.406-408. DOI: 10.1109/ISSCC.2012.6177067.

[134] Zhao W, Belhaire E, Chappert C, Mazoyer P. Spin transfer torque (STT)-MRAM-based runtime reconfiguration FPGA circuit. *ACM Trans. Embed. Comput. Syst.*, 2009, 9(2): Article No. 14. DOI: 10.1145/1596543.1596548.

[135] Chen Y, Wong W, Li H, Koh C, Zhang Y, Wen W. On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations. *ACM Journal on Emerging Technologies in Computing Systems*, 2013, 9(2): Article No. 16. DOI: 10.1145/2463585.2463592.

Hai-Kun Liu received his Ph.D. degree in computer architecture in Huazhong University of Science and Technology (HUST), Wuhan, in 2012. He was the recipient of Outstanding Doctoral Dissertation Award in Hubei, China. He is an associate professor in School of Computer Science and Technology, HUST, Wuhan. His current research interests include in-memory computing, virtualization technologies, cloud computing, and distributed systems. He is a senior member of CCF and a member of ACM and IEEE.

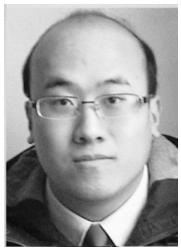
Di Chen received his B.E. degree in computer science and technology in Huazhong University of Science and Technology (HUST), Wuhan, 2013. He is currently working towards his Ph.D. degree in HUST, Wuhan. His current research interest is hybrid memory systems.

Hai Jin is a Cheung Kung Scholars Chair Professor of computer science and engineering at Huazhong University of Science and Technology (HUST) in Wuhan. Jin received his Ph.D. degree in computer engineering from HUST, Wuhan, in 1994. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. Jin worked at The University of Hong Kong, Hong Kong, between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. Jin is the chief scientist of ChinaGrid, the largest grid computing project in China, and the chief scientist of National 973 Basic Research Program Project of Virtualization Technology of Computing System, and Cloud Security. Jin is a fellow of CCF, a fellow of the IEEE and a member of ACM. He has co-authored 22 books and published over 800 research papers. His research interests include computer architecture, cloud computing, peer-to-peer computing, and network security.

Xiao-Fei Liao received his Ph.D. degree in computer science and engineering from Huazhong University of Science and Technology (HUST), Wuhan, in 2005. He is now a professor in the School of Computer Science and Engineering, HUST, Wuhan. His research interests are in the areas of system virtualization, system software, and cloud computing.

Bingsheng He received his Bachelor's degree in computer science from Shanghai Jiao Tong University, Shanghai, and his Ph.D. degree in computer science in Hong Kong University of Science and Technology, Hong Kong. He is an associate professor in School of Computing, National University of Singapore, Singapore. His research interests are high-performance computing, cloud computing, and database systems. He has been awarded with the IBM Ph.D. fellowship (2007–2008) and with NVIDIA Academic Partnership (2010–2011).

Kan Hu received his Ph.D. degree in computer architecture from Huazhong University of Science and Technology (HUST), Wuhan, in 2007. He is an associate professor in School of Computer Science and Technology, HUST, Wuhan. His research interests are in the areas of system virtualization and distributed computing.



Yu Zhang received his Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, in 2016. He is now an associate professor in School of Computer Science and Technology at HUST, Wuhan. His research interests include computer architecture, system software, runtime optimization, programming model, and graph processing. He is a member of CCF, ACM, IEEE, and USENIX.