RZ 3809 (# Z1109-001) 09/15/2011
Computer Science 8 pages

Research Report

Subsystem and System-level Implications of PCM

R. Haas, X.-Y. Hu, I. Koltsidas, R.A. Pletka

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Almaden • Austin • Brazil • Cambridge • China • Haifa • India • Tokyo • Watson • Zurich

Subsystem and System-level Implications of PCM

Robert Haas, Xiao-Yu Hu, Ioannis Koltsidas, Roman A. Pletka
IBM Research – Zurich, 8803 Rüschlikon, Switzerland, {rha,xhu,iko,rap}@zurich.ibm.com

ABSTRACT

Emerging data-centric workloads necessitate systems with highly scalable storage components, not only in terms of capacity, but also in terms of performance and energy efficiency. These requirements, in combination with advances in Storage Class Memory (SCM) technologies, will drive the departure from traditional memory and storage architectures. The focus of this paper is on systems that employ Phase-Change Memory (PCM) to either complement or replace traditional media such, as HDDs, protected DRAM, and Flash. We study possible uses of PCM across the entire stack, and present an analysis of the implications this has for the various components, such as the main memory subsystem, storage subsystems, operating systems, filesystems and databases, as well as of its impact on the overall system architectures. Key problems introduced by the new medium are identified and solutions are outlined, which, we believe, will be very valuable for the designers of next-generation data-centric systems and applications.

1. Introduction

Newer workloads pose ever higher performance and/or capacity demands on main memory and storage, creating technology-scaling challenges and new possibilities for disruptions [1]. Whereas the main-memory Dynamic Random Access Memory (DRAM) technology may exhibit difficulties to scale down below 30 nm, the Hard Disk Drive (HDD) technology not only exhibits a substantial slow-down in areal-density growth, but also a continuing decrease in the number of IOPS (I/O operations per second) normalized to the storage capacity per platter or per device. Recently, NAND-Flash-based Solid-State Drives (SSDs) started to fill in the widening performance gap between processors and storage systems. However, scaling also has a detrimental impact on Flash, particularly with the more aggressive multi-level cell technology. Workloads that require fast access over increasingly large datasets therefore necessitate alternatives to these traditional technologies, with inevitable architectural implications at not only the memory and storage subsystem but also at the software and system levels.

In this paper, we provide a holistic analysis of the implications the introduction of Phase-Change Memory (PCM) has for subsystems and at the system level. Section 2 looks at the issues arising when PCM replaces or is combined with existing DRAM main memory, and Section 3 focuses on the implications of persistency in this context. Then, Section 4 delves into aspects of the storage subsystem level. Section 5 covers the software-level aspects in a broader sense, and specifically discusses how PCM will affect filesystem and database architectures. Finally, the system-level implications are addressed in Section 6, and conclusions are presented in Section 7.

2. Traditional main-memory usage

In terms of main memory, PCM, given its higher density and energy efficiency, has the potential to complement or replace DRAM provided that suitable solutions address its finite endurance, read/write latency penalty and asymmetry. A number of techniques have been developed to achieve write reduction, and hence increase the lifetime of PCM. Starting at the PCM circuit level itself, the shorter read latency can be exploited to eliminate redundant bit writes by first comparing the current value against the preceding bit value. Also, wear leveling can mitigate the address locality of workloads and help wear out cells evenly: wear-leveling techniques can be implemented in the PCM circuit itself to perform fine-granularity shifting within a line in a page, or at the memory controller to periodically swap larger memory segments containing hot and cold memory pages [2], or a moving gap can be used [3]. Combinations of such techniques have been shown to increase the lifetime by almost two orders of magnitude

using a typical benchmarking program directly accessing PCM as main memory and assuming that the technology can sustain 10^8 rewrite cycles [2].

In addition, the main memory can be made hybrid by placing a DRAM buffer in front of PCM to reduce the number of writes to PCM. The memory controller uses the DRAM buffer as a hardware cache to perform intelligent write scheduling to PCM as well as to write only the data in a cached page that has been modified. By combining such techniques the lifetime of hybrid architectures could be extended by a factor of three with only a small hardware overhead [4].

All the above write-reduction schemes also contribute to reducing access contention due to the read/write asymmetry of PCM. However, because the write latency is typically four times higher than the read latency, an incoming read request may still be unacceptably delayed by an on-going write operation to the same PCM memory bank. This is addressed by the preemption of write operations to expedite reads, as is done by adaptive write cancellation and Multi-Level Cell (MLC) write pausing schemes [5]. The drawback of those schemes is that they increase the complexity of the memory controller because of the large queue of write requests.

Given the widening speed gap between CPU and DRAM, PCM should not aggravate performance issues even if it enables density scaling. Hybrid schemes that can hide the performance penalty of PCM are of course one solution, but simulations have also shown that the use of narrower buffers in PCM and other architectural optimizations can lead to similar performance levels with Single-Level Cell (SLC) PCM as those of DRAM in terms of application delay [6]. Given that MLC PCM significantly increases the read and write latency while boosting capacity, shifting the operation mode of cells between single-level and multi-level operation based on the actual memory capacity demand at runtime can achieve a dynamic trade-off between latency and density, but requires both hardware and OS support to dynamically adapt the memory capacity [7].

3. Persistent main-memory usage

An even more disruptive characteristic than the density scaling is the *persistency* of PCM. High-performance access to persistent data can not only accelerate systems significantly (e.g., boot process, hibernation, and writing of application data and file-system check-pointing), but also render them more reliable. Enterprise systems already protect some of their DRAM memory content against power failures with *protected memory* based on battery-backed, Flash-backed, or disk-backed mechanisms. One approach is to let applications access a protected memory region through specific APIs to store their data structures [8],[9]. In case of power failure, applications reconstruct their state from those protected data structures, whereas the content in the unprotected region is lost. In a more comprehensive approach, not only application data, but the entire OS is located in protected memory so that upon a normal shutdown or a power failure, processor caches must be flushed and additional information (i.e., processor registers) must be written back into protected memory. This process is similar to hibernating using a disk. A special boot procedure then has to restore registers before normal operation can be resumed. However, similarly to a corrupted filesystem, where special off-line tools are required to regain a consistent state, starting such a system from its protected main memory cannot fix potential memory leaks, invalid pointers, or locking violations. In these cases, the protected main-memory content must be discarded, and a reboot from disk must be done, resulting in a potential data loss. Similarly, with a PCM-only main memory, the discard operation would require resetting some regions of the PCM. The advantage of a PCM-based main memory is that, in contrast to traditional protected memory mechanisms, which only scale with the DRAM size, it does not have this limitation.

A drawback of a persistent main memory is that typical optimizations of memory accesses from the processor may cause a reordering of a group of writes and thus leave the system in an inconsistent state if the system stops before all writes complete. Hence suitable solutions for updating non-volatile data structures are required to not only guarantee *durability*, i.e., ensuring that all writes reached persistent memory, but also *consistency*, in the sense that a group of write operations is performed atomically and that the ordering between certain writes is preserved. In systems without protected main memory, this has been solved by using synchronous writes or by committing grouped writes directly to a storage device together. Especially ungrouped synchronous writes to HDD are detrimental to system performance; this is not the case on PCM where small writes incur low latency. Therefore, PCM has the capability to accelerate

system performance with synchronous writes independent of whether they are grouped or not. Many concepts that provide consistency exist, e.g., buffer cache in file systems, persistence interfaces such as the Java Persistence API [9] or RVM [8], databases, transactional memory [10], or persistent key value stores. All of them have been optimized to bring data into an appropriate format (e.g., inodes, files, or serialized objects) for disk-based storage systems and to schedule I/Os according to the characteristics of block-based disk storage. For all these concepts, the means to guarantee consistency are log structures or copy-on-write operations [11]. These concepts may also be used with persistent main memory. For instance, fine-grained updates with copy-on-write have been proposed in BPFS [12] at the cost of additional hardware primitives for atomic updates. BPFS shows how an interface built on top of storage devices can be re-implemented on persistent main memory.

Consistency guarantees in main memory have also been addressed using transactional memory (TM) [10]. As committing transactions to persistent storage can be delayed or not done at all, TM only provides persistency as long as a clean shutdown can be guaranteed. A transaction keeps modified data in a log-like dedicated write set until it is committed successfully and changes are made visible. Otherwise, the transaction is aborted by restoring the initial state. The programmer can simply mark the code section to be executed atomically accordingly. The combination of PCM and a TM would augment the system durability ideally. In [13] it was shown that such a combination can be achieved with limited overhead on top of Flash.

Another challenge arises from combining volatile and persistent main memory: References to and from data structures in non-volatile memory have to be handled with care. Any reference in volatile memory to an object in non-volatile memory might cause a memory leak after a power failure. Similarly, data loss can result from references in non-volatile data structures to objects in volatile memory. Even references between two distinct non-volatile memory regions are critical because one of the regions might not be accessible after a failure. Although current compilers do not yet make such checks, libraries to perform such checks at run time exist [14].

Persistency may require the use of encryption of main memory. Storage systems use block cipher modes to encrypt sectors using a single key, and tweakable modes even allow encryption of large numbers of blocks with the same key. Byte-addressable persistent memory conflicts with the fixed sector sizes of those encryption modes: Although the encryption block size is rather small (up to 32 bytes), the block cipher mode chains all blocks in a sector so that an entire sector must be decrypted and re-encrypted even if only a single word has been modified. This can be addressed by a hybrid DRAM-PCM memory in which DRAM is used as a byte-addressable cache in front of page-based PCM.

4. Storage subsystem usage

Existing storage architectures cannot fully benefit from the low-latency and byte-addressability characteristics of PCM. The high latency of the I/O bus and the use of traditional block-based interfaces prevent the execution of reads and writes at word granularity and low latency. Even high-performance I/O buses, such as PCI-e, are not suitable as they are primarily designed for bulk data transfers [12]. As discussed above, exploiting the full benefits of PCM requires that the CPU can access it directly with common load and store commands over the memory bus.

Nevertheless, attaching persistent low-latency memories to the I/O bus is attractive for at least three reasons. Firstly, the latency and endurance characteristics of PCM are orders of magnitude better than those of any other type of non-volatile memory currently in use (with the exception of battery-backed DRAM) and can still be leveraged despite the performance limitations of the I/O bus. Secondly, integrating PCM on the memory bus will require extensive hardware and software changes, with the risk of incurring more subtle performance bottlenecks, such as lock contention in kernel data structures associated with serving I/O requests. Therefore, leveraging PCM in the storage hierarchy is a more immediate path towards supporting increasingly demanding workloads. Thirdly, similarly to HDDs, aggregating pools of PCM storage in network-attached storage controllers has very desirable reliability, availability, serviceability and fault-tolerance benefits, which will drive the adoption of storage-attached PCM in enterprise environments.

Storage architectures increasingly leverage heterogeneous technologies such as Flash, HDDs, and tape to sustain the combined performance and capacity growth, similarly to multi-level and hybrid memory architectures. Given the

significantly higher endurance and performance of PCM as compared to Flash, the shift towards PCM will be determined by the availability of high-density and low-cost chips. Until sufficient such chips are available, storage subsystems can already make use of hybrid devices: Flash memory can be used as the permanent storage for data, whereas PCM is used to store logical-to-physical address mappings and other metadata. For these purposes, the byte addressability of PCM may even be an additional advantage [15]. The PCM portion of the device can also be used to absorb writes, either as a low-latency, update-in-place write cache in front of the Flash memory or for logging small updates to Flash pages, effectively reducing the write amplification in Flash. As a result, the lifetime of the Flash device is extended because most writes are being absorbed in PCM. Of course, this requires careful design to ensure that in such a device PCM will not wear out faster than Flash does.

5. Software-level usage

The I/O software stack of current systems, including operating system I/O schedulers, device drivers, filesystems and database storage engines, has been designed and optimized to accommodate the millisecond-range seek times of HDDs. A performance improvement of almost two orders of magnitude over HDDs can be achieved thanks to the byte addressability of PCM; however, new bottlenecks will be incurred because of inefficiencies and overheads in the current stack that were previously hidden by the high latency of HDDs [16]. For instance, about 20,000 instructions are required to issue and complete a typical 4 kB I/O request in Linux. When using a PCM-based storage device instead of HDDs, this overhead accounts for more than 60% of the total latency per request. A significant amount of the latency can be saved by completely bypassing the I/O scheduler of the OS and by removing certain locking structures in the kernel and re-implementing others as lock-free data structures, so that many threads can serve interrupts in parallel. Of course, a system could completely avoid relying on interrupts for I/O by allowing threads to spin in busy-loop instead of sleeping, thus saving the latency of context switching. Although this technique can reduce latency, it entails a significant increase in CPU utilization, and thus may only be suitable for very small requests [16].

The main mechanisms to organize and store data persistently are filesystems and databases, operating directly on top of the I/O stack. Similarly to hybrid memory and storage subsystems, filesystems can take direct advantage of PCM, possibly in combination with Flash and HDDs, either for their metadata or the data itself.

PCM will be particularly suitable for filesystem metadata for three reasons. Firstly, accesses to filesystem metadata typically follow random-like patterns, and with PCM the latency of random accesses for both reads and writes is lower than with either Flash or HDDs. Secondly, the byte addressability of PCM allows metadata updates at word granularity, resulting in less bandwidth consumption and fewer writes, provided a suitable interface is used instead of the block-based interface (typically employing 512 B or 4 kB blocks). Thirdly, the non-volatility of PCM ensures that modifications to metadata are durable once the CPU cache has been flushed. Thereby, filesystem metadata does not need to occupy precious DRAM buffers, even though parts of it may be cached in-memory for better performance and longer PCM lifetime [17], [12]. In particular, PCM can be used to support metadata for log-structured filesystems tailored specifically for Flash-based devices: such filesystems suffer from a large DRAM memory footprint for metadata and long mounting times, as the entire metadata has to be read from Flash to reconstruct the in-memory metadata image upon mount. By separating the metadata from the data and maintaining the former on PCM, both drawbacks are overcome, making scalable filesystems on hybrid storage possible [17], [18].

PCM is clearly also suitable for filesystem data, especially with short reads and writes, for the same reasons as above: Whereas traditional filesystems employ a large block size and large I/O requests to amortize the cost of accessing the HDD over a large amount of data, a PCM-aware filesystem only needs to issue small requests. Most importantly, by departing from traditional filesystems that rely either on write-ahead logging (journaling) or on copy-on-write (shadow paging) techniques to guarantee consistency, a PCM-based filesystem can achieve consistency without writing metadata out of place. Instead, small, atomic, in-place metadata updates can be used to only touch the modified blocks of a file tree, thereby avoiding the copying of metadata that did not change. Such atomic operations to PCM can be implemented using hardware for 8-byte atomic write support [12].

Instead of offering a filesystem interface to persistent storage, PCM-equipped systems can provide a memory allocation interface to applications, such as a fully persistent heap where persistent objects can be allocated and

manipulated directly on PCM, using system calls similar to malloc() and free() to (de)allocate persistent memory [14]. The advantage of this approach is that it eliminates the overhead of the I/O stack and the filesystem, while allowing applications to maintain their internal data representation persistently without the need to transform it to serialized formats, such as files. The disadvantage, of course, is that current applications would have to be re-written, losing the backwards compatibility ensured by a file-based interface. For instance, an application upgrade would require to clearly distinguish between data structures being utilized in the new version (either unchanged or converted) and those requiring re-initialization or removal. In any case, the exchange of data between applications would anyway require that the applications transform their data into standard formats (like files). Thus, in the future, filesystem interfaces will not disappear, but they very likely will be complemented by persistent memory allocation interfaces similar to those that exist for main memory.

Databases will profit from PCM in many respects: The byte addressability of PCM presents many opportunities for databases to optimize the layout of data on persistent storage. As only those records and attributes are read that are necessary to answer a query, the memory bandwidth is used more efficiently and the processors can be fed with useful data at a much higher rate, resulting in orders-of-magnitude better throughput, as column-oriented storage engines tailored specifically to read-optimized databases (e.g., for data warehousing) show [19]. Of course, database systems will need to identify which objects have to be stored on PCM and which on lower-performance media. For database buffer management, the main memory buffering will also have to be adapted to the cost metrics of PCM. For instance, it would make sense to cache PCM-resident data in DRAM only if such data is bound to be accessed multiple times in the near future [20]. Furthermore, PCM can also be used as a direct extension of the main-memory buffer pool [21], acting as a second-level cache for data that is classified as hot but does not fit into the main-memory buffer pool. In terms of database logging, PCM is particularly suitable as a synchronous database log to relieve the overhead of write-ahead logging, which is one of the most critical bottlenecks in transaction processing. Firstly, the logging architecture can be simplified by eliminating the need for DRAM log buffers used for group commits to hide the disk latency. Secondly, the performance can be improved through better concurrency, i.e., by allowing transactions to write small log records independently of one another, instead of having to synchronize to fill out DRAM log buffers in the proper order for consistency [22]. Finally, database data structures will have to be adapted to the performance and endurance characteristics of PCM. For instance, dropping the requirement of B^+-tree records being sorted on their key and packed can significantly reduce the wear of PCM as well as the energy consumption of the system [23]. The same is true for specifically tailoring hash-join algorithms to PCM [23].

6. System-level implications of PCM

This section first examines the system-level impact of PCM on legacy storage systems, and then looks at novel storage architectures. Traditionally, a large part of the storage capacity is provided by external HDD-based storage systems. Block-level access is provided with a Storage Area Network (SAN), whereas shared access at the file level is provided via Network-Attached Storage (NAS). Such systems employ sophisticated storage controllers with protected or unprotected DRAM caches.

Although HDD storage still scales well in terms of capacity, performance scaling of systems relies on increasing the number of HDDs or increasing the DRAM cache size. This is challenging because of the typical space and power constraints in datacenters. The emerging use of Flash memory as a DRAM cache extension and/or as a top storage tier already mitigates this performance-scaling challenge to some extent. Given the significant performance, scalability and endurance advantages of PCM, PCM will eventually enter external legacy storage systems and complement or replace Flash for caching and tiering purposes. Tiering-aware systems are capable of intelligently placing and dynamically moving data across tiers, with cold data in cost-efficient HDDs and hot data in high-performing PCM and Flash memory.

To improve application performance, a recent trend is to introduce a new layer consisting of a cluster of distributed read-only cache nodes between application servers and database systems. Distributed caches can make efficient use of the main memory available at each node, as for instance with memcached [24]. Such distributed caches offer object caching based on simple in-memory key-value data stores to speed up interactive web applications by reducing database load. Owing to the clustered architecture, scaling is not limited by the DRAM density, but by the power

consumption. PCM will enable the total size of such caches to be increased, thus further reducing the load on the back-end systems and improving the energy efficiency. In addition, PCM persistency may be exploited to turn such caches into persistent data stores, e.g., by using memcacheDB [25].

To further improve application latency, a new storage tier that extends the traditional SAN-based hierarchical storage directly into the servers is emerging. This new tier consists of distributed Direct Attached Storage (DAS) clustered in a "share-nothing" architecture, i.e., one in which the nodes do not share the directly attached storage; rather, locality properties are exploited to split data across nodes. Such systems have already been introduced using Flash-based DAS [26]. The use of PCM will enable an even better integration of this new storage tier than is possible with Flash.

Beyond the techniques described above, novel approaches are being pursued that aim at achieving a much higher level of scalability, performance, and energy efficiency than can be attained by systems that still eventually rely on HDDs. Motivated by the observation that the total amount of RAM used by popular social networking sites equals about 75% of the total size of their data, RAMCloud uses DRAM as primary storage by aggregating the main memories of thousands of commodity servers and relies on replication and backup to provide data reliability [27]. This results in an orders of magnitude improvement in terms of both I/O throughput and access latency, albeit with a comparable increase in power consumption. Other approaches use Flash memory for primary storage and thus have to deal with the intricacies of Flash management, such as out-of-place writes and wear. To this end, a new Flash translation layer was introduced in [28], whereas a log-like data store is used in [29]. However, both solutions complicate the overall system design. In all of these three examples, the transition to PCM for primary storage would substantially reduce the power consumption, increase capacity, and simplify the design.

Further disruptions at the system level may result from more subtle technology changes: Merging processing and memory chips into a single chip reduces memory latency, increases the memory bandwidth, and improves the energy efficiency [30]. Nevertheless, this approach has scaling limits because of the high power consumption of DRAM. In the future, PCM combined with 3D-stacking technologies will allow a collection of processor and PCM dies to be mounted on a single chip. By bringing processors together with non-volatile storage, many intervening levels of the storage hierarchy can be eliminated (see Section 3). In particular, so-called through-silicon vias can provide wide, low-energy data paths between the processors and the data stores. Equipped with network interfaces and onboard connectors, a cluster of such tightly-integrated processors and data stores will be particularly suitable for data-centric computing. One such proposed architecture is Nanostore [31], in which PCM makes it possible to store all data in a flat memory hierarchy that replaces traditional disk and DRAM layers.

7. Conclusions and outlook

The character of enterprise and scientific workloads is shifting from being compute-centric to being data-centric, requiring low latency and high bandwidth in a diverse set of access patterns. These requirements are best addressed with in-memory computing, i.e., by more main memory that is directly accessible by processors. PCM is the prime candidate to scale main memory beyond the DRAM. The adoption of PCM for persistent main memory has certain implications on the software and hardware architecture:

a) Synchronous writes to PCM can be used to supersede traditional optimizations for disk systems. This allows simplifications in existing implementations (e.g., file systems and persistency interfaces) and fosters the emergence of transactional memory approaches. Augmented with hardware support, transactional memory on PCM provides durability with significant performance benefits.

b) The persistency of PCM requires measures to protect against unrecoverable system failures and consistency violations: Firstly, references to and from data structures in non-volatile memory require additional protection on the software level. Secondly, procedures such as a "clean reboot" must be explicitly tailored for PCM. Thirdly, standardized data formats for persistent data form an essential part of the software architecture. The conversion from older formats becomes an integral part of any upgrade procedure.

c) If the content of persistent memory must be protected with encryption, the advantage of the byte addressability offered by PCM can no longer be fully exploited because of the cipher block modes used. This can be addressed with hybrid DRAM-PCM memory architectures.

When the typical dataset no longer fits into the main memory, storage subsystems are still indispensable. In the light of the increasing predominance of multi-processing environments using hundreds of cores, request streams to the storage subsystem tend to transform into random access patterns. Therefore, designers of future PCM-aware storage subsystems will have to take the following into account:

a) Using PCM as a write cache or as a log supporting in-place updates will be necessary to alleviate the poor performance and lifetime of Flash under random writes. With PCM, in-place updates are supported, even in "logging" structures, thus eliminating the need to read multiple locations before the original page can be reconstructed.

b) Storing the controller- and filesystem metadata on PCM will reduce the access latency and enable high-granularity mappings and statistics tracking. Because metadata is more frequently updated than data, storing metadata on PCM will also significantly reduce write amplification on Flash. As PCM is byte-addressable and persistent, it is particularly suitable for holding metadata of storage systems that are often stored and accessed in the sizes of tens to hundreds of bytes.

c) To achieve the extremely low latency of PCM at the system level, it will be necessary to eliminate much of the overhead caused by legacy code in the I/O software stack of operating systems, filesystems and databases, e.g., by removing legacy I/O schedulers and replacing locking structures with lock-free counterparts.

d) RAID controllers will have to be re-architected to support the latency and throughput capabilities of PCM-based devices, e.g., by using specialized chips instead of the general-purpose processors found in today's RAID adapters.

The scalability, performance and endurance advantages of PCM over other storage media, such as Flash, DRAM, and disks, will increase its impact on the overall architecture for data-intensive systems. In summary:

a) The share of PCM-based clusters in data-intensive applications and data centers will increase. In these clusters, PCM may play a dual role as memory and storage, enabling scaling to high capacities with low latencies. Directly-attached PCM in large clusters may be used as a distributed caching layer, speeding up applications in a cost-efficient way.

b) Alternatively, PCM may be used as a dedicated storage tier to provide low latency and high performance. Ultimately, this architecture leads to the collocation of processors with PCM for extremely high-throughput computing systems at high parallelism.

c) In light of the proliferation of multi-core systems and the shortened distance between the CPU and the data stored on PCM, future operating systems, filesystems, and databases will need to scale very well to tens or hundreds of processing cores accessing the same data simultaneously with the lowest possible latency.

References

[1] Eleftheriou, E., Haas, R., Jelitto, J., Lantz, M., and Pozidis, H. Trends in Storage Technologies. IEEE Data Engineering Bulletin, 33(4), Dec. 2010.

[2] Zhou, P., Zhao, B., Yang, J., and Zhang, Y. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. Proc. ISCA, 2009.

[3] Qureshi, M. K., Karidis, J., Franceschini, M., Srinivasan, V., Lastras, L., and Abali, B. Enhancing Lifetime and Security of PCM-Based Main Memory with Start-Gap Wear Leveling. Proc. MICRO, 2009.

[4] Qureshi, M. K., Srinivasa, V., and Rivers, J. A. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. Proc. ISCA, 2009.

[5] Qureshi, M. K., Franceschini, M., and Lastras, L. Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. Proc. HPCA, 2010.
[6] Lee, B, Ipek, E., Mutlu, O., and Burger, D. Architecting PCM as a Scalable DRAM Alternative. Proc. ISCA, 2009.
[7] Qureshi, M. K., Franceschini, M., Lastras, L., and Karidis, J. P. Morphable Memory System: A Robust Architecture for Exploiting Multi-Level Phase Change Memories. Proc. ISCA, 2010.
[8] Satyanarayanan, M., Mashburn, H. H., Kumar, P., Steere, D. C., and Kistler J. J. Lightweight Recoverable Virtual Memory. ACM Transactions on Computer Systems 12(1), Feb. 1994.
[9] Biswas, R., and Ort, E. The Java Persistence API - A Simpler Programming Model for Entity Persistence. http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html, May 2006.
[10] Herlihy, M., Moss, J., and Eliot, B. Transactional Memory: Architectural Support for Lock-Free Data Structures. Proc. ISCA, 1993.
[11] Mohan, C. Repeating History beyond ARIES. Proc. VLDB, 1999.
[12] Condit, J., Nightingale, E. B., Frost, C., Ipek, E., Lee, B., Burger D., and Coetzee, D. Better I/O through Byte-Addressable, Persistent Memory. Proc. SOSP, 2009.
[13] Prabhakaran, V., Rodeheffer, T. L., and Zhou, L. Transactional Flash. Proc. OSDI, 2008.
[14] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson S. NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories. Proc. ASPLOS, 2011.
[15] Kim, J. K., Lee, H. G., Choi, S., and Bahng, K. I. A PRAM and NAND Flash Hybrid Architecture for High-Performance Embedded Storage Subsystems. Proc. EMSOFT, 2008.
[16] Caulfield, A., De, A., Cobur,n J., Mollov, T., Gupta, R., and Swanson, S. Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-Volatile Memories. Proc. IEEE MICRO, 2010
[17] Jung, J., Won, Y., Kim, E., Shin, H., and Jeon, B. FRASH: Exploiting Storage Class Memory in Hybrid File System for Hierarchical Storage. Proc. ACM TOCS, 2010
[18] Park, Y., Lim, S., Lee, C., and Park, K. PFFS: A Scalable Flash Memory File System for the Hybrid Architecture of Phase Change RAM and NAND Flash. Proc. ACM SAC, 2008
[19] Stonebraker, M., Abadi, D., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. C-Store: A Column-Oriented DBMS. Proc. VLDB, 2005.
[20] Koltsidas, I., and Viglas, S. Flashing Up The Storage Layer. Proc. VLDB, 2008.
[21] Bhattacharjee, B., Mustafa, C., Lang, C., Mihaila, G., and Ross, K. Storage Class Memory Aware Data Management. IEEE Data Engineering Bulletin, 33(4), 35-40, 2010.
[22] Fang, R., Hsiao, H., He, B., Mohan, C., and Wang, Y. High Performance Database Logging using Storage Class Memory. Proc. ICDE, 2011.
[23] Chen, S., Gibbons, P., and Nath, S. Rethinking Database Algorithms for Phase Change Memory. Proc. CIDR, 2011.
[24] Memcached. http://memcached.org/
[25] MemcacheDB. http://memcachedb.org/
[26] Fitch, B. G., Rayshubskiy, A., Ward, T. J., Pitman, M., Metzler, B., Schick, H. J., Krill, B., Morjan, P., and Germain, R. S. Blue Gene Active Storage. Proc. HEC FSIO, 2010.
[27] Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Rosenblum, M., Rumble, S. M., Stratmann, E., and Stutsman, R. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. Proc. ACM SIGOPS 2009.
[28] Caulfield, A. M., Grupp, L. M., and Swanson. S. Gordon: An Improved Architecture for Data-Intensive Applications. Proc. IEEE Micro, 2010.
[29] Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan. V. FAWN: A Fast Array of Wimpy Nodes. Proc. SOSP, 2009.
[30] Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., and Yelick, K. A Case for Intelligent RAM: IRAM. Proc. IEEE Micro, 1997.
[31] Ranganathan, P. From Microprocessors to Nanostores: Rethinking Data-Centric Systems. IEEE Computer, 44(1), 39-48, 2011.