BIBIM: A Prototype Multi-Partition Aware Heterogeneous New Memory

Gyuyoung Park[1], Miryeong Kwon[1], Pratyush Mahapatra[2], Michael Swift[2], and Myoungsoo Jung[1]
Computer Architecture and Memory Systems Laboratory (CAMEL),
Yonsei University[1], University of Wisconsin-Madison[2]
http://camelab.org

Abstract

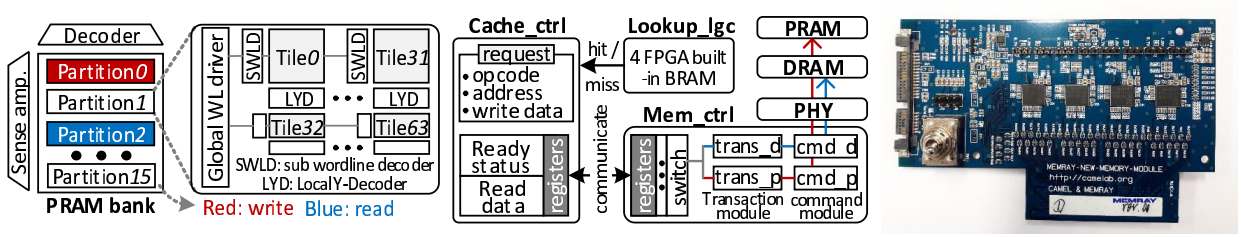
We describe a prototype multi-partition aware new memory controller and subsystem, BIBIM, which precisely integrates DRAM with 3x nm phase change RAM (PRAM) modules. In this work, we reveal the main challenges of a new type of PRAMs in getting closer to a main processor by evaluating our real 3x nm PRAM with diverse persistent memory benchmarks. BIBIM implements hybrid cache logic into a 2x nm FPGA device, which can hide the long latency imposed by the underlying PRAM modules as well as can support persistent operations. The cache logic of our controller is also able to serve multiple read requests while writing data into a target PRAM bank by considering the multi-partition architecture of such new memory. The evaluation results demonstrate that the read and write latencies of our BIBIM are 115 ns and 125 ns, which are 38% and 99% shorter than those of a pure PRAM-based memory subsystem. In addition, BIBIM can remove blocking reads by 53%, on average, thereby shortening the average latency of write-after-read memory operations by 48%.

1 Introduction

New memory systems offer data consistency even when there is an unexpected power loss or system crash. Typically, these memory systems combine storage class memory, such as 3D XPoint (e.g., Optane DIMM) or fast flash (e.g., NVDIMM-F/N) with battery-backed DRAM modules [17, 39]. NVDIMM-P is similar to the aforementioned memory systems, but it is expected to employ byte-addressable non-volatile memory (NVM), such as phase change RAM (PRAM) [27, 37, 48] or memrister-based RAM (RRAM) [21, 25, 28, 40] with a few DRAMs. Even though these hybrid NVDIMM technologies [7, 14, 39] have potential to optimize the system costs and performance (by reducing the crash recovery times), unfortunately, there is no publicly-available prototype that reveals specific information for the internal architecture and controller design of such new memory systems.

In this work, we design and implement BIBIM, a prototype multi-partition aware New memory that incorporates a heterogeneous memory controller and subsystem into a 2x nm FPGA device. We evaluate the real performance of 3x nm new PRAM modules and analyze their challenges to be realized as a new memory solution in modern systems. Specifically, while the majority of published research in architecture and system areas assumes that the memory timing parameters of PRAM are similar to or slightly worse than DRAM timings [10, 19, 27, 29, 47], we observe that the write latency of PRAM is not as promising as what the studies expect. To address these challenges, this work designs hybrid cache logic that integrates a small amount of DRAM into the new PRAM, and also offers a new memory scheduling technique, called *non-blocking read service* (NBRS). The cache logic in our controller uses the DRAM to hide the long latency of the underlying PRAM modules by considering persistent memory (PM) operations, such as PM write ordering and flushing commands. Many in-flight read services can be potentially blocked owing to a single write, which is scheduled by an internal task of the controller, such as a cache eviction or persistent operation. Even in cases where all such memory requests target the same PRAM bank, the proposed NBRS technique enables our controller to serve the in-flight read requests in parallel with the write by exploiting PRAM internal architecture, referred to as *multi-partition* architecture.

We evaluate BIBIM and compare the proposed approaches with other memory subsystem technologies (i.e., DRAM-only and PRAM-only memory) by executing diverse persistent memory benchmarks implemented by two NVM libraries [11, 42] upon the real memory subsystems. Overall, the reads and writes of our heterogeneous memory subsystem take 115 ns and 125 ns, respectively. These average read and write laten-



(a) PRAM internals. (b) Architecture overview. (c) PRAM module.

Figure 1: PRAM details and overview of BIBIM.

cies of BIBIM are 38% and 99% shorter than those of the PRAM-only memory subsystem, respectively; the latency characteristics of our heterogeneous memory system are slightly worse than the DRAM-only memory subsystem under the executions of diverse persistent workloads.

2 Background

PRAM operations. PRAM is a resistive memory whose storage core exploits the unique behavior of chalcogenide glass [15, 32, 38]. The storage core of PRAM exhibits two different resistive states: i) *amorphous* and ii) *crystalline*, each respectively representing two digits, '0' and '1' [44, 45]. When a passage of currents produces a temperature, near 300°C (via a heating element), the core status changes to the amorphous that exhibits a low resistance ('1'). In cases where the applied temperature is higher than a melting threshold (around 600°C), the status of storage core is updated to the crystalline, which indicates a high resistance ('0') [9, 36]. Memory subsystems can recognize the target data based on a difference of resistance values that the storage core represents.

A techinical issue behind these phase change processes is that a write on PRAM requires a long operation time to pump charges and to change the status between amorphous and crystalline whereas a read operation is significantly faster than the write (as it only needs to sense out the current from the target). The prior circuit-level studies [27, 29, 47] forecast the read and write latencies of PRAM around 60 ns and 150 ns, respectively, which are similar to or slightly worse than DRAM latency characteristics. Even though many system-level researches leverage this latency model to simulate PRAM memory subsystem [10, 33, 41], we observe that a charge pump logic and a cell program operation take 8 us and 10 us, respectively, which make a write approximately 133 time longer than the write latency of the most prior studies used. Our hybrid controller is designed towards addressing the challenges brought by such long latency on PRAM writes.

Multi-partition architecture. Figure 1a illustrates the internal architecture of our 3x nm PRAM bank. In contrast to a conventional DRAM bank, the PRAM bank is composed of 16 different memory partitions, each being able to serve one or more requests in parallel. A partition (within a bank) contains 64 memory islands, called *tiles*, each employing its own local Y-decoder (LYD) [23, 24]. To address the circuit-level challenges of PRAM such as parasitic resistance and sneak path, all the tiles within a partition are connected to the corresponding partition's local sub-wordline driver (LSWD), which is linked to the main global wordline driver [5, 22, 23]. These separate local decoders and wordline drivers allow a system to potentially access different partitions in parallel. However, there are two limits for the simultaneous memory accesses to multiple partitions within a PRAM bank. First, different partitions can support parallel I/O services "only if" incoming memory requests have a different operation type. For example, while a read and a write (and/or an erase) can be served in parallel from two different partitions, two parallel reads (or two writes) across the two corresponding memory partitions are prohibited. Second, since the global wordline and datapath are shared across all the partitions, data transfers (requested from a processor) should be serialized.

Note that, even though these characteristics of the multi-partition architecture are not as powerful as the multi-die architecture of flash, which allows multiple writes or multiple reads across different dies (within a flash package) [8, 18, 35], we believe that the multiple partitions should be recognized and appropriately exploited from a PRAM controller. Our PRAM controller can serve multiple reads when a write/erase operation is in service, which can hide the long latency of a PRAM write. These multiple reads can also be well harmonized with several system-level parallelization techniques, such as memory bank interleaving [2, 13, 34] and channel striping [20, 46].

3 Heterogeneous New Memory Controller

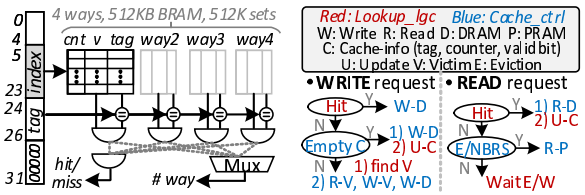
BIBIM uses DRAM (12.5% of total memory capacity) as an inclusive cache, but it caches only write requests since the read performance of PRAM is comparable with DRAM. One of the challenges for the internal DRAM

cache is to guarantee the data persistence in cases of power loss. To address this issue, BIBIM supports persistent memory fence (PM_FENCE) and flush (PM_FLUSHOPT) [31], which are used by NVM libraries [11, 42] and processor instructions, such as Intel's cache line write back (clwb) [12]. Specifically, we guarantee that DRAM-cached data, associated with the PM_FENCE requests, are written to PRAM when a PM_FLUSHOPT command is received. In our design, PM_FENCE operations guarantee an order of delivery based on processor requests, while the PM_FLUSHOPT operations support guaranteed delivery to the underlying PRAM modules. In addition, to hide the long write latency caused by a cache eviction or persistent operation, BIBIM employs multi-partition aware non-blocking read service (NBRS).

Note that the hybrid memory design of BIBIM is used as a byte-addressable working memory, which has a different goal, compared to the previous first-generation PRAM-based SSDs that use a PCIe block interface [3].

Basic building blocks. Figure 1b depicts a high-level view of BIBIM's internal architecture. An incoming memory request consists of an operation type, address, and target data. The cache controller, called cache_ctrl, checks if the target data is in DRAM by using our lookup logic. The lookup logic (lookup_lgc) contains four FPGA built-in storage modules, which are a byte-addressable block RAM (BRAM) [26]. We configure these BRAMs to retrieve all the data within 2 cycles in our prototype implementation. When cache_ctrl decides the target memory module (DRAM or PRAM) for the request service, our cache logic accesses the underlying memory controller (i.e., mem_ctrl), which contains DRAM and PRAM transaction modules, called trans_d and trans_p, respectively. For a better communication between cache_ctrl and mem_ctrl, mem_ctrl exposes a switch register that indicates which memory module will be used and a set of registers that contains the request information such as operation type, address and write-related data. In addition, mem_ctrl exposes two different registers, each containing a ready status and the read data in attempt to support asynchronous operations. The hybrid memory modules that we implemented are shown in Figure 1c.

Heterogeneous interface control. Our controller employs a transaction module (trans_p) to manage different memory operations on the heterogeneous memory interface (DRAM/PRAM). Specifically, trans_p generates three LPDDR writes for a single memory request, each containing the command code/address, data, and execution command of the memory request; for the DRAM operations, trans_d directly forwards the incoming memory requests to the underlying *DRAM command module* (cmd_d). Specifically, cmd_d sends row/column addresses (through an active command) and



(a) Lookup logic. (b) Control flow.

Figure 2: Lookup logic and cache control.

a read/write command to the target DRAM module, whereas our *PRAM command module* (cmd_p) converts each request (received from trans_p) into an LPDDR NVM request. The LPDDR request is composed of three different memory phases; preactive phase, active phase and read/write phase of the target memory device. In contrast to the PRAM commands, the DRAM commands bring the corresponding row address through two memory phases, preactive and active; the target column address is issued with a read/write command. All the memory timings of these commands (issued to the target memory module) are managed by the hybrid cache physical layer (PHY) that we implemented.

Multi-partition aware NBRS. In our design, cache_ctrl stores the issued memory address related to the latest write on PRAM, and it compares the stored address with all incoming read request address(es). Specifically, in cases where there is a read request, cache_ctrl first checks if the target data is cached via lookup_lgc. If lookup_lgc reports a hit, cache_ctrl immediately serves the request from the underlying DRAM via trans_d. Otherwise, cache_ctrl examines if there is a conflict on the target partition (within a PRAM bank) by comparing upper 4 bits of the issued write and the incomming read addresses (which indicates the target partition number). When the target partition numbers for the write and read are different, cache_ctrl composes a read command to trans_p, thereby serving the read in parallel with a DRAM write in progress. If the target partition has a conflict, cache_ctrl suspends the reads until the partition is available for a memory service. Note that, without this NBRS technique, we observe that all read requests will be delayed, whenever there is a cache eviction or a persistent operation which can block upto 108 read services .

4 Details of Implementation

Lookup logic. Figure 2a shows our lookup_lgc implementation that contains the four built-in BRAMs. Each BRAM includes a tag array whose entry size is 3 bits. In addition, each BRAM maintains a counter (2 bits) and a valid bit (1 bit), which are used for our LRU replace-

Inter -face	Workloads	#req	Read ratio(%)	Write ratio(%)	Fence ratio(%)	Flush ratio(%)	Brief description
Native	echo	64K	46	28	26	0	Scalable key-value store [6]
	tpcc	196K	66	15	8	11	H-store like DB. Undo logs for consistency (TPC-C like) [4]
	ycsb	188K	69	12	8	10	H-store like DB. Undo logs for consistency (YCSB like) [4]
Library	ctree	182K	55	37	8	0	NVML, Micro benchmark (INSERT and DELETE) for persistent crit-bit tree [1]
	hashmap	189K	67	22	11	0	NVML, Micro benchmark (INSERT and DELETE) for persistent hashmap [1]
	vacation	0.24K	9	86	5	0	Mnemosyne, online travel reservation system [30]

Table 1: Important characteristics of persistent memory traces collected from [31]

ment module of `lookup_lgc`. In our implementation, `lookup_lgc` uses 512 KB BRAM for cache and LRU information, which occupy only 50% gates of the built-in BRAM space of our target FPGA platform [43]. A cache way of our controller contains 512 thousand sets, each taking 1 byte, which can indicate a 32 byte DRAM block. The address of an incoming request (32 bits) for a PRAM splits into a 3 bit tag and a 19 bit cache index, such that all BRAMs are enabled with the parsed cache index in parallel. The tag information and valid bit for each way can be simultaneously sensed, and all of the tags are simultaneously compared with the input tag by our four simple comparators. The comparison results and their valid bits are then forwarded to the corresponding AND gate. The hit result is delivered by an XOR gate while the target way (in cases of a cache hit) is filtered by a 4*1 MUX logic. On the other hand, our LRU replacement module simply compares the counter of four cache sets, associated with a target index, and returns the way number that has an initial counter value (the oldest set). In our design, the initial counter value is 0x100; when there is an update request from `cache_ctrl`, it decreases the counter. The `lookup_lgc` completes all the cache lookup and LRU search processes, including BRAM accesses, within 2 cycles. These short cycles are even invisible to a processor as they are completely overlapped with the latency of DRAM/PRAM operations.

Cache controller. Figure 2b shows the control flow of our `cache_ctrl`. When `lookup_lgc` returns a cache miss, `cache_ctrl` checks if an empty cache line (associated with the underlying DRAM space) exists across different cache ways. When there is an empty cache line, `cache_ctrl` composes a write request to `trans_d` and updates the tag and LRU information via `lookup_lgc`. Otherwise, `cache_ctrl` raises a signal to the LRU replacement module for `lookup_lgc` to retrieve the way that contains 0x100 (the oldest set). In our implementation, this replacement module can detect the victim line within a single cycle. Then, `cache_ctrl` reads the victim data from the underlying DRAM through `trans_d` and writes it back to `trans_p`. Lastly, `cache_ctrl` writes the new data to the DRAM and updates `lookup_lgc` to clean up the cache information. If the cache miss is related to a read, `cache_ctrl` checks if there is no cache eviction in progress or NBRS is available to use. If it is, `cache_ctrl` issues a read to `trans_p`. Otherwise, `cache_ctrl` waits for the eviction or write. In cases of a cache hit, `cache_ctrl` generates the corresponding DRAM request, issues `trans_d`, and updates the target cache line via `lookup_lgc`.

Persistent operation. When `cache_ctrl` receives a `PM_FLUSHOPT` request, it checks `lookup_lgc` and enforces write the data (corresponding to the `PM_FLUSHOPT`'s address) back to the underlying PRAM in background. This is performed by converting the persistent request to multiple writes managed from our cache eviction module. `cache_ctrl` serializes the writes based on the order of `PM_FENCE` requests received. If there is a read, after the `PM_FENCE`, `cache_ctrl` is still available to serve the read through NBRS. The background operation is turned into a foreground task when `cache_ctrl` receives a `PM_FENCE` request, and all I/O services will be suspended if a write is requested after the `PM_FENCE`. Thus, the host can issue many reads without a stall even after `PM_FENCE` (but before another write is issued) with our NBRS mechanism.

5 Evaluation

Workloads and configuration. We use persistent memory workload traces, which are extracted from Whisper [31]. The platform that collected the traces employs eight CPUs (Intel i7-6700, 3.4GHz), 8MB last level cache (LLC) and 32GB memory subsystems, guided by the persistent model of [31]. Table 1 summarizes the important characteristics and short descriptions for each workload that we evaluated; *echo*, *tpcc*, and *ycsb* employ a native NVM interface, which accesses the underlying memory subsystems by using persistent instructions such as `clflushopt` [31] and `sfence` [16], while *ctree*, *hashmap*, and *vacation* use a software-based interface whose system applies NVM libraries such as NVML [11] and Mnemosyne [42]. Note that, while most workloads only generate persistent ordering commands (i.e., `PM_FENCE`) to apply their consistency model, *tpcc* and *ycsb* introduce many flush instructions (i.e., `PM_FLUSHOPT`) in addition to the persistent ordering commands, since they have multiple undo logs and periodically flush all such log records to the underlying memory subsystems in order to support the data consistency. Most workloads intensively generate reads, which

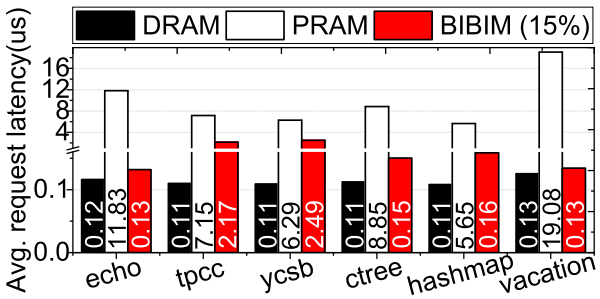
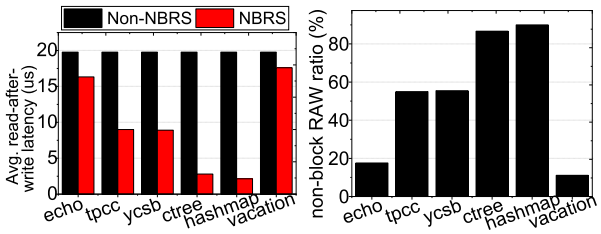


Figure 3: Latency comparison for three different memory subsystems.

is an usual behavior of application execution. However, *vacation* is a write-intensive application (its write fraction is 86%). This is because the client threads of *vacation* perform a number of write-related transactions for travel reservations and cancellations.

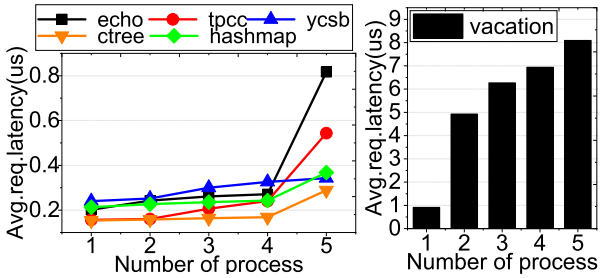
Latency comparisons. Figure 3 compares the latency of DRAM-only (DRAM), PRAM-only (PRAM) and our BIBIM memory subsystems by executing the persistent memory workloads. As shown in the figure, DRAM exhibits an excellent latency characteristic, average latencies ranging from 0.116 us to 0.125 us for all workload executions, while the shortest latency of PRAM is longer than 5 us on *hashmap*. This is because, while the read latency of PRAM is 185 ns, which is comparable with that of DRAM, the write latency of PRAM is 20 us; the actual latency of PRAM is 190x longer than that of DRAM. Due to this long write latency, the performance of PRAM is, on average, 85x worse than DRAM in the write-intensive workloads (cf. Table 1). This performance degradation becomes notable for *vacation* whose write fraction is 86% of the total execution. In contrast, the hybrid architecture of BIBIM shortens the average latency by 87% compared to PRAM. However, BIBIM has less performance benefits in the workloads *tpcc* and *ycsb*. Specifically, BIBIM decreases the average latency of *tpcc* and *ycsb* by around 30% and 40%, respectively, compared to PRAM. The reason behind this performance diminishing is that the persistent operations of those workloads enforce BIBIM write cached data back to the underlying PRAM in a synchronous manner. Nevertheless, considering the overall performance of BIBIM, we believe that it is reasonably acceptable to put PRAM between DRAM memory and storage subsystems. Lastly, even compared with the DRAM, BIBIM increases the average latency by 44%, on average.

Non-blocking Read Service analysis. Figure 4a compares the latency of non-NBRS and NBRS (applied to BIBIM), which is also related to the internal parallelism of multiple partition architecture. In this evaluation, the latency is measured for the reads submitted to `cache_ctrl` right after a cache eviction or a persistent



(a) Average RAW latency. (b) non-blocking read ratio.

Figure 4: NBRS characteristics under various workloads.



(a) Read-intensive (b) Write-intensive.

Figure 5: Latency analysis of BIBIM with different numbers of workload executions.

operation, which is referred to as *read-after-write (RAW)*. The RAW latency of non-NBRS is 19.8 us, on average, across all the workloads that we tested. This is because all the reads are blocked when the target PRAM bank is involved in writing data into its memory island, which increases the actual read latency than that of normal operations by 107 times, on average. In contrast, NBRS-applied BIBIM can successfully reduce the RAW latency by 52%, on average. This performance advantage becomes more promising if read and write operations are well balanced. For example, *ctree* and *hashmap* exhibit 2.8 us and 2.1 us for its RAW latency, which are 70% and 77% shorter than other workloads respectively. While the advantage of NBRS diminishes for the workloads that have many writes (*vacation*) or PM_FENCEs (*echo*), it can still reduce the RAW latency compared with non-NBRS by 11% (*vacation*) and 18% (*echo*), respectively. Figure 4b analyzes the fraction of non-blocking reads of the total read requests. NBRS-applied BIBIM reduces the number of reads, which are blocked by a cache eviction or a persistent operation by 11% $\sim$ 90%. This in turn can reduce CPU stalls compared with non-NBRS by 6.5%, on average.

Stress tests with multiple processes. To evaluate the worst-case performance scenario, we perform a stress test by increasing the number of processes, each executing a same workload application, but working on a different address offset based on its process ID. Figure 5a illustrates the overall latency of our BIBIM memory subsystem. By executing four processes, the overall la-

tency increases by 30%, on average, which is insignificant by considering the many process executions. The overall latency increases 300 ns, on average, when we have more than four. This is because `cache_ctrl` introduces more cache conflicts to serve the heavy memory requests. Especially the latency of *echo* and *tpcc* is as high as 800 ns and 500 ns, respectively. The reason behind such performance degradation is that *echo* exhibits many persistent operations related to write ordering and `PM_FLUSHOPT`, respectively (in addition to the heavy memory requests coming from many process executions). As shown in Figure 5b, even though *vacation* has few `PM_FENCE` commands, the performance degrades by 7 us at the worst. Since the writes in *vacation* account for 86% of the total memory requests, `cache_ctrl` generates many cache evictions with the executions of multiple processes, which in turn increases than the latency by 783%, compared with a single process execution scenario. Note that, even though BIBIM shows the performance degradation with multiple processes in this stress test, still the worst-latency is much shorter than the average latency of PRAM-only memory subsystem. For example, the worst-case latency BIBIM on the workload *echo* and *vacation* executions, it exhibits 7% and 42% better latency characteristics compared with that of `PRAM`.

6 Acknowledgement

This work is supported by MemRay grant (2015-11-1731). The authors thank Swapnil Haria and Pratyush Mahapatra who prepared the trace collection environment. Gyuyoung Park and Miryeong Kwon equally contribute to this work. The authors also thank MemRay and Samsung for their engineering sample donations and technical support. Myoungsoo Jung is the corresponding author.

7 Conclusion

We implemented BIBIM, a prototype multi-partition aware new memory controller and subsystem that precisely integrates DRAM with 3x nm phase change RAM (PRAM). Through empirical evaluations on the real system that we implemented, this work demonstrated that our BIBIM can successfully hide the long latency imposed by the underlying PRAM modules and can support persistent operations. The latency characteristics of our BIBIM are on average 87% better than those of PRAM-only memory and are similar to or slightly worse than DRAM-only memory.

References

[1] Crit-bit tree. `cr.yp.to/critbit.html`.

[2] AKAOGI, T., CLEVELAND, L. E., AND NGUYEN, K. Multiple bank simultaneous operation for a flash memory, May 29 2001. US Patent 6,240,040.

[3] AKEL, A., CAULFIELD, A. M., MOLLOV, T. I., GUPTA, R. K., AND SWANSON, S. Onyx: A prototype phase change memory storage array. *HotStorage 1* (2011), 1.

[4] ARULRAJ, J., PAVLO, A., AND DULLOOR, S. R. Let's talk about storage & recovery methods for non-volatile memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (2015), ACM.

[5] BAE, Y.-C., PARK, J.-Y., RHEE, S. J., KO, S. B., JEONG, Y., NOH, K.-S., SON, Y., YOUN, J., CHU, Y., CHO, H., ET AL. A 1.2 v 30nm 1.6 gb/s/pin 4gb lpddr3 sdram with input skew calibration and enhanced control scheme. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International* (2012), IEEE.

[6] BAILEY, K. A., HORNYACK, P., CEZE, L., GRIBBLE, S. D., AND LEVY, H. M. Exploring storage class memory with key value stores. In *Proceedings of the 1st Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads* (2013), ACM, p. 4.

[7] CHANG, J., AND SAINIO, A. Nvdimm-n cookbook: A soup-to-nuts primer on using nvdimm-ns to improve your storage performance, 2015.

[8] CHEN, F., LEE, R., AND ZHANG, X. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on* (2011), IEEE.

[9] CHEN, Y., ZHAO, J., AND XIE, Y. 3d-nonfar: Three-dimensional non-volatile fpga architecture using phase change memory. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on* (2010), IEEE.

[10] CHO, S., AND LEE, H. Flip-n-write: a simple deterministic technique to improve pram write performance, energy and endurance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture* (2009), ACM.

[11] 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. *ACM Sigplan Notices 46*, 3 (2011), 105–118.

[12] COOPERATION, I. Intel architecture instruction set extensions programming reference. *Intel Corp., Mountain View, CA, USA, Tech. Rep* (2016), 319433–030.

[13] GRUPP, L. M., DAVIS, J. D., AND SWANSON, S. The harey tortoise: Managing heterogeneous write performance in ssds. In *USENIX Annual Technical Conference* (2013).

[14] GUDDEKOPPA, V. Method and system providing file system for an electronic device comprising a composite memory device, 2016. US Patent App. 15/390,021.

[15] IELMINI, D., LAVIZZARI, S., SHARMA, D., AND LACAITA, A. L. Physical interpretation, modeling and impact on phase change memory (pcm) reliability of resistance drift due to chalcogenide structural relaxation. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International* (2007), IEEE.

[16] INTEL CORPORATION. Persistent memory programming. `http://pmem.io`.

[17] INTEL CORPORATION. Intel optane technology. `https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html`, 2016.

[18] JUNG, M. Exploring parallel data access methods in emerging non-volatile memory systems. *IEEE Transactions on Parallel and Distributed Systems 28*, 3 (2017), 746–759.

[19] JUNG, M. Nearzero: An integration of phase change memory with multi-core coprocessor. *IEEE Computer Architecture Letters 16*, 2 (2017), 136–140.

[20] JUNG, M., AND KANDEMIR, M. T. An evaluation of different page allocation strategies on high-speed ssds. In *HotStorage* (2012).

[21] JUNG, M., SHALF, J., AND KANDEMIR, M. Design of a large-scale storage-class rram system. In *Proceedings of the 27th international ACM conference on International conference on supercomputing* (2013), ACM.

[22] KANG, M., PARK, T., KWON, Y., AHN, D., KANG, Y., JEONG, H., AHN, S., SONG, Y., KIM, B., NAM, S., ET AL. Pram cell technology and characterization in 20nm node size. In *Electron Devices Meeting (IEDM), 2011 IEEE International* (2011), IEEE.

[23] KANG, S., CHO, W. Y., CHO, B.-H., LEE, K.-J., LEE, C.-S., OH, H.-R., CHOI, B.-G., WANG, Q., KIM, H.-J., PARK, M.-H., ET AL. A 0.1um 1.8-v 256-mb phase-change random access memory (pram) with 66-mhz synchronous burst-read operation. *IEEE Journal of Solid-State Circuits 42*, 1 (2007), 210–218.

[24] KIM, D. K., AND YOON, T. H. Phase change memory apparatus and test circuit therefor, Mar. 6 2012. US Patent 8,130,541.

[25] KIM, K.-H., GABA, S., WHEELER, D., CRUZ-ALBRECHT, J. M., HUSSAIN, T., SRINIVASA, N., AND LU, W. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano letters 12*, 1 (2011), 389–395.

[26] LAFOREST, C. E., AND STEFFAN, J. G. Efficient multi-ported memories for fpgas. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays* (2010), ACM.

[27] LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable dram alternative. In *International Symposium on Computer Architecture (ISCA)* (2009).

[28] LINN, E., ROSEZIN, R., KÜGELER, C., AND WASER, R. Complementary resistive switches for passive nanocrossbar memories. *Nature materials 9*, 5 (2010), 403.

[29] LIU, S., KOLLI, A., REN, J., AND KHAN, S. Crash consistency in encrypted non-volatile main memory systems.

[30] MINH, C. C. *Designing an effective hybrid transactional memory system*. Stanford University, 2008.

[31] NALLI, S., HARIA, S., HILL, M. D., SWIFT, M. M., VOLOS, H., AND KEETON, K. An analysis of persistent memory use with whisper. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (2017), ACM.

[32] OVSHINSKY, S. R. Reversible electrical switching phenomena in disordered structures. *Physical Review Letters 21*, 20 (1968), 1450.

[33] PARK, H., YOO, S., AND LEE, S. Power management of hybrid dram/pram-based main memory. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE* (2011), IEEE.

[34] PARK, S.-H., PARK, J.-W., KIM, S.-D., AND WEEMS, C. C. A pattern adaptive nand flash memory storage structure. *IEEE transactions on computers 61*, 1 (2012), 134–138.

[35] PARK, S.-Y., SEO, E., SHIN, J.-Y., MAENG, S., AND LEE, J. Exploiting internal parallelism of flash-based ssds. *IEEE Computer Architecture Letters 9*, 1 (2010), 9–12.

[36] QURESHI, M. K., FRANCESCHINI, M. M., JAGMOHAN, A., AND LASTRAS, L. A. Preset: improving performance of phase change memories by exploiting asymmetry in write times. *ACM SIGARCH Computer Architecture News 40*, 3 (2012).

[37] QURESHI, M. K., SRINIVASAN, V., AND RIVERS, J. A. Scalable high performance main memory system using phase-change memory technology. In *International Symposium on Computer Architecture (ISCA)* (2009).

[38] REDAELLI, A., PIROVANO, A., PELLIZZER, F., LACAITA, A., IELMINI, D., AND BEZ, R. Electronic switching effect and phase-change transition in chalcogenide materials. *IEEE Electron Device Letters 25*, 10 (2004), 684–686.

[39] SAINIO, A. Nvdimm - changes are here so what's next? *In-Memory Computing Summit* (2016).

[40] STRUKOV, D. B., SNIDER, G. S., STEWART, D. R., AND WILLIAMS, R. S. The missing memristor found. *nature 453*, 7191 (2008), 80.

[41] SUN, G., NIU, D., OUYANG, J., AND XIE, Y. A frequent-value based pram memory architecture. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference* (2011), IEEE Press.

[42] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. In *ACM SIGARCH Computer Architecture News* (2011), vol. 39, ACM.

[43] XILINX. 7 series fpgas data sheet. `https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`, 2018.

[44] YANG, B.-D., LEE, J.-E., KIM, J.-S., CHO, J., LEE, S.-Y., AND YU, B.-G. A low power phase-change random access memory using a data-comparison write scheme. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on* (2007), IEEE.

[45] YI, J., HA, Y., PARK, J., KUH, B., HORII, H., KIM, Y., PARK, S., HWANG, Y., LEE, S., AND AHN, S. Novel cell structure of pram with thin metal layer inserted gesbte. In *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International* (2003), IEEE.

[46] YU, F., LEE, C. C., MA, A. C., AND SHIN, M. Multi-level striping and truncation channel-equalization for flash-memory system, Sept. 11 2012. US Patent 8,266,367.

[47] YUE, J., AND ZHU, Y. Accelerating write by exploiting pcm asymmetries. In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on* (2013), IEEE.

[48] ZHOU, P., ZHAO, B., YANG, J., AND ZHANG, Y. A durable and energy efficient main memory using phase change memory technology. In *International Symposium on Computer Architecture (ISCA)* (2009).