NemOS: A Framework for Exploring Architecture-OS Design in Hybrid Memory Systems

Arun KP
Indian Institute of Technology
Kanpur, India
kparun@cse.iitk.ac.in

Debadatta Mishra
Indian Institute of Technology
Kanpur, India
deba@cse.iitk.ac.in

***Abstract*—The hybrid memory system provides the benefit of both DRAM and NVM technology. NVM provides a higher capacity and data persistence, and DRAM provides better performance. A memory manager's goal in a hybrid memory system is to reap the benefits of both technologies for better overall system performance. One of the important decisions to make in a hybrid memory system is placing and migrating pages between NVM and DRAM. Using NVM for persistence requires underlying mechanisms to ensure consistency of memory modifications.**

In this paper, we introduce an open-source framework, NemOS, based on gem5 and GemOS for hybrid memory exploration in architecture and operating systems. NemOS provides a quick way to study hybrid memory systems as it reduces simulated instructions by more than 50% in comparison with Linux. NemOS aids researchers in the operating system and architecture study of NVM hybrid memory systems.

***Index Terms*—Non-volatile memory, NVM, hybrid memory**

I. Introduction

The volume of data generated by different computing applications is constantly increasing over the years, and is predicted to reach 463 zettabytes per day by 2025 [1]. The amount of data generated at this scale drives technologies by creating opportunities for big data computing [43], graph processing [27], and cloud storage services [33]. The advanced and sophisticated big data processing requirements increase the demand for physical memory on compute nodes. Traditionally, service providers relied on the volatile memory (DRAM) to meet these demands. However, the energy consumption and capacity constraints limit feasible DRAM size, hence the service providers have shifted their focus to new memory technologies such as the Non-Volatile Memory (NVM). Apart from support for high capacity, NVM provides persistence removing the cost of data serialization (from DRAM) into the secondary storage (such as HDDs).

NVM is byte addressable and has higher read/write access latency than DRAM [22]. As a result, it is not feasible to use NVM as a drop-in replacement for DRAM, as systems with only NVM may perform poorly compared to their DRAM counterparts [25]. Therefore, a suitable approach to maintain application performance is by using hybrid memory consisting of NVM and DRAM to leverage the combined benefits of both DRAM and NVM. Moreover, as the hybrid memory system allows placing an application's data in NVM and/or DRAM, memory managers can manage allocations efficiently to provide high memory capacity with low access latency. For example, the memory manager could place frequently accessed, hot memory pages in DRAM and migrate less frequently accessed, cold pages, to NVM. Several existing research such as efficient access for large workloads by intelligent data placement strategies across the NVM and DRAM [42], reduced energy consumption using data migration across the DRAM and NVM tiers [45], [23] have demonstrated the benefits of using a hybrid memory system. NVM also provides persistence, enabling it to complement or replace the high latency persistent storage devices like hard disk in a hybrid memory system, providing persistence while providing latency comparable to memory access latency. Application developers can incorporate consistency and durability semantics into the software design, such as persistent object store [18] or allow failure atomicity in lock based multi-threaded programs [44], [10], [17] to tackle the data consistency problems.

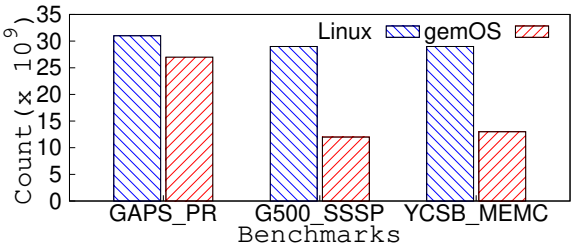


Fig. 1: Comparison of simulated instructions in gem5 for Graph500 and YCSB.

The domain for exploring novel design ideas for hybrid memory systems are either limited to the software solution space (i.e., the OS and the application layer) or the architecture layers. Ideas exploring the software-hardware co-design space require an end-to-end system with support for extension, validation and evaluation. Full system simulators such as Gem5 [26] with the Linux Kernel has been a leading platform to prototype ideas crossing the hardware-software boundaries. However, for hybrid memory systems, the Gem5-Linux platform has the following shortcomings. First, while Gem5 models the NVM controller and the Linux kernel can detect NVM on real hardware (such as Intel DCPMM), the integration is non-trivial, especially considering an evolving hardware technology. Second, Linux kernel is heavyweight

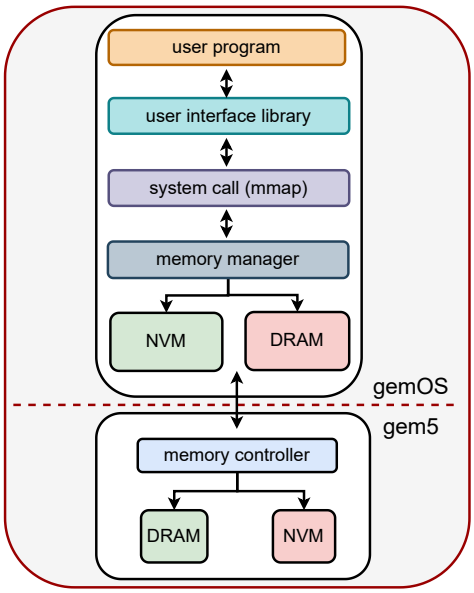


Fig. 2: Schematic diagram of NemOS hybrid memory system.

whereby the OS functions and services can consume a significant part of simulation which may not be desirable for quick prototype of design ideas. As we show in Figure 1, for the benchmark applications (Graph500 and YCSB), Linux executes more than 2x instructions compared to GemOS [28]. Third, designing PoCs in the Linux kernel requires significant understanding and changes in the Linux memory management subsystem which is has non-trivial complexity.

In this paper, we showcase a hybrid memory framework, NemOS[1], for architecture and operating system exploration and prototyping. NemOS bridges the gap in realizing end-to-end design ideas for hybrid memory systems by enabling full-system simulation on gem5 with hybrid memory support. We implement a simple hardware extension in Gem5 to realize the hybrid memory and expose them to the OS. We extend GemOS [28] to configure the hybrid memory and expose user APIs to operate on them. We show a prototype implementation of a state-of-the-art hardware-software scheme (SSP [32]) to demonstrate the efficacy of NemOS in realizing complex design goals and analyzing new insights. SSP provides failure atomicity for NVM updates using shadow sub-paging abd performs cache-line level remapping to maintain old and new copy of data at cache-line granularity in two memory pages. The prototype (section III) enabled to enhance the understanding of the memory consistency scheme like SSP and its influence on end-end-end performance of applications under varying consistency requirements.

II. Design and Implementation

NemOS framework consists of two components, the cycle accurate architecture simulator gem5 [8], [26] and a lightweight operating system, gemOS [28], specialized in executing on gem5. Figure 2 shows the interaction between these two components in NemOS. The NemOS allows researchers and developers to study architecture and operating system interactions in a hybrid memory system with NVM.

[1]Can be downloaded from https://github.com/arunkp1986/NemOS.git

TABLE I: gem5 Memory Configuration

Parameter	Used Setting
DRAM interface	DDR4-2400 16x4
NVM interface	PCM ‡
NVM Write buffer	48
NVM Read buffer	64
Memory capacity	2GB DRAM + 2GB NVM
‡PCM timing parameters based on [40]	

TABLE II: Benchmark Details

Benchmark	Total Ops	read %	write %
GAPBS_PR [7]	1,000,000	94	6
G500_SSSP [30]	1,000,000	66	34
YCSB_MEMC* [14]	1,000,000	83	17
*YCSB workload-B running against Memcached			

```
int main(){

    char* ptr1= (char*)mmap(NULL,4096,PROT_WRITE,
    MAP_NVM); //allocation in NVM
    char* ptr2= (char*)mmap(NULL,4096,PROT_WRITE,0);
     //allocation in DRAM
    ptr1[0] = 'A'; //store to NVM
    ptr2[0] = 'B'; //store to DRAM
    //munmap allocations
    return 0;

}
```

Listing 1: Sample mmap() code to allocate in NVM

The OS component of NemOS exposes the NVM to user applications through `mmap` system call. As shown in the sample code Listing 1, we introduced an additional flag `MAP_NVM` in `mmap` to indicate mapping from NVM. In NemOS, the OS memory manager exposes the hybrid memory (2GB DRAM + 2GB NVM) by partitioning the physical memory address range and creating corresponding BIOS e280 entries in gem5. NemOS configures the NVM memory controller interface in gem5 with specifications mentioned in the Table I to provide a flat address hybrid memory.

Figure 1 shows the benefit of running applications in NemOS, by comparing the number of instructions simulated in gem5 for applications running in Linux and gemOS. In this experiment, we traced applications in Table II using SniP [21], and replayed read-write memory accesses in these traces using a micro-benchmark in Linux and gemOS. The OS component of NemOS helps to reduce the number of simulated instructions, ∼50% reduction for G500_SSSP and YCSB_MEMC in Figure 1.

III. Results

In this section, we show the benefit of using the `NemOS` framework to prototype and perform an initial evaluation of existing or new ideas on a hybrid memory system. We prototyped Shadow Sub-Paging (SSP) [32] to show the capability

of NemOS to fulfill state-of-the-art designs in hybrid memory systems, both in system software and hardware.

SSP comes under the data persistence use case of NVM. SSP ensures memory persistence for an application by allowing a consistent memory state after the application's planned or abrupt end. While tracking changes to the execution state of an application, the memory state of an application contributes significantly to the tracked changes apart from CPU registers, opened files, etc., and prior works on persisting the memory state of an application used logging [9], [39], [16], [11], shadow paging [13], [32] or dual-copy [15]. SSP uses a shadow paging-based approach to ensure memory persistence.

In shadow paging, a private copy of the memory page is made before modification, allowing in-place updates to pages. Shadow paging avoids the write-twice problem in logging-based schemes, in which one change to data also causes another write to make a log entry. In contrast, a dual-copy method maintains two copies of data, one as working and another as a consistent copy. Only the working copy is changed as part of memory modifications, and the consistent copy is updated only after all memory changes are committed [6].

We show the implementation approach that we followed in NemOS and the initial results of SSP in the below section.

Prototype: Shadow Sub-Paging (SSP)

SSP [32] allocates two physical pages for each virtual page and uses a remapping scheme at the cache controller hardware to route modifications to alternate physical pages. SSP ensures consistency of memory modification by maintaining a copy of unmodified data at cache line granularity in one of the two pages. In NemOS, we allocate the additional physical page as part of the page allocation routine call in gemOS. The original and the extra page addresses are recorded in a metadata area (i.e., SSP cache).

The translation lookaside buffer (TLB) [2] extension in SSP [32] adds extra fields per entry to capture the supplementary physical page mapping, current bitmap, and updated bitmap. We extended the page walker hardware in gem5 x86-64 to fill these fields on a TLB miss during address translation. TLB may contain translations for DRAM and NVM pages in a hybrid memory system, and the memory consistency requirement applies only to NVM pages. Therefore, in the prototype design, we used Model Specific Registers (MSRs) [3] to communicate the virtual address range corresponding to NVM allocation. We also use MSR to pass the base address of the metadata region to translation hardware in gem5.

```
//input array contains random numbers
checkpoint_start();
index = input[0];
for( int i=0; i<NUM_ACCESS; i++){
    index = input[index];
    input[index] += 0;
    }
checkpoint_end();
```

Listing 2: Sample consistency user code

The address translation hardware reads the address range and sets the corresponding bit in the updated bitmap in extended

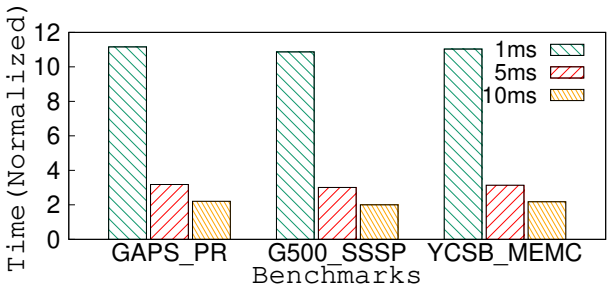


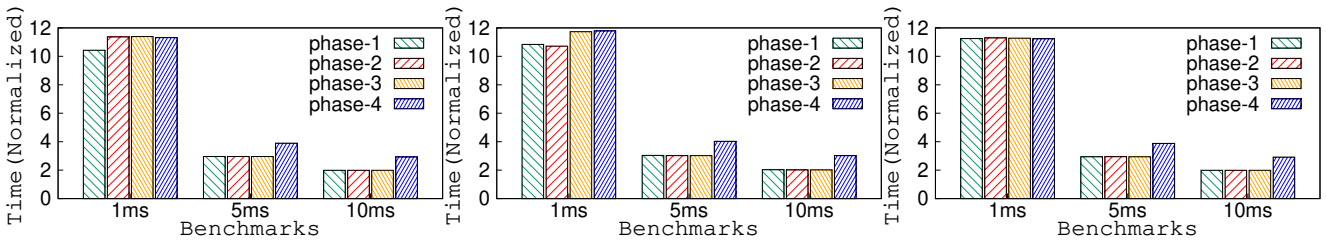
Fig. 3: Influence of memory consistency interval on performance. Y-axis shows normalized execution time with no memory consistency.

TLB if a write happens to the tracked address range, as mentioned in the SSP [32]. In the prototype implementation, the translation hardware generates a memory request to update metadata maintained in the metadata region when a consistency interval ends or a TLB entry eviction happens. We mark the entry as `TLB evicted` in the metadata region. The programming model we used in the prototype is shown in listing 2. The sample code performs random accesses to an array demarcated using start and end intervals. In the prototype, the user marks the failure atomic section (FASE) in her code using `checkpoint_start` and `checkpoint_end`.

The call to `checkpoint_start` enables custom hardware components in the translation and cache controller hardware in gem5 and periodic consistency of memory changes at a fixed interval (specified through a knob in gemOS). For example, setting period consistency intervals as 5ms ensures that at every 5ms, the activities associated with `checkpoint_end` are performed, i.e., gemOS kernel instructs the translation hardware to initiate a memory request to send all updated bitmap in TLBs to the metadata region. The gemOS kernel then calls `clwb` write back instructions to flush all data and metadata updates in hardware caches to NVM. Finally, the gemOS kernel iterates through the metadata region entries and consolidates physical pages corresponding to evicted TLB entries. In SSP [32], the page consolidation is carried out by an OS thread in the background when the reference count for a page drops to zero.

Figure 3 shows the overhead introduced by the SSP ($\sim$11x for GAPS_PR with 1 ms consistency interval) in making the memory state consistent for an application. In this experiment, we replayed memory accesses of applications in Table II using a micro-benchmark. The varied consistency interval to 1, 5 and 10 milliseconds in gemOS. Figure 3 shows the execution time of applications normalized to the execution time with no memory consistency applied (i.e., without `checkpoint_start` and `checkpoint_end`). Having a wider consistency interval (10ms) in comparison to a shorter (1ms) interval reduces the consistency overhead as the number of metadata inspection and `clwb` call reduce with a wider interval. Calling `clwb` write back instruction flush all modified data to memory.

Applications show different memory access patterns during their execution, and the consistency overhead of a scheme such



(a) GAPBS˙PR (b) G500˙SSSP (c) YCSB˙MEMC

Fig. 4: Comparison of memory consistency overhead at different phases of execution. Y-axis shows normalized execution time with no memory consistency.

as SSP varies at different phases in an application's lifetime. We experimented by capturing the consistency overhead at four phases of execution for applications in Table II. We divided each phase to perform 400,000 memory operations from the trace generated using SniP [21]. Figure 4 shows the consistency overhead at different phases of applications under 1, 5, and 10 milliseconds intervals. All three applications showed a marginal increase in overhead at the final phase of execution when consistency is ensured at 5 and 10 ms intervals.

NemOS allows us to carry out a quick evaluation of ideas on hybrid memory hardware, and system software changes as NemOS provides an environment for quick simulation (Figure 1).

IV. Related Work

The ever increasing demand for data processing along with the capacity scaling limitation of DRAM [31] has laid out the path for new byte addressable Non-Volatile Memory technology. NVM provides the opportunity for capacity expansion and data persistence at memory access latency.

NVM for capacity:

When used for capacity, NVM complements DRAM in a hybrid memory setup to provide better read-write latency along with capacity as NVM demonstrates higher read-write latency than DRAM [22], [35]. An ideal goal for a hybrid memory system is to have NVM's capacity with DRAM access latency. A common strategy to attain memory performance close to the ideal hybrid memory system is directing memory access to NVM for capacity and DRAM for latency, i.e., maintaining frequently accessed memory pages in DRAM and others in NVM [36], [47]. Using approaches such as a hardware scheme using the memory controller to monitor access patterns and categorize pages as hot and cold for migration, OS later updates migrated pages' virtual to physical memory mapping by consulting the memory controller [36]. There is also scope for using DRAM as a cache for NVM in hybrid memory managed by software, configuring DRAM and NVM in a flat address and mapping the physical address from NVM to DRAM through page table and TLB extension [25]. The benefit of using simple page migration policies based on heuristics is limited by access to NVM pages overlapping with access to migrated pages [24].

OS also plays a vital role in page migration, and the performance of page migration depends upon page management policies within OS. As existing page migration policy in OS bottlenecks performance, a memory management system supporting huge page migration with multi-threaded and concurrent migration of multiple pages improves performance [46]. OS level page management focusing on access tier of memory along with access locality provides benefits in multi-tiered memory systems [20]. Huge pages offer performance benefits for applications with large memory footprints, and Thermostat [4] proposed a cold page identification scheme, supporting huge pages with a limit on performance degradation. The thermostat uses TLB miss to calculate the page access rate. In addition to software/hardware approaches for managing hybrid memory for better application performance, an analytic model for hybrid memory predicts the hit ratio based on the access profile of workload, the probability of promoting a page from NVM to DRAM on a hit, and the probability of evicting a page [37].

NVM for data persistence:

Using NVM for data persistence has challenges different from using it for capacity. The challenges are due to the volatile nature and the order of writebacks from caches [5]. These challenges necessitate a framework such as memory persistency [34] that allows application developers to reason about the order of writes to NVM and mechanisms to ensure consistency of memory updates [5], either by enclosing updates inside an atomic failure section (FASE) [6], [12], [29], by using specialized memory allocation routines [38], [49], [48] or through ISA and architecture extensions [48]. In FASE schemes, logging is a common approach for consistency, undo, redo logging [41], or justdo logging [19] provides required FASE guarantees by logging either at the hardware level using a fine-grained undo log at cache line granularity [12] or at software by deriving FASE from existing critical sections in code [10].

The state-of-the-art works on NVM's capacity and data persistence goals show the need for an efficient (Figure 1) infrastructure such as NemOS to prototype and evaluate hybrid memory ideas.

V. Conclusion

References

[1] "53 important statistics about how much data is created every day," https://financesonline.com/how-much-data-is-created-every-day/, accessed: 2023-01-18.

[2] "Model-specific register," https://en.wikipedia.org/wiki/Translation_lookaside_buffer, accessed: 2023-01-27.

[3] "Model-specific register," https://wiki.osdev.org/Model_Specific_Registers, accessed: 2023-01-27.

[4] N. Agarwal and T. F. Wenisch, "Thermostat: Application-transparent page management for two-tiered main memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 631–644.

[5] K. Arun, D. Mishra, and B. Panda, "Empirical analysis of architectural primitives for nvram consistency," in *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2021, pp. 172–181.

[6] A. Baldassin, J. Barreto, D. Castro, and P. Romano, "Persistent memory: A survey of programming support and implementations," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–37, 2021.

[7] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[9] M. Cai, C. C. Coats, and J. Huang, "Hoop: efficient hardware-assisted out-of-place update for non-volatile memory," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 584–596.

[10] D. R. Chakrabarti, H.-J. Boehm, and K. Bhandari, "Atlas: Leveraging locks for non-volatile memory consistency," *ACM SIGPLAN Notices*, vol. 49, no. 10, pp. 433–452, 2014.

[11] J. Coburn, A. M. Caulfield, A. Akel, L. M. Grupp, R. K. Gupta, R. Jhala, and S. Swanson, "Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, pp. 105–118, 2011.

[12] N. Cohen, D. T. Aksun, H. Avni, and J. R. Larus, "Fine-grain checkpointing with in-cache-line logging," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 441–454.

[13] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 133–146.

[14] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[15] A. Correia, P. Felber, and P. Ramalhete, "Romulus: Efficient algorithms for persistent transactional memory," in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, 2018, pp. 271–282.

[16] E. Giles, K. Doshi, and P. Varman, "Bridging the programming gap between persistent and volatile memory using wrap," in *Proceedings of the ACM International Conference on Computing Frontiers*, 2013, pp. 1–10.

[17] T. C.-H. Hsu, H. Brügner, I. Roy, K. Keeton, and P. Eugster, "Nvthreads: Practical persistence for multi-threaded applications," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 468–482.

[18] T. Hwang, J. Jung, and Y. Won, "Heapo: Heap-based persistent object store," *ACM Transactions on Storage (TOS)*, vol. 11, no. 1, pp. 1–21, 2014.

[19] J. Izraelevitz, T. Kelly, and A. Kolli, "Failure-atomic persistent memory updates via justdo logging," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 427–442, 2016.

[20] J. Kim, W. Choe, and J. Ahn, "Exploring the design space of page management for {Multi-Tiered} memory systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 715–728.

[21] A. KP, S. Kumar, D. Mishra, and B. Panda, "Snip: an efficient stack tracing framework for multi-threaded programs," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 408–412.

[22] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 2–13.

[23] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[24] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 152–165.

[25] H. Liu, Y. Chen, X. Liao, H. Jin, B. He, L. Zheng, and R. Guo, "Hardware/software cooperative caching for hybrid dram/nvm memory architectures," in *Proceedings of the International Conference on Supercomputing*, 2017, pp. 1–10.

[26] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[27] J. Malicevic, S. Dulloor, N. Sundaram, N. Satish, J. Jackson, and W. Zwaenepoel, "Exploiting nvm in large-scale graph analytics," in *Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads*, 2015, pp. 1–9.

[28] D. Mishra, "Gemos: Bridging the gap between architecture and operating system in computer system education," in *Proceedings of the Workshop on Computer Architecture Education*, 2019, pp. 1–8.

[29] I. Moraru, D. G. Andersen, M. Kaminsky, N. Tolia, P. Ranganathan, and N. Binkert, "Consistent, durable, and safe memory management for byte-addressable non volatile main memory," in *Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems*, 2013, pp. 1–17.

[30] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, vol. 19, pp. 45–74, 2010.

[31] O. Mutlu, "Memory scaling: A systems architecture perspective," in *2013 5th IEEE International Memory Workshop*. IEEE, 2013, pp. 21–25.

[32] Y. Ni, J. Zhao, H. Litz, D. Bittman, and E. L. Miller, "Ssp: Eliminating redundant writes in failure-atomic nvrams via shadow sub-paging," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 836–848.

[33] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar *et al.*, "The case for ramcloud," *Communications of the ACM*, vol. 54, no. 7, pp. 121–130, 2011.

[34] S. Pelley, P. M. Chen, and T. F. Wenisch, "Memory persistency," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 265–276, 2014.

[35] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, 2009, pp. 24–33.

[36] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *Proceedings of the international conference on Supercomputing*, 2011, pp. 85–95.

[37] R. Salkhordeh, O. Mutlu, and H. Asadi, "An analytical model for performance and lifetime estimation of hybrid dram-nvm main memories," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1114–1130, 2019.

[38] D. Schwalb, T. Berning, M. Faust, M. Dreseler, and H. Plattner, "nvm malloc: Memory allocation for nvram." *Adms@ Vldb*, vol. 15, pp. 61–72, 2015.

[39] S. Shin, S. K. Tirukkovalluri, J. Tuck, and Y. Solihin, "Proteus: A flexible and fast software supported hardware logging approach for nvm," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 178–190.

[40] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Improving phase change memory performance with data content aware access," in *Proceedings of the 2020 ACM SIGPLAN International Symposium on Memory Management*, 2020, pp. 30–47.

[41] H. Wan, Y. Lu, Y. Xu, and J. Shu, "Empirical study of redo and undo logging in persistent memory," in *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2016, pp. 1–6.

[42] B. Wang, J. Tang, R. Zhang, W. Ding, S. Liu, and D. Qi, "Energy-efficient data caching framework for spark in hybrid dram/nvm memory architectures," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2019, pp. 305–312.

[43] C. Wang, H. Cui, T. Cao, J. Zigman, H. Volos, O. Mutlu, F. Lv, X. Feng, and G. H. Xu, "Panthera: Holistic memory management for big data processing over hybrid memories," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 347–362.

[44] Z. Wu, K. Lu, A. Nisbet, W. Zhang, and M. Luján, "Pmthreads: Persistent memory threads harnessing versioned shadow copies," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 623–637.

[45] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *IEEE design & test of computers*, vol. 28, no. 1, pp. 44–51, 2011.

[46] Z. Yan, D. Lustig, D. Nellans, and A. Bhattacharjee, "Nimble page management for tiered memory systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 331–345.

[47] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu, "Row buffer locality aware caching policies for hybrid memories," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE, 2012, pp. 337–344.

[48] J. Zhao, S. Li, D. H. Yoon, Y. Xie, and N. P. Jouppi, "Kiln: Closing the performance gap between systems with and without persistence support," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 421–432.

[49] P. Zuo, Y. Hua, and J. Wu, "Write-optimized and high-performance hashing index scheme for persistent memory," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 461–476.