Kindle: A Framework to Explore Architecture-OS Design Ideas in Hybrid Memory Systems

Arun KP[1] Debadatta Mishra[2]

***Abstract*—Computers with hybrid memory can offer the benefits of both DRAM and NVM technologies, where NVM provides higher capacity and data persistence while DRAM provides better performance. One of the primary objectives in hybrid memory systems is to design mechanisms and policies to take advantage of the underlying memory technologies to improve overall system performance.**

In this paper, we introduce an open-source framework, Kindle, based on gem5 and gemOS, to explore and prototype research ideas in hybrid memory setups in a holistic manner crossing the hardware-software boundaries. Kindle provides a quick way to study hybrid memory systems as it reduces simulated instructions by more than 2× compared to a Linux setup. We demonstrate the utility of Kindle by prototyping state-of-the-art solutions for hybrid memory systems to show that new insights can be derived using the proposed OS-HW simulation setup.

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

I. Introduction

Non-volatile memory (NVM) provides high memory capacity with reduced energy cost compared to volatile memory (DRAM). However, the increased read/write latency of NVM compared to DRAM [13] raises performance concerns when NVM is used as a drop-in replacement for DRAM. Therefore, a more attractive memory organization can be a hybrid memory system with DRAM and NVM, designed to complement each other for reduced energy cost, persistence, and better performance. A hybrid memory system allows placing data of applications in NVM and/or DRAM, enabling memory managers to coordinate allocations for high memory capacity with low access latency. For example, one can place frequently accessed hot memory pages in DRAM and migrate cold pages to NVM. Several existing works on hybrid memory systems propose efficient access for large workloads by intelligently placing data across the NVM and DRAM [16], [25]–[27], reducing energy consumption by migrating data across the DRAM and NVM tiers [14], [21], [22], [30]. Another usage of NVM is to use it as an alternative to external storage hardware (HDD or SSD) for data persistence. In this usage scenario, application developers need to incorporate consistency and durability semantics into their design. Existing research works attempt to address the consistency issues by designing solutions such as persistent object store [10], lock-based failure atomicity for multi-threaded programs [4], [9], [29], and hardware and/or software memory consistency mechanisms [1], [5], [8], [12], [20].

The nature of problems and proposed solutions for hybrid memory systems require an infrastructure allowing *extension, validation, and evaluation* of the complete system stack. Architecture simulators capable of full-system simulation, such as gem5 [17], provide a platform for prototyping ideas crossing the hardware-software boundaries. However, using gem5-Linux full-system simulation setup to explore new ideas in hybrid memory systems 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) [11], [28], their integration is non-trivial (in gem5), especially considering constantly evolving hardware and OS support.

[1,2]CSE Department, Indian Institute of Technology, Kanpur, India kparun@cse.iitk.ac.in

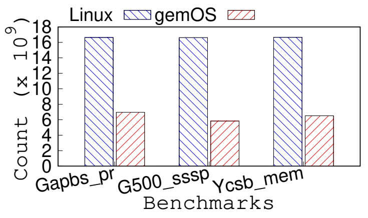

Fig. 1: Comparison of simulated instructions with gem5.

Second, Linux kernel is heavyweight, whereby the OS functions and services can consume a significant part of a simulation which may not be desirable for quick prototyping of design ideas. As we show in Figure 1, for Gapbs_pr [2], G500_sssp [19], and Ycsb_memc [7] benchmarks, Linux executes more than 2× instructions compared to gemOS [18], the OS component of Kindle. *Third*, designing PoCs in Linux requires significant understanding and changes in the Linux memory management subsystem, which has non-trivial complexity. Therefore, a full-system simulation framework with a lightweight OS, exposing hybrid memory to applications is essential. We show that such a platform allows exposing design challenges and unearthing new performance insights in existing state-of-the-art hybrid memory systems using prototype implementations of two well-established research ideas (Section III).

In this paper, we showcase a hybrid memory framework, Kindle, built using a lightweight OS (i.e., *gemOS*) as its base, to explore solutions *crossing architecture and operating system boundaries*. Kindle 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 extended gem5 to expose hybrid memory to gemOS and modified the gemOS system call APIs to allow user applications to allocate memory in DRAM and/or NVM using memory allocation API. Kindle aids in quickly realizing complex design goals and providing new insights into existing schemes, as we show in two prototype implementations of state-of-the-art hardware-software hybrid memory schemes, SSP [20] and HSCC [15]. SSP handles the memory consistency requirement of NVM by using shadow sub-paging. HSCC provides memory capacity by arranging DRAM and NVM in a flat address space and managing DRAM as a cache to NVM using a hardware/software cooperative caching mechanism. These prototype implementations show the capability of Kindle to quickly validate ideas in hybrid memory systems and collect additional results. Through the SSP prototype, Kindle provides new insights into the impact of consistency interval on the performance overhead of SSP, showing that a wider consistency interval reduces the overhead. HSCC prototype with Kindle provides insights into the migration overhead due to the OS activities, which was not shown in the original work as the evaluation framework did not have an OS component.

The main contributions of this paper are:

- An efficient and lightweight simulation framework with full-system simulation capability for hybrid memory systems.

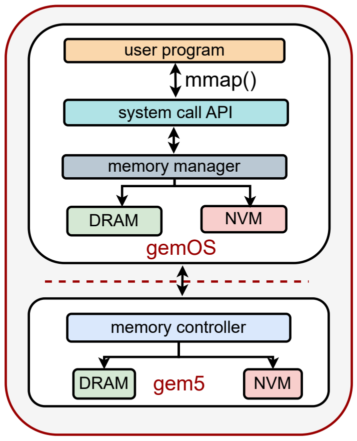

Fig. 2: Schematic diagram of Kindle hybrid memory system.

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	3GB DRAM + 2GB NVM
‡PCM timing parameters based on [24]	

- Prototype implementation of two state-of-the-art proposals followed by an experimental evaluation to establish new insights and conclusions.

II. Design and Implementation

Kindle enables applications to allocate memory from both NVM and DRAM. Figure 2 shows a high-level schematic diagram of Kindle. It consists of two components—a cycle-accurate architecture simulator gem5 [3], [17] and a lightweight operating system (gemOS [18]) specialized for running on gem5. Kindle allows researchers to investigate the OS, architecture changes independently, and also the intriguing interactions between them in a hybrid memory system.

Kindle exposes DRAM and NVM to user-space applications through the mmap system call. An application making mmap can differentiate memory requests for NVM from DRAM by passing an additional flag MAP_NVM in mmap() system call as shown in the sample code Listing 1.

```
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

For provisioning memory from the two memory hardware, Kindle partitions the physical memory address range between NVM and DRAM, and inserts corresponding entries in BIOS e280 of gem5. It then configures the NVM memory controller interface in gem5 with specifications mentioned in Table I. Kindle provides a hybrid memory system in flat address mode, allowing the OS to expose DRAM and NVM to applications.

TABLE II: Benchmark Details

Benchmark	Total Ops	read %	write %
Gapbs_pr [2]	10,000,000	77	23
G500_sssp [19]	10,000,000	68	32
Ycsb_mem [7]	10,000,000	71	29

Kindle uses gemOS [18] as the OS component since it allows a quick full-system simulation on gem5 compared to Linux. In gemOS, we modify the process address space implementation by adding new virtual memory areas (VMA) to denote the memory type based on the request from an application. We also modify the physical memory allocation routines in gemOS to assign physical page frames from NVM or DRAM based on the memory type of the VMA at different page allocation scenarios such as demand paging and lazy allocation. Kindle source code is currently available at https://anonymous.4open.science/r/NemOS-F0C0/

III. Results

In this section, we show the benefit of using Kindle to prototype and perform initial evaluation of existing or new ideas on a hybrid memory system. We implemented two research ideas with Kindle—*(i)* shadow sub-paging (SSP) [20] to ensure consistent memory state of an application in NVM by employing shadow paging [6], [20] at sub-page cache line granularity, *(ii)* a hardware-software co-operative caching mechanism, HSCC [15],for managing DRAM as cache to NVM. In these studies, we replayed read-write memory access traces for applications in Table II using a micro-benchmark to measure the performance. For the experiments, Gem5 was configured with Intel 64-bit in-order CPU at 3GHz with 32KB L1, 512KB L2 and 2MB/core LLC.

We show the implementation approach we followed in Kindle for these prototypes and the initial results below.

Prototype 1: Shadow Sub-Paging (SSP)

SSP [20] provides memory consistency in NVM. It ensures consistency of memory modifications by maintaining a copy of unmodified data at cache line granularity. SSP allocates two physical pages for each virtual page and uses a remapping scheme at the cache controller hardware to route modifications at cache line granularity to alternate physical pages. SSP also extends the TLB by adding extra fields per entry to capture the supplementary physical page mapping and bitmaps (updated, current) to track the page containing the latest modification. SSP proposes a background OS thread to consolidate two physical pages but leaves out the detailed implementation and evaluation of the *the consolidation* aspects in the paper.

In Kindle, we allocate the additional physical page in the page allocation routine call in gemOS. The original and the extra page addresses and the bitmap values (commit, current) are recorded in a metadata area (i.e., SSP cache). We extend the page walker hardware in gem5 to fill fields in the TLB during an address translation on TLB miss. 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 use Model Specific Registers (MSRs) to communicate the virtual address range corresponding to NVM allocation to hardware. We also use MSR to pass the base address of SSP cache to translation hardware in gem5. The address translation hardware checks the address range and sets the corresponding bit in the *updated* bitmap in TLB if a write happens to the NVM address range. The translation hardware generates a memory request to modify metadata in SSP

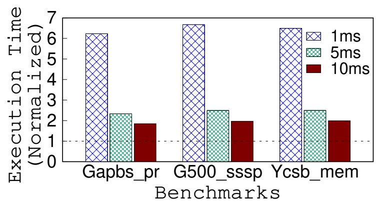

Fig. 3: Influence of memory consistency interval on performance. Y-axis shows normalized execution time with no memory consistency.

cache when a consistency interval ends, or a TLB entry eviction happens. We mark the entry as *TLB evicted* in the SSP cache.

We use a programming model in which the user demarcates the failure atomic section (FASE) in code using `checkpoint_start` and `checkpoint_end` calls. `checkpoint_start` enables custom hardware components in the address translation and cache controller hardware in gem5. A consistency interval of choice is set in gemOS. For example, setting consistency intervals as 5ms ensures that at every 5ms interval ends, and activities associated with `checkpoint_end` are performed, i.e., gemOS kernel instructs the address translation hardware to initiate a memory request to send all modified 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. Physical page consolidation happens asynchronously; a thread periodically calls page consolidation routine to merge pages corresponding to evicted TLB entries by inspecting the SSP cache entries in gemOS.

Figure 3 shows the overhead introduced by SSP in making the memory state of applications consistent. This study used consistency intervals of 1, 5, and 10 milliseconds. The page consolidation thread interval is fixed to 1 ms as a lower interval would result in higher consolidation overhead. Figure 3 shows the execution time of applications normalized to the execution time with no memory consistency applied. Having a wider consistency interval (10ms) reduces the consistency overhead as the number of metadata inspections and `clwb` calls to writeback cache lines reduce with a wider consistency interval. All applications in Figure 3 show an average $\sim 3\times$ reduction in memory consistency overhead with 1ms compared to a 10ms consistency interval.

Kindle provides an easy way to extend studies such as the influence of consistency interval on the application performance, and it also allows carrying out additional studies on the influence of page consolidation thread invocation frequency on an application by varying the thread time interval, which is not explored in original SSP proposal.

Prototype 2: Hardware/Software Cooperative Caching

HSCC [15] aims to utilize high NVM capacity in a hybrid memory system for improved system performance. HSCC maintains NVM and DRAM in a flat address space and uses DRAM as a cache managed by OS in a hardware/software cooperative manner. HSCC tracks access count of NVM pages to select candidate pages for migration to DRAM and maintains an NVM-to-DRAM page mapping after migration. HSCC extends the page table and TLB for handling NVM to DRAM remapping and tracking the access count of NVM pages. NVM pages with an access count exceeding a specific fetch threshold value in a migration interval are selected for migrating to DRAM.

HSCC extends page table entry (PTE) to record DRAM and NVM page frame numbers, using 96 bits (12 bytes) for PTE as opposed to 64 bits in conventional x86-64 systems. In this case, the last level page table in HSCC can only map 341 pages (i.e., 4KB/12B), leaving 171 pages unmapped in a 2MB address region. In our implementation, we have designed NVM to DRAM mapping in a lookup table to avoid the previously mentioned PTE size issue. The mapping table entries can be looked up using both DRAM and NVM page frame numbers as an offset. We also maintain a clean and dirty list of DRAM pages, updated at the start of each migration interval of 31.25 ms (equivalent to 10^8 cycles mentioned in the HSCC paper). The migration activity inspects the page access count maintained in PTEs corresponding to NVM pages and migrates the pages to DRAM cache if the count exceeds the fetch threshold. The page access count is also maintained in TLB and is incremented if the data access misses in the LLC. The access count in TLB is written out to PTE on TLB eviction or once during the translation in a migration interval. We have not incorporated dynamic fetch threshold adjustment in our implementation and have fixed the threshold to static values. The page access count in PTEs is reset in each migration interval to ensure that NVM pages from the most recent interval are considered for migration.

TABLE III: Number of Pages Migrated

Benchmark	**Th-5**	**Th-25**	**Th-50**
Gapbs_pr	353	270	113
G500_sssp	4604	1463	1340
Ycsb_mem	22788	1746	224

We investigated the overhead of migration-related activities performed by OS on an application's execution time. The migration activities include inspecting page access count in PTEs of NVM mapped pages by walking the page table, allocating DRAM pages for migration from the free, clean, or dirty list of DRAM pages, and copying back DRAM page to NVM if the selected candidate page is from dirty list. The migration activity then flushes cache lines corresponding to the NVM page under migration before copying data from NVM to DRAM and invalidates the corresponding TLB entry. The page access count in all PTEs is reset, and corresponding TLB entries are invalidated in a migration activity to ensure that page accesses for the most recent interval are considered for migrations.

Figure 4 shows the overhead of migration activities performed by OS. The figure shows the execution time of applications with migration (i.e., performing hardware and OS migration activities) normalized to the execution time without OS migration activities (i.e., performing only hardware migration activities) under different fetch thresholds; the fetch threshold decides the number of candidate NVM pages for migration. Two factors majorly influence the execution time of an application with migration: the overhead of activities performed by OS as part of the migration and the benefit in memory access time after migrating pages to DRAM. All applications in Figure 4 show migration overhead due to OS activities, and a higher value indicates that the overhead of activities performed by OS as part of the migration overshadows the benefit in memory access time after migrating pages to DRAM. Gapbs_pr shows the minimum overhead, indicating that Gapbs_pr has the maximum benefit in memory access time after migrating pages to DRAM. For all applications, the migration overhead reduces with an increase in the fetch threshold as the number of candidate pages migrated reduces with an increase in the threshold as shown in Table III; hence, the overhead of OS activity reduces. For example, Ycsb_mem showed $\sim 13\times$ and $\sim 101\times$ reduction in the number of pages migrated for Th-25 and Th-50 compared to Th-5 respectively.

HSCC in original work used ZSim [23], a user-level simulator, and Zsim does not support OS-level simulation [15]; hence, it can not account for the overhead of OS migration activities, such as

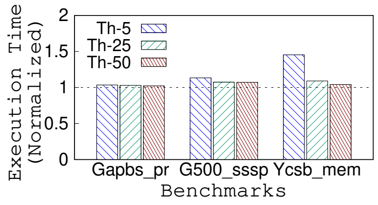

Fig. 4: Influence of DRAM fetch threshold on performance. Y-axis shows normalized execution time with no DRAM migration.

copying pages from NVM to DRAM. As Kindle is based on a full-system simulator, it allows studying the overhead of OS activities on page migration, the influence of other OS activities such as context switches, and the effect of cache pollution due to OS activities on migration. on the contrary, user-level simulators like ZSim miss out on such insights about OS interactions in hybrid memory systems. Kindle also allows separately investigating performance overhead due to hardware and OS activities, as shown in Figure 4 for OS migration activities.

Kindle allows researchers to carry out quick evaluation of ideas crossing hardware-software layers on hybrid memory systems, as we show in two prototype implementations.

IV. Conclusion

The hybrid memory system provides the benefit of both DRAM and NVM technology. NVM offers high capacity and data persistence, and DRAM delivers better performance. The existing infrastructure for hybrid memory exploration crossing architecture-OS boundary using simulators such as gem5 is limited by the complexity of integrating NVM support in Linux for gem5 and the simulation overhead of Linux due to OS services and functions running.

In this paper, we introduced an open-source framework, Kindle, based on gem5 and gemOS for hybrid memory exploration in architecture and operating systems. Kindle provides a quick way to study and prototype solutions for hybrid memory systems as it reduced simulated instructions by more than 2× on average compared to Linux. We showed prototype implementation of state-of-the-art hardware-software hybrid memory schemes, SSP [20] and HSCC [15], using Kindle to demonstrate its efficacy in realizing complex design goals and analyzing new insights.

References

[1] K. Arun, D. Mishra, and B. Panda, "Empirical analysis of architectural primitives for nvram consistency," in *Proceedings of HiPC*. IEEE, 2021, pp. 172–181.

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

[3] 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.

[4] 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.

[5] N. Cohen, D. T. Aksun, H. Avni, and J. R. Larus, "Fine-grain checkpointing with in-cache-line logging," in *Proceedings of ASPLOS*, 2019, pp. 441–454.

[6] 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 Symposium on Operating systems principles*, 2009, pp. 133–146.

[7] 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.

[8] A. Correia, P. Felber, and P. Ramalhete, "Romulus: Efficient algorithms for persistent transactional memory," in *Proceedings of SPAA*, 2018, pp. 271–282.

[9] 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.

[10] 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.

[11] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[12] J. Jeong, C. H. Park, J. Huh, and S. Maeng, "Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 520–532.

[13] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *Proceedings of ISCA*, 2009, pp. 2–13.

[14] 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.

[15] 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.

[16] H. Liu, R. Liu, X. Liao, H. Jin, B. He, and Y. Zhang, "Object-level memory allocation and migration in hybrid memory systems," *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1401–1413, 2020.

[17] 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.

[18] 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.

[19] 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.

[20] 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 MICRO*, 2019, pp. 836–848.

[21] H. Park, S. Yoo, and S. Lee, "Power management of hybrid dram/pram-based main memory," in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 59–64.

[22] B. Peng, Y. Dong, J. Yao, F. Wu, and H. Guan, "Flexhm: A practical system for heterogeneous memory with flexible and efficient performance optimizations," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 1, pp. 1–26, 2022.

[23] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

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

[25] 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 *Proceedings of International Conference on High Performance Computing and Communications*. IEEE, 2019, pp. 305–312.

[26] X. Wang, H. Liu, X. Liao, J. Chen, H. Jin, Y. Zhang, L. Zheng, B. He, and S. Jiang, "Supporting superpages and lightweight page migration in hybrid memory systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 2, pp. 1–26, 2019.

[27] W. Wei, D. Jiang, S. A. McKee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*. IEEE, 2015, pp. 163–173.

[28] M. Weiland, H. Brunst, T. Quintino, N. Johnson, O. Iffrig, S. Smart, C. Herold, A. Bonanni, A. Jackson, and M. Parsons, "An early evaluation of intel's optane dc persistent memory module and its impact on high-performance scientific applications," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2019, pp. 1–19.

[29] Z. Wu, K. Lu, A. Nisbet, W. Zhang, and M. Luján, "Pmthreads: Persistent memory threads harnessing versioned shadow copies," in *Proceedings of PLDI*, 2020, pp. 623–637.

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