Persistent Memory Research in the Post-Optane Era

Peter Desnoyers
Northeastern University
p.desnoyers@northeastern.edu

Ian Adams
Intel Corporation
ian.f.adams@intel.com

Tyler Estro
Stony Brook University
testro@cs.stonybrook.edu

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

Geoff Kuenning
Harvey Mudd College
geoff@cs.hmc.edu

Mike Mesnier
Intel Corporation
michael.mesnier@intel.com

Carl Waldspurger
Carl Waldspurger Consulting
carl@waldspurger.org

Avani Wildani
Emory University and Cloudflare
agadani@gmail.com

Erez Zadok
Stony Brook University
ezk@fsl.cs.sunysb.edu

ABSTRACT

After over a decade of researcher anticipation for the arrival of persistent memory (PMem), the first shipments of 3D XPoint-based Intel Optane Memory in 2019 were quickly followed by its cancellation in 2022. Was this another case of an idea quickly fading from future to past tense, relegating work in this area to the graveyard of failed technologies?

The recently introduced Compute Express Link (CXL) may offer a path forward, with its persistent memory profile offering a universal PMem attachment point. Yet new technologies for memory-speed persistence seem years off, and may never become competitive with evolving DRAM and flash speeds. Without persistent memory itself, is future PMem research doomed? We offer two arguments for why reports of the death of PMem research are greatly exaggerated.

First, the bulk of persistent-memory research has not in fact addressed memory persistence, but rather in-memory crash consistency, which was never an issue in prior systems where CPUs could not observe post-crash memory states. CXL memory pooling allows multiple hosts to share a single memory, all in different failure domains, raising crash-consistency issues even with volatile memory.

Second, we believe CXL necessitates a "disaggregation" of PMem research. Most work to date assumed a single technology and set of features, *i.e.*, speed, byte addressability, and CPU load/store access. With an open interface allowing new topologies and diverse PMem technologies, we argue for the need to examine these features individually and in combination.

While one form of PMem may have been canceled, we argue that the research problems it raised not only remain relevant but have expanded in a CXL-based future.

CCS CONCEPTS

• **Information systems** → **Storage class memory**.

KEYWORDS

Persistent memory, PMem, 3D XPoint, Optane, CXL.

ACM Reference Format:
Peter Desnoyers, Ian Adams, Tyler Estro, Anshul Gandhi, Geoff Kuenning, Mike Mesnier, Carl Waldspurger, Avani Wildani, and Erez Zadok. 2023. Persistent Memory Research in the Post-Optane Era. In *1st Workshop on Disruptive Memory Systems (DIMES '23), October 23, 2023, Koblenz, Germany*. ACM, New York, NY, USA, 8 pages. *https://doi.org/10.1145/3609308.3625268*

1 INTRODUCTION

As CPU processing speeds and core counts continue to grow, so too do the I/O speeds needed to feed data to ever-faster CPUs, with some workloads (*e.g.*, indexes, Bloom filters) being particularly sensitive to I/O latency. Yet as storage latencies drop into the 10s of microseconds, improvements in device speed begin to be overshadowed by software delays and overheads in the OS storage stack. While various strategies have been used to reduce these overheads [54], persistent memory allows them to be bypassed entirely for most accesses.

In recent years, the availability of persistent memory (PMem) has spurred a flurry of research [5, 12, 18, 22, 25, 26, 28, 29, 37, 47, 53]. PMem's unique properties encouraged research in the storage community and beyond:

DIMES '23, October 23, 2023, Koblenz, Germany

ACM ISBN 979-8-4007-0300-3/23/10…$15.00
https://doi.org/10.1145/3609308.3625268

algorithms [6, 10, 12], compilers [23, 32, 33], data structures [18, 28, 29, 37], file systems [25, 31, 48], key-value stores [5, 26, 55], operating systems [3, 27, 40, 50], and even non-systems areas have been affected. Industry efforts produced the Storage Networking Industry Association (SNIA) programming model [46] and the PMDK [21] library.

When Intel canceled its 3D XPoint-based Optane product line [20], researchers were suddenly left wondering whether persistent-memory technologies had any future. Yet behind the headlines, both Micron [36] and Intel [19] embraced the industry *Compute Express Link* (CXL) [8] standard as their future direction for persistent and hierarchical memory. Others have also begun to discuss the lessons learned and outline future prospects for PMem [2, 15, 24, 45].

Persistent memory has in effect taken one step backwards, losing a storage technology, and one tentative step forwards, gaining an alternate, arguably superior interface. This new interface is not only vendor-independent but multipurpose, with use cases (*e.g.*, cache-coherent GPU-to-host access, CXL memory pooling) that are likely to ensure its viability independent of market demands for persistent memory. Before CXL, only CPU vendors could consider integrating persistent memory into a system; with CXL, even academic researchers can design and deploy FPGA-based PMem prototypes. But should they?

Answering this question requires examining the defining characteristics of PMem in more detail: (a) persistence, (b) byte addressability, and (c) direct access via CPU load/store instructions.

Byte addressability reduces the cost of small accesses; load/store access dramatically accelerates some I/O tasks, providing direct user-space access without kernel intervention. In addition to those features, Optane provided near-DRAM speed and better-than-DRAM cost per bit.

Given multiple potential persistent technologies and methods of access, we believe it is important to consider PMem's features—including persistence itself—individually as well as in various combinations. Is load/store access important, or would user-space byte-granular access via an RDMA-like mechanism provide similar performance? Which Optane performance improvements require near-DRAM speed, vs. those that are enabled by merely better-than-NVMe performance? What about the non-persistent case with CXL memory pooling, and does multi-host access across multiple failure domains pose the same challenges as single-host PMem, or new ones? Finally, how important is price, and in particular would PMem be viable if it were no cheaper than DRAM?[1]

[1]We note that "cheaper than DRAM" is a vague target, as high-density DIMMs carry a cost premium of up to 10× over lower densities.

2 WHAT IS PERSISTENT MEMORY?

By *persistent Memory* or *PMem* we refer to media with byte-addressable access (*e.g.*, via hardware access at cache-line granularity)via CPU load/store instructions, with coherent caching, but with the persistence properties of storage. PMem supports direct memory access (DMA) by other devices, and is fast enough to warrant waiting for a load instruction rather than context-switching to another thread as is done with slower storage (*e.g.*, NAND Flash) [42]. Software support (*e.g.*, via libraries conforming to the SNIA NVM Programming Model [46]) allows PMem implementations using natively persistent media (*e.g.*, 3D XPoint) or natively volatile (*e.g.*, DRAM) devices with hardware support for persistence in the event of power loss.

Additional higher-level functions supported by the SNIA model include: (a) PMem-aware file systems—*e.g.*, `ext4` with the DAX option—which provide naming, access control, and the ability to map persistent data into the virtual address space. (b) Library APIs that allow applications to discover whether store instructions are considered persistent as soon as they are visible to other threads, or if flush operations are required to guarantee that stores have been committed. (c) Software mechanisms to detect failures unique to PMem, *e.g.*, an incomplete *flush on fail* execution after a power failure.

A Brief timeline of PMem products. Battery-backed RAM has a long history of use for RAID stripe buffers [16], and before that magnetic core memory was persistent across power loss [2] [39]. However persistent memory as we know it can be traced to shortly after 2000—both conceptual work on *storage-class memory* [11] and products in the form of *NVDIMM-N* [49], DRAM DIMMs with energy storage and flash backup that allow memory contents to last across power loss. NVDIMMs used standard memory sockets, but required platform support for power-loss notification. They were shipped by several companies for nearly a decade [49], but because they were much more expensive than conventional DRAM, they were rarely if ever deployed as entire storage systems.

Later in that decade, emerging technologies such as Phase Change Memory [11] resulted in sustained research interest in persistent memory, accelerated by Intel and Micron's announcement of 3D XPoint memory and Intel's Optane plans. In 2019, Intel began shipping Optane memory devices, using the *DDR-T* variant of standard memory sockets. Optane had much higher capacity and lower cost per gigabyte than NVDIMM-Ns, since it leveraged the native persistence of 3D XPoint. However, it had lower performance—around 3× the latency of DRAM, with bandwidth somewhat lower for read and much lower for write [22]. Since Optane greatly

[2]Due to cost, this persistence was rarely used for anything except boot loaders.

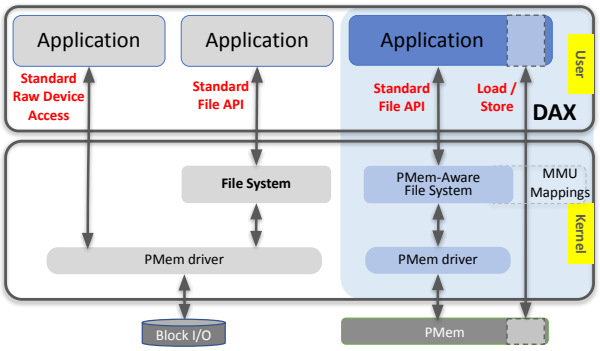

Figure 1: Block and PMem data paths. Direct access (DAX, upper right) incurs no software overhead.

outperformed NAND Flash, its primary use case was as a persistent write cache for very large data structures such as in-memory databases. Due to its high capacity and (arguably) lower cost per gigabyte, Optane was also considered as a potential second tier of volatile memory, cached by DRAM.

Micron stopped production of 3D XPoint in 2021, and in 2022 Intel discontinued their Optane product line. As of this writing no other high-capacity PMem products are available commercially, and no future 3D XPoint products are expected. The number of companies shipping NVDIMM-Ns has declined recently, although they are still available in capacities of around 16–32GB.

PMem benefits. Figure 1 illustrates the difference between the common *Block I/O* data path and the *PMem* data path. The rightmost application in the figure uses standard file APIs to open and memory-map a PMem file; all PMem I/O is then able to use the standard load/store model. This is made possible by the *DAX* (Direct Access) feature in specific file systems, allowing `mmap` to directly map underlying address-space-resident memory, along with hardware persistence support, *e.g.*, enabled by appropriate PMDK library operations.

These accesses are far more efficient than access via the block I/O data path. In the PMem case individual instructions retrieve data from cache, while the memory controller issues a single read to the memory device for each cache line accessed. In contrast, block access typically requires user/kernel transitions for each access, multiple PCIe transactions for data and descriptor transfers and doorbell register writes, and a significant in-kernel software path[3].

The performance difference is even larger for small accesses, as block I/Os are typically rounded up to a 4 KB block size, while PMem is accessed at cache-line granularity. Data structures can be mapped into application memory as shown by the rightmost arrow in the figure, and then accessed directly, without needing to copy data into DRAM. This ability to access persistent data in place is one of the major benefits of PMem [44].

PMem challenges. Systems supporting PMem have two levels of store persistence, as per the SNIA Programming model. The most common level, avoiding the need for more expensive platform logic, requires applications to flush stores explicitly to ensure persistence. While storage has always worked this way, programmers are not used to having to flush memory stores; this introduces new software complexities. The problem is exacerbated by existing code that assumes block writes are atomic, which allows atomic updates of large data structures. Libraries like PMDK [44] normally handle some of the complex logic around flushing and transactions, but significant work may be needed to adapt existing software [34].

The second level of persistence is provided by platforms that automatically flush all CPU caches to PMem on power loss or system crash. This relieves the software from that responsibility. But since this feature is not guaranteed to exist on every platform with PMem, the software must typically handle both cases, so no complexity is avoided.

The lack of native language support for PMem is also problematic, requiring libraries to use non-idiomatic constructs like preprocessor macros to support PMem, adding to debugging complexity. Although it is possible that idiomatic, usable PMem extensions to high-level languages will emerge in the future, such improvements typically arrive only slowly. The fact that software must be modified to use PMem at all is itself a problem, since software changes are expensive. To mitigate this, a number of ways to leverage PMem transparently have emerged. Ideas such as Whole System Persistence [38] and Whole Process Persistence [17] can leverage the benefits of PMem's in-place access without application modification. In many cases language support for transparent use of PMem may be difficult—existing code often assumes that data structures are assembled in ephemeral buffers, never visible outside a limited range of code; the lack of buffering in PMem accesses may necessitate significant changes in strategy.

Finally, a consistent pain point for PMem has been that it is *directly attached* to a single host; if the host goes down, access to that persistent data is lost. This differs from other storage systems that can be attached on the network and made accessible from multiple hosts (*e.g.*, NAS, SAN). Several solutions to replicate PMem in software have been implemented [13, 14, 52], but they increase complexity further.

[3]User-space access through SPDK [54] can reduce the software overhead of this process, but the PCIe overhead remains.

3 CXL: A NEW PMEM INTERFACE

CPU changes were needed to efficiently support new PMem products. Modifications to the DDR protocol supported variable timing [1] and power-loss notification. For performance [7, 9], new instructions and memory controller designs [43] were needed to quickly and reliably persist data.

In 2019, the first version of the *Compute Express Link* (CXL) specification was released by a consortium of over 250 companies. As of November 2020, version 2.0 of the CXL specification contains first-class support for PMem, rather than adding it as an afterthought as was done for DDR protocols.

CXL 1.1 and 2.0 run over PCIe 5, while CXL 3.0 uses PCIe 6, introducing three new protocols:

- **CXL.io**: PCIe functionality, including device enumeration and PCIe-style data transfers.
- **CXL.cache**: allows device caches as part of the CPU cache-coherency domain.
- **CXL.mem**: allows hosts to access device-attached memory with cache-coherent loads and stores.

A CXL *Type 3 Memory Device*, built using the CXL.io and CXL.mem protocols, allows OSes to have a single, generic driver supporting both volatile memory and PMem, even on the same device [8, 24]. Moreover, while previous PMem devices required explicit CPU support, CXL allows independent vendors and even researchers to build a wide variety of PMem devices. CXL incorporates the lessons learned from prior PMem products, and in many cases allows binary compatibility for applications developed for NVDIMM [49] and Optane devices.

With CXL, *memory pooling*, supported by the Multi-Headed Device (MHD) model in CXL 3.0, allows multiple hosts to access memory presented by a single device. This provides the ability to disaggregate both volatile and persistent memory, and to dynamically assign it to different hosts over time [30], as shown in Figure 2, providing a separate "memory appliance" with its own power, failure domain, and reliability characteristics. As an example, Pond [30] uses a custom controller to provide single-host cache coherence, coupled with dynamic access control assigning each memory region to a single host at a time.

Such memory pooling offers an opportunity for application-transparent data replication across failure-domain boundaries. This in turn addresses a key limitation of prior PMem configurations, where such replication required explicit application support, typically requiring slower software intervention rather than being implemented in hardware.

Pond and similar approaches allow non-concurrent sharing of memory where, for example, a new host may take ownership of a memory region after a crash; additional features allow *concurrent* sharing of memory regions by multiple hosts, with either non-coherent access (requiring explicit flush operations) or optionally with full cache coherency across hosts.

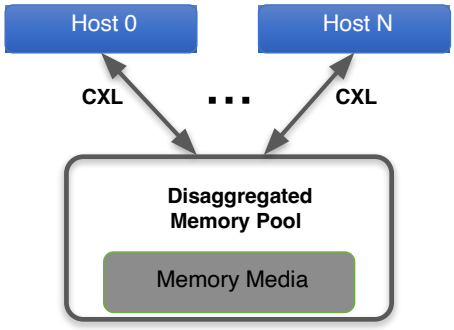

Figure 2: Basic CXL memory pooling.

Memory pooling and sharing also introduce new security concerns. The CXL specification supports low-level encryption for memory and interconnect links [8, Section 11.0]; further research is needed in this area.

4 RESEARCH GOING FORWARD

Although other persistent memory technologies predated it, 3D XPoint was perhaps the first solid-state technology to offer both cost and performance midway between contemporary main memory and block storage technologies—the cheaper-than-DRAM, faster-than-NAND flash window. Since this "storage-class memory" window is a moving target, the emergence of a new and competitive persistent-memory technology is heavily dependent on progress at both ends of this window—progress driven by enormous investments based on the size of these markets.

As a result, it is entirely possible that we will not see a solid-state memory technology arise that directly replaces 3D XPoint. Yet we argue that in the CXL era, persistent-memory research remains just as relevant, for two reasons:

- Hybrid persistent memory [41]. Even in the absence of new technologies, hybrid strategies combining DRAM, flash, and energy storage will enable future CXL-attached persistent-memory systems at varying price and performance points.
- Multi-host consistency. PMem raised the new (at the time) problem of crash consistency for memory; in previous systems memory contents were lost on power failure, and the CPU could never observe crash-inconsistent memory states. CXL memory pooling allows memory to be observed from multiple independent failure domains, leading to similar challenges

#	Example Storage / Memory Technologies	Coherent	Byte-addressable	Persistent
1	Volatile RAM disk	✘	✘	✘
2	Conventional HDD, SSD	✘	✘	✔
3	Incoherent load/store to PCIe address space	✘	✔	✘
4	Byte-addressable I/O device (*e.g.*, object storage)	✘	✔	✔
5	Memory pooling	✔	✔	✘
6	Conventional PMem use cases, including NVDIMM	✔	✔	✔

Table 1: A taxonomy of storage technologies that may support coherency, byte-addressability, and persistence.

even in the absence of persistence, while doing so under a range of topologies and speeds.

Changes brought about by CXL. Historically (*i.e.*, before PMem) memory researchers have not had to worry about issues like data persistence, durability, and availability; these were issues specific to storage systems. PMem changed this and opened up a decade's worth of research. A key resulting artifact is PMDK [21]—a suite of libraries providing a single consistency model across a range of hardware persistence features. Storage researchers working at the device or block level watched with interest as memory researchers tackled key storage issues like transactions and atomic writes.

Looking up from the block layer, PMem changed very little. Researchers quickly dealt with the low-hanging fruit (*e.g.*, block-mode abstractions to PMem [4]). Otherwise, there were few opportunities at the storage (*i.e.*, block) layer.

But CXL will change this in two ways: (1) by bringing memory abstractions to a standardized I/O interconnect, and (2) by making persistence optional (as discussed earlier, CXL works with both volatile and non-volatile memory). This means that memory and storage researchers will need to coordinate, especially if the goal is an optimized solution that spans all hardware and software layers.

For example, although a CXL device could be exposed as a hybrid device with a completely separate memory API (CXL.mem and/or CXL.cache) and storage API (CXL.io), designing such a solution is a missed opportunity. Rather, the memory "half" of a device should leverage the storage half for bulk data, and the storage half should leverage the memory half for coherence and byte addressability. One example is a computational SSD that modifies data in host memory, without resorting to bulk DMA operations. Alternatively, consider a GPU or an FPGA using CXL.cache to gain coherent access to host memory. If that same data is destined for block storage, we do not want to send it to the PCI layer a second time; the data may already be partially present in the device, just in a memory form. Hence, CXL introduces the need for the memory and storage halves to coordinate, and therein lies the potential for new research.

New research opportunities. We introduce new opportunities brought about by CXL across three dimensions: persistence, byte addressability, and coherence. We consider six of the eight possible combinations: three map to existing memory or storage technologies and three are entirely new, representing research opportunities going forward.

For the taxonomy in Table 1, we define *persistent* as being able to survive a cold reboot or loss of power, *coherent*[4] to mean that read operations (across CPUs or hosts as appropriate) will transparently see the result of write operations from other CPUs or hosts, and *byte-addressable* as allowing accesses smaller than a single sector (512 bytes). It is worth noting that byte addressability does not require a coherent memory interface. Indeed, object storage protocols already allow for byte-granular access [35] on the PCIe bus using versions of standard I/O commands; we therefore treat coherency and byte-addressability independently.

A number of rows represent conventional storage technologies. Rows 1 and 2 represent RAM disks and conventional block devices such as NVMe drives. Access is at a block granularity, and cached data (*i.e.*, kernel buffer caches) is managed "manually". Row 6, in turn, corresponds to existing PMem architectures, combining persistence, cache coherency, and byte addressability.

Other combinations are less common. In row 3, read and write operations can be performed at byte granularity, but without coherence or persistence. PCIe address space provides these semantics, with operations performed via load and store instructions. Although the NVMe specification defines an optional PCIe address space allowing such direct access, it is not supported by any commonly available devices. Alternatively, InfiniBand RDMA verbs provide an I/O-operation-based mechanism that is byte-addressable but offers noncoherent access to (remote) volatile memory.

In row 4, byte addressability and persistence are combined with non-coherent access, *e.g.*, via I/O commands rather than

[4]We note that coherence in the non-byte-addressable model is not novel, as it is the traditional access model for block devices.

CPU load/store operations. This model is used by object storage devices that provide byte-aligned read and write operations, although it could also be applied to flat address spaces. At present there are no commercially available modern object storage devices; the flat-address-space model corresponds to RDMA access to remote persistent memory.

Combinations with cache-coherent access at block granularity seem either impossible or impractical, and are omitted from Table 1.

Finally, row 5—cache-coherent byte-addressable access to volatile storage—corresponds to CXL memory pooling with volatile RAM.

Research questions. In a post-Optane landscape with CXL attached volatile and non-volatile devices, we see a range of problems which remain to be addressed.

Latency and memory access: Optane memory is no slower than cross-NUMA-node access to DRAM, while potential future technologies may have significantly higher worst-case latency. At what point are architectural changes in the CPU or memory controller needed to address non-uniform access times? Is there a point where software-controlled access commands become more efficient than handling operations with wildly different latencies within the hardware pipeline?

Performance factors: Optane memory provides both byte addressability and low latency—10× less than the fastest (Optane) NVMe devices, and 100× less than typical ones. Optane-based applications and systems have been shown to provide significantly higher performance than NVMe-based ones, but how much of this improvement is due to byte addressability, and how much due to performance? Future PMem technologies may be slower than Optane, and the answer to this question is important for assessing their potential.

Memory pooling and crash consistency: Will the approaches used to provide crash consistency with a single host attached to a single persistent memory be appropriate for multiple attached hosts across multiple failure domains?

Application intent: Operating systems go to great lengths to infer application intent, allowing, *e.g.*, I/O prefetching and migration of data to lower-performance memory tiers. This is more difficult with PMem, where accesses are performed by hardware rather than software, and may be especially important for hybrid PMem systems.

Byte-granular I/O devices: High-performance PMem-based systems often achieve some of their gains by performing small atomic updates to stored data structures, *e.g.*, by atomically swapping pointers [51]. Extensions to the NVMe protocol might allow such accesses to be performed on external storage, without coherent load/store access from the CPU. Is direct load/store access even necessary to achieve the benefits of byte-granular access, or can I/O protocols evolve to incorporate this model?

Collectively, these CXL-enabled opportunities motivate more distributed storage systems research, including job decomposition, scheduling, safely sharing data, and programming and managing storage devices that speak both byte and block protocols. It remains to be seen whether this takes the form of computational memory, computational storage, or some hybrid. Indeed, CXL will blur the lines between memory and storage, allowing us to rethink and expand the role of a "device." Devices will become computing peers, bringing a wide and exciting array of possibilities.

5 CONCLUSION

We posit that the current lack of commercial PMem availability does not detract from its importance and promise as a core storage technology, both in academia and industry. The Compute Express Link (CXL) interconnect carries forward the lessons from previous PMem implementations and lowers the barrier for developing new PMem products. The wide adoption of the CXL standard allays vendor lock-in concerns, and is a core reason that we believe PMem is worth continued research effort. In particular, CXL enables one to consider each PMem attribute separately or in combination: byte addressability, persistence, and direct access via CPU load/store instructions. Finally, new CXL features such as memory pooling and sharing are seeing considerable interest as rich areas for future PMem research and development.

ACKNOWLEDGMENTS

We thank Andrew Rudoff for his extensive contributions to this work. We thank the anonymous reviewers for their constructive feedback. This work was made possible in part thanks to Dell-EMC, NetApp, Facebook, and IBM support; a SUNY/IBM Alliance award; and NSF awards CNS-1910327, CCF-1918225, CNS-1900706, CNS-1951880, CNS-2106263, CNS-2106434, and CNS-2214980.

REFERENCES

[1] JEDEC Solid State Technology Association. 2021. JEDEC Publishes DDR4 NVDIMM-P Bus Protocol Standard.

[2] Lawrence Benson, Marcel Weisgut, and Tilmann Rabl. 2023. What We Can Learn from Persistent Memory for CXL. In *20th Conference on Database Systems for Business, Technology and Web (BTW)*, Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen (Eds.). Gesellschaft für Informatik e.V., Dresden, Germany, 535–554. *https://doi.org/10.18420/BTW2023-48*

[3] Miao Cai and Hao Huang. 2021. A survey of operating system support for persistent memory. *Frontiers of Computer Science* 15 (2021), 154207.

[4] Feng Chen, Michael Mesnier, and Scott Hahn. 2014. A Protected Block Device for Persistent Memory. In *Proceedings of the 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, Santa Clara, CA, 1–12.

[5] Youmin Chen, Youyou Lu, Fan Yang, Qing Wang, Yang Wang, and Jiwu Shu. 2020. FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Lausanne, Switzerland, 1077–1091.

[6] Zhaole Chu, Yongping Luo, and Peiquan Jin. 2021. An Efficient Sorting Algorithm for Non-Volatile Memory. *Int. J. Softw. Eng. Knowl. Eng.* 31 (2021), 1603–1621.

[7] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. 2011. NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, Newport Beach, CA, 105–118.

[8] Compute Express Link. 2022. Compute Express Link (CXL) Specification. Available from *http://www.computeexpresslink.org*.

[9] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin Lee, Doug Burger, and Derrick Coetzee. 2009. Better I/O through Byte-Addressable, Persistent Memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM, Big Sky, Montana, USA, 133–146. *https://doi.org/10.1145/1629575.1629589*

[10] Laxman Dhulipala, Charles McGuffey, Hong Kyu Kang, Yan Gu, Guy E. Blelloch, Phillip B. Gibbons, and Julian Shun. 2019. Sage: Parallel Semi-Asymmetric Graph Algorithms for NVRAMs. *Proc. VLDB Endow.* 13 (2019), 1598–1613.

[11] R. F. Freitas and W. W. Wilcke. 2008. Storage-Class Memory: The Next Storage System Technology. *IBM Journal of Research and Development* 52, 4/5 (July 2008), 439–447. *https://doi.org/10.1147/rd.524.0439*

[12] G. Gill, Roshan Dathathri, Loc Hoang, Ramesh V. Peri, and Keshav Pingali. 2019. Single Machine Graph Analytics on Massive Datasets using Intel Optane DC Persistent Memory. *Proceedings of the VLDB Endowment* 13 (2019), 1304 – 1318.

[13] Tomasz Gromadzki and Jan Marian Michalski. 2019. Persistent Memory Replication Over Traditional RDMA Part 4: Persistent Memory Development Kit (PMDK)-Based PMEM Replication. *https://www.intel.com/content/www/us/en/developer/articles/technical/persistent-memory-replication-over-traditional-rdma-part-4-persistent-memory-development.html*.

[14] Shashank Gugnani, Scott Guthridge, Frank Schmuck, Owen Anderson, Deepavali Bhagwat, and Xiaoyi Lu. 2022. Arcadia: A Fast and Reliable Persistent Memory Replicated Log. arXiv:cs.DC/2206.12495

[15] Jim Handy and Tom Coughlin. 2023. Optane's Dead: Now What? *Computer* 56, 3 (2023), 125–130. *https://doi.org/10.1109/MC.2023.3235096*

[16] Dave Hitz, James Lau, and Michael Malcolm. 1994. File System Design for an NFS File Server Appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference (ATC)*. USENIX Association, San Francisco, California, 19–19.

[17] George Hodgkins, Yi Xu, Steven Swanson, and Joseph Izraelevitz. 2023. Zhuque: Failure Isn't an Option, It's an Exception. *http://nvmw.ucsd.edu/nvmw2023-program/nvmw2023-paper16-presentation_slides.pdf* 14th Non-Volatile Memories Workshop.

[18] Deukyeon Hwang, Wook-Hee Kim, Youjip Won, and Beomseok Nam. 2018. Endurable Transient Inconsistency in Byte-Addressable Persistent B+-Tree. In *USENIX Conference on File and Storage Technologies*. USENIX Association, Oakland, CA, 187–200.

[19] Intel Corporation. 2022. Intel Optane persistent memory and Intel® Xeon® scalable processors offer a practical migration path to memory expansion, tiering, and pooling with Compute Express Link (CXL™)-attached memory devices. *https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-11/optane-pmem-to-cxl-tech-brief.pdf*

[20] Intel Corporation. 2022. Intel Reports Second-Quarter 2022 Financial Results. *https://www.intc.com/news-events/press-releases/detail/1563/intel-reports-second-quarter-2022-financial-results*.

[21] Intel Corporation. 2023. Persistent Memory Development Kit (PMDK). *pmem.io*.

[22] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R. Dulloor, Jishen Zhao, and Steven Swanson. 2019. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *https://doi.org/10.48550/ARXIV.1903.05714*

[23] Jungi Jeong and Changhee Jung. 2021. PMEM-Spec: Persistent Memory Speculation (Strict Persistency Can Trump Relaxed Persistency). In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, virtual, 517–529.

[24] Myoungsoo Jung. 2022. Hello Bytes, Bye Blocks: PCIe Storage Meets Compute Express Link for Memory Expansion (CXL-SSD). In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage)*. ACM, Virtual Event, 45–51. *https://doi.org/10.1145/3538643.3539745*

[25] Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap, Taesoo Kim, Aasheesh Kolli, and Vijay Chidambaram. 2019. SplitFS: Reducing Software Overhead in File Systems for Persistent Memory. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, Huntsville, Ontario, Canada, 494–508.

[26] Olzhas Kaiyrakhmet, Song Yeon Lee, Beomseok Nam, Sam H. Noh, and Young ri Choi. 2019. SLM-DB: Single-Level Key-Value Store with Persistent Memory. In *USENIX Conference on File and Storage Technologies*. USENIX Association, Boston, MA, 191–205.

[27] Rajat Kateja, Andrew Pavlo, and Greg Ganger. 2020. Vilamb: Low Overhead Asynchronous Redundancy for Direct Access NVM. , 17 pages. *https://arxiv.org/abs/2004.09619*

[28] Se Kwon Lee, Jayashree Mohan, Sanidhya Kashyap, Taesoo Kim, and Vijay Chidambaram. 2019. RECIPE: Converting Concurrent DRAM Indexes to Persistent-Memory Indexes. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, Huntsville, Ontario, Canada, 462–477.

[29] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. 2019. Evaluating Persistent Memory Range Indexes. *Proc. VLDB Endow.* 13 (2019), 574–587.

[30] Huaicheng Li, Daniel S. Berger, Stanko Novakovic, Lisa R. Hsu, Dan Ernst, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2022. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vol. 2. ACM, Lausanne, Switzerland, 574–587.

[31] Jen-Kuang Liu and Sheng-De Wang. 2022. CFFS: A Persistent Memory File System for Contiguous File Allocation With Fine-Grained Metadata. *IEEE Access* 10 (2022), 91678–91698.

[32] Qingrui Liu, Joseph Izraelevitz, Se Kwon Lee, Michael L. Scott, Sam H. Noh, and Changhee Jung. 2018. iDO: Compiler-Directed Failure Atomicity for Nonvolatile Memory. In *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Fukuoka, Japan, 258–270.

[33] Sara Mahdizadeh-Shahri, Seyed Armin Vakil-Ghahani, and Aasheesh Kolli. 2020. (Almost) Fence-less Persist Ordering. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, Virtual, 539–554.

[34] Virendra J. Marathe, Margo Seltzer, Steve Byan, and Tim Harris. 2017. Persistent Memcached: Bringing Legacy Code to Byte-Addressable Persistent Memory. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17)*. USENIX Association, Santa Clara, CA. *https://www.usenix.org/conference/hotstorage17/program/presentation/marathe*

[35] Michael P. Mesnier, Gregory R. Ganger, and Erik Riedel. 2003. Object-based Storage. *IEEE Communications* 44, 8 (August 2003), 84–90.

[36] Micron. 2021. Micron Updates Data Center Portfolio Strategy to Address Growing Opportunity for Memory and Storage Hierarchy Innovation. *https://investors.micron.com/news-releases/news-release-details/micron-updates-data-center-portfolio-strategy-address-growing*

[37] Moohyeon Nam, Hokeun Cha, Young ri Choi, Sam H. Noh, and Beomseok Nam. 2019. Write-Optimized Dynamic Hashing for Persistent Memory. In *USENIX Conference on File and Storage Technologies*. USENIX Association, Boston, MA, 31–44.

[38] Dushyanth Narayanan and Orion Hodson. 2012. Whole-System Persistence. *SIGARCH Comput. Archit. News* 40, 1 (mar 2012), 401–410. *https://doi.org/10.1145/2189750.2151018*

[39] Emerson W. Pugh, Lyle R. Johnson, and John H. Palmer. 2003. *IBM's 360 and early 370 systems*. MIT Press, Cambridge, Massachusetts.

[40] Han Jie Qiu, Sihang Liu, Xinyang Song, Samira Khan, and Gennady Pekhimenko. 2022. Pavise: Integrating Fault Tolerance Support for Persistent Memory Applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. ACM, Chicago, IL, 109–123.

[41] The Register. 2022. Last week Intel killed Optane. Today, Kioxia and Everspin announced comparable tech. *https://www.theregister.com/2022/08/02/kioxia_everspin_persistent_memory/*

[42] Andy Rudoff. 2017. Persistent Memory Programming. *USENIX ;login:* 42, 2 (July 2017), 34–40.

[43] Andy M. Rudoff. 2016. Deprecating the PCOMMIT Instruction. *https://www.intel.com/content/www/us/en/developer/articles/technical/deprecate-pcommit-instruction.html*.

[44] Steve Scargall. 2020. *Programming Persistent Memory: A Comprehensive Guide for Developers*. Apress, New York, New York. 5–7 pages. *https://doi.org/10.1007/978-1-4842-4932-1*

[45] Xinyang (Kevin) Song, Sihang Liu, and Gennady Pekhimenko. 2022. Persistent Memory —- A New Hope. *https://www.sigarch.org/persistent-memory-a-new-hope/*

[46] Storage Networking Industry Association. 2017. NVM Programming Model (NPM). *https://www.snia.org/sites/default/files/technical-work/npm/release/SNIA-NVM-Programming-Model-v1.2.pdf*

[47] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. 2019. Persistent Memory I/O Primitives. In *Proceedings of the 15th International Workshop on Data Management on New Hardware (DaMoN)*. ACM, Amsterdam, Netherlands, 1–7.

[48] Jingyu Wang, Shengan Zheng, Ziyi Lin, Yuting Chen, and Linpeng Huang. 2022. Zebra: An Efficient, RDMA-Enabled Distributed Persistent Memory File System. In *International Conference on Database Systems for Advanced Applications*. ACM, Virtual, 341–349.

[49] Wikipedia. 2023. NVDIMM — Wikipedia, The Free Encyclopedia. *http://en.wikipedia.org/w/index.php?title=NVDIMM&oldid=1141063008*. [Online; accessed 27-March-2023].

[50] Jian Xu, Juno Kim, Amir Saman Memaripour, and Steven Swanson. 2019. Finding and Fixing Performance Pathologies in Persistent Memory Software Stacks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, Providence, RI, 427–439.

[51] Jian Xu and Steven Swanson. 2016. NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies*. USENIX Association, Santa Clara, CA, 323–338.

[52] Jian Xu, Lu Zhang, Amirsaman Memaripour, Akshatha Gangadharaiah, Amit Borase, Tamires Brito Da Silva, Steven Swanson, and Andy Rudoff. 2017. NOVA-Fortis: A Fault-Tolerant Non-Volatile Main Memory File System. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, Shanghai China, 478–496. *https://doi.org/10.1145/3132747.3132761*

[53] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. 2020. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory. In *USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, Santa Clara, CA, 169–182.

[54] Ziye Yang, James R. Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E. Paul. 2017. SPDK: A Development Kit to Build High Performance Storage Applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Hong Kong, 154–161. *https://doi.org/10.1109/CloudCom.2017.14*

[55] Wenhui Zhang, Xingsheng Zhao, Song Jiang, and Hong Jiang. 2021. ChameleonDB: A Key-value Store for Optane Persistent Memory. In *Proceedings of the Sixteenth European Conference on Computer Systems (Eurosys)*. ACM, Edinburgh, Scotland, 194–209.