# A brief primer on lightweight Persistent Memory Objects

Derrick Greenspan
derrick.greenspan@ucf.edu
University of Central Florida
Orlando, Florida, United States

Naveed Ul Mustafa
num@nmsu.edu
New Mexico State University
Orlando, Florida, United States

Mark Heinrich
heinrich@ucf.edu
University of Central Florida
Orlando, Florida, United States

Yan Solihin
yan.solihin@ucf.edu
University of Central Florida
Orlando, Florida, United States

Jongouk Choi
jongouk.choi@ucf.edu
University of Central Florida
Orlando, Florida, United States

*Note: click **here** [3] for the original paper.*

## 1 Motivation

Persistent Memory Objects (PMOs) are the state-of-the-art OS-based approach for persistent memory (PM) management. Recent PMO designs have limited performance due to the properties of the PM substrate and the requirement that persistent data be secure and robust against corruption.

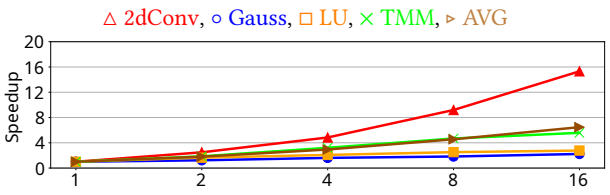



**Figure 1: Microbenchmarks [2] scalability by thread count using PM (with the x-axis representing the number of threads).**

To analyze this problem, we evaluated the state-of-the-art LOaAPP PMO design [4] in a real system with Intel Optane Pmem. We ran four benchmark applications: 2d Convolution, Gaussian Elimination, LU Decomposition, and Tiled Matrix Multiplication, varying the number of threads from 1 to 16. Figure 1 shows the speedup ratios of several benchmarks from 1 to 16 threads with the state-of-the-art Linux-based PMO [4]. Except for 2d Convolution, which is not particularly memory intensive, all benchmarks show poor speedup scaling, e.g. only $\approx 3\times$ at 16 threads for LU. We solved this problem via our LPMO design, which achieves up to 1.25× speedup over a state-of-the-art PMO. When combined with page prediction, the speedup improves further to 1.81×. In CXL-memory configurations, LPMO can improve the performance by up to 3× compared to the local PM-based PMO systems, despite the added latency of CXL.

## 2 Threat and Trust Model

Similar to files, PMOs keep persistent data for a long time, and most of them will spend most of their lifetime at rest. Our threat model assumes the attacker's goal is to either reveal or tamper with the confidential data belonging to a user-process stored within an at-rest PMO.

One attack we consider is data remanence, where a stolen or improperly disposed of PM may be analyzed by the attacker to obtain sensitive data. Such attacks have long been documented for files in hard drives [7] and prompted filesystem encryption which is widely used today. Likewise, PMO encryption was motivated by the same concerns [4].

Another attack we consider has the adversary compromise a user account to steal or corrupt PMO data. But without having the correct key, the attacker cannot read the plaintext of PMO data, or modify PMO data without being discovered later. PMO encryption keys can be further managed by a Trusted Platform Module (TPM) to avoid the attacker reading keys from memory.

Our trust is limited to specific components of the system software, Linux Kernel Crypto API [1], crucial kernel memory functions like memcpy and memset, and our PMO kernel subsystem.

In an unprotected system, the attack proceeds as follows: 1) The attacker discovers the physical address (PA) of the PMO. 2) The attacker maps the PMO to its address space, and 3) performs the attack by either corrupting the PMO by writing incorrect data, which will not be caught in the absence of integrity verification [5], or alternatively, in the absence of encryption, the attacker can read from the PMO and steal secrets from it. In either case, the attacker is able to do these actions silently, by simply 4) unmapping the address space of the PMO from the attacker process's address space, which leaves no evidence that the PMO was accessed or modified.

## 3 LPMO Design

### 3.1 DRAM Shadow Paging

Prior PMO systems placed each PMO entirely in the PM, both shadow and primary pages. While in this approach the PMO retains data on power loss (or crashes), it suffers from high access latencies (especially writes) and low write bandwidth. Furthermore, a typical memory system with PM has a mixture of DRAM and PM, and the DRAM is underutilized. We propose a solution where shadow pages are placed in DRAM, while primary pages are placed in PM. Only shadow pages are lost on a crash.

Although our approach incurs a risk of data loss; as long as the data loss is limited to any modifications after the last successful `psync()`, the semantics are not violated. Our solution requires that shadow pages are merged into the primary pages on psync. The cost of this is that psync takes slightly longer to complete, but psync occurs less often than writes. Also, for encryption, we keep the shadow in plaintext while the primary is in ciphertext. The cost is minor: a psync involves encrypting the shadow into the primary instead of copying it.

### 3.2 Page Access Prediction

While DRAM shadow paging helps performance, accessing the page for the first time still requires a page fault (to place the shadow page into DRAM), decryption, and integrity verification. Together, these incur a substantial critical-path delay that places an upper-bound on performance improvements. To tackle this problem, we propose predicting access patterns, decrypting them into DRAM, and verifying their integrity ahead of time.

For this to work, the predictor should be lightweight and fast at generating predictions, and yet also achieve high accuracy, coverage, and timeliness. To satisfy this, we use a page-usage pattern predictor based on stream buffers [6].

### 3.3 CXL design

We envision that the LPMO abstraction can be applied both to local and CXL-attached PM systems, enabling new memory hierarchies. Some examples are: a CXL PM device without a DRAM cache, a CXL PM device with a local DRAM cache, and a CXL PM device with DRAM also attached to it. These are reflected as L (for local), F (for far) and D (for DRAM) in Figure 4.

We also utilize enhanced memory functions (EMFs) to simulate hardware-based support for PMO functions, such as encryption and integrity verification. We will test this in our evaluation and apply a 1% overhead to attach and psync.

## 4 Evaluation

We evaluated LPMO with a system that consists of a dual socket Supermicro X11DPi-NT, two Xeon Gold 6230s, and four 128GiB Optane memory sticks. We use the microbenchmarks described in Section 1. Our legend is depicted in Figure 2.


**Figure 2: Microbenchmark Legend**

### 4.1 LPMO Performance

Figure 3 compares the execution times of various microbenchmarks and their average for the state-of-the-art GPMO system. O is the original GPMO design, D adds DRAM without prediction, and the numbers after D are the prediction depth. Our scheme reduces execution time by ≈ 21% on average, due to a reduction in the page fault delay. The performance impact of Integrity Verification is also lessened, and in some cases completely eliminated, when using DRAM and predictive encryption together.

### 4.2 CXL Performance

Figure 4 demonstrates the latency of encryption and integrity verification of a CXL device. We find that despite the added latency of CXL, the LPMO abstraction has performance comparable to, and sometimes faster than, the prior most-performant PMO design. Note that for LU, the page fault overhead (from faulting unnecessary pages) is very high for our CXL system at a depth of 8, which is likely because of the added latency of CXL NUMA nodes and the fact that LU is heavily memory bound.

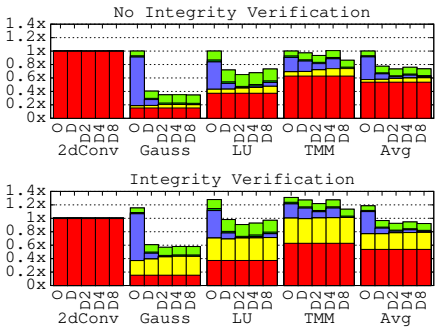
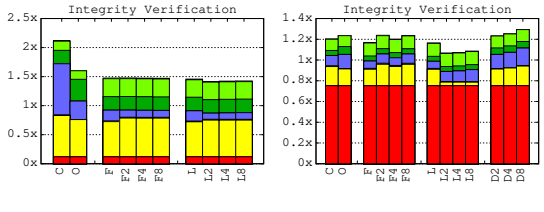


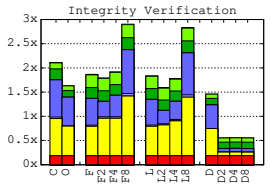

**Figure 3: Execution time with and without DRAM prediction.**



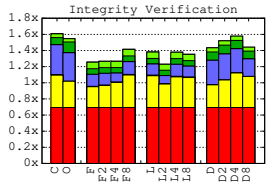

**(a) Gaussian Elimination**




**(b) 2D Convolution**




**(c) LU Decomposition**




**(d) Tiled Matrix Multiplication**

**Figure 4: CXL Microbenchmark Evaluation**

## References


[1] Kernel Development Community. 2023. Block Cipher Algorithm Definitions. https://www.kernel.org/doc/html/v5.14/crypto/api-skcipher.html#symmetric-key-cipher-api https://www.kernel.org/doc/html/v5.14/crypto/api-skcipher.html#symmetric-key-cipher-api.

[2] Hussein Elnawawy, Mohammad Alshboul, James Tuck, and Yan Solihin. 2017. Efficient Checkpointing of Loop-Based Codes for Non-volatile Main Memory. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, Portland, Oregon, USA, 318–329. doi:10.1109/PACT.2017.58

[3] Derrick Greenspan, Naveed Ul Mustafa, Jongouk Choi, Mark Heinrich, and Yan Solihin. 2025. Persistent Memory Objects on the Cheap. In *Proceedings of the 39th ACM International Conference on Supercomputing*. 1234–1249.

[4] Derrick Greenspan, Naveed Ul Mustafa, Andres Delgado, Connor Bramham, Christopher Prats, Samu Wallace, Mark Heinrich, and Yan Solihin. 2024. LOaPP: Improving the Performance of Persistent Memory Objects via Low-Overhead at-Rest PMO Protection. In *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, IEEE, Orlando, Florida, USA, 131–142.

[5] Naveed Ul Mustafa, Yuanchao Xu, Xipeng Shen, and Yan Solihin. 2021. Seeds of SEED: New Security Challenges for Persistent Memory. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, IEEE, Virtual, 83–88.

[6] S. Palacharla and R. E. Kessler. 1994. Evaluating stream buffers as a secondary cache replacement. In *Proceedings of the 21st Annual International Symposium on Computer Architecture* (Chicago, Illinois, USA) *(ISCA '94)*. IEEE Computer Society Press, Washington, DC, USA, 24–33. doi:10.1145/191995.192014

[7] P Roberts. 2003. MIT: Discarded hard drives yield private info. https://www.computerworld.com/article/1334164/mit-discarded-hard-drives-yield-private-info.html