SuperMem: Enabling Application-transparent Secure Persistent Memory with Low Overheads [*]

Pengfei Zuo[*†], Yu Hua[*], Yuan Xie[†]
[*]*Huazhong University of Science and Technology,* [†]*University of California, Santa Barbara*

1 Background and Motivation

As DRAM suffers from limited scalability and high power leakage, persistent memory (PM) becomes promising candidates of the next-generation main memory. PM has the advantages of high scalability, high density, and near-zero standby power. However, two fundamental issues need to be addressed to effectively use PM in memory systems, i.e., data persistence and security.

First, the non-volatility of PM enables data to be persistently stored into main memory for instantaneous failure recovery. In order to ensure the correctness of persistent data, crash consistency guarantee is non-trivial, a factor which needs to achieve the correct recovery of persistent data in case of a system failure. Second, the non-volatility of PM also causes the security problem of data remanence vulnerability, since PM still retains data after systems are powered down. If a PM DIMM is stolen, an attacker can easily stream out the data from the DIMM. Hence, memory encryption becomes important to ensure data security in PM. Counter mode encryption [2] is usually used in secure PM, due to its low decryption latency and high security level.

Nevertheless, it is challenging to ensure both crash consistency and security in PM. This is because each data write to encrypted PM generates two write requests, i.e., one for the data and the other for its counter. To guarantee crash consistency, the two writes have to be persisted at the same time. If a system failure occurs when the data is persisted into PM but its counter is not, the data fails to be decrypted upon the system recovery due to no correct counter. Similarly, if a system failure occurs when the counter is persisted into PM but its data is not, the old-version data in PM fails to be correctly decrypted. Unfortunately, current computer systems cannot atomically perform the two write requests to PM, since the data is evicted from CPU caches and the counter is evicted from the counter cache managed by the memory controller.

To guarantee crash consistency of secure PM and reduce its performance overhead, all existing works are based on *a write-back counter cache* and propose three solutions, i.e., battery backup [1,6], software-level modification [3], and error correction [4].

[*]Original Paper: Pengfei Zuo, Yu Hua, Yuan Xie, “SuperMem: Enabling Application-transparent Secure Persistent Memory with Low Overheads”, in Proc. of MICRO, 2019, pages: 479–492.

Different from existing work, our proposed SuperMem leverages *a write-through counter cache*, without a large battery backup, software-layer modifications, and error correction. Moreover, by leveraging cross-bank counter storage and locality-aware counter write coalescing, SuperMem achieves the performance comparable to a secure PM with an ideal write-back counter cache that presents the optimal performance of an encrypted PM.

2 Our Approach

Our paper proposes **SuperMem**, a software-transparent **Secu**re **per**sistent **Mem**ory. SuperMem employs a write-through counter cache with a register (§2.1) that enables crash consistency to be much easier than existing work using a write-back counter cache. When the data is written into PM, its corresponding counter is also written into PM following the data. Thus SuperMem is application-transparent which does not require programmers to actively flush counters from the counter cache into PM. However, using a write-through counter cache always generates two write requests for each data write, decreasing the system performance. We propose an efficient cross-bank counter storage (Xbank) scheme (§2.2) to distribute the data write and its counter write to different banks, thus improving the system performance by leveraging band parallelism. Moreover, we also propose a counter write coalescing (CWC) scheme (§2.3) in SuperMem to significantly reduce the number of write requests by leveraging the spatial locality of counter and data writes.

2.1 Write-through Counter Cache

SuperMem employs a write-through counter cache with a register to guarantee crash consistency of data and counter writes. During encrypting the data (i.e., a cache line) evicted from CPU caches ($Enc(A)$), we store its corresponding counter in the register ($Sto(Ac)$) instead of directly appending the counter in the write queue, as shown in Figure 1. After encrypting the data, we first store the encrypted data in the register ($Sto(A)$), and then simultaneously append the encrypted data and its counter in the write queue ($App(Ac+A)$). We observe that either data and its associated counter simultaneously exist in the write queue or not, by using a register. As a result, the crash consistency of data and its counter is ensured in SuperMem.

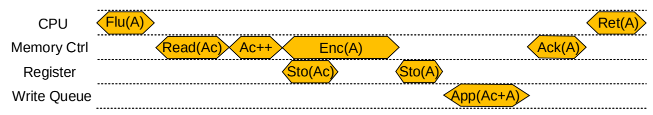
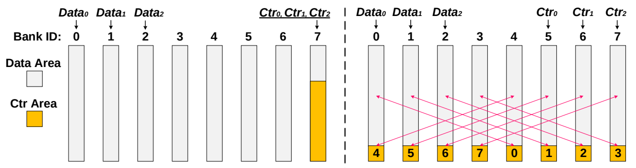


Figure 1: The sequence dealing with a cache line flush.



(a) SingleBank (b) XBank

Figure 2: Different counter storage approaches.

2.2 Cross-bank Counter Storage

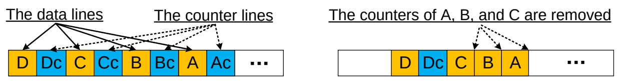
In existing secure PM work with counter mode encryption [1,3, 5,6], counters are generally stored in a continuous area in PM (SingleBank), as shown in Figure 2a. The storage approach is efficient for the write-back counter cache, since most counter writes are buffered in the counter cache and not written into PM. However, when employing a write-through counter cache, each data write produces a counter write to the counter storage bank. Thus the bank storing counters becomes a bottleneck, decreasing the system performance. For example, three data write requests, i.e., $Data_0$, $Data_1$, and $Data_2$, are sent to Banks 0, 1, and 2, respectively, as shown in Figure 2a. All their counter write requests, i.e., Ctr_0, Ctr_1, and Ctr_2, are sent to the counter storage bank, i.e., Bank 7. The three data write requests can be served simultaneously due to the parallelism among banks. However, the three counter write requests have to be served one by one, thus degrading the system performance.

To reduce the processing time of write requests, we propose *a cross-bank counter storage (XBank) scheme* which stores each data and its counter into different banks instead of the same bank. The two banks storing the data and their corresponding counters are one-to-one way as shown in Figure 2b. Moreover, the interval between the two bank numbers storing data and their corresponding counters should be as large as possible. By performing the XBank scheme, data and counter writes are distributed into different banks, efficiently reducing bank access conflicts. Hence, writes are sped up by leveraging bank parallelism.

2.3 Locality-aware Counter Write Coalescing

In counter mode encryption, the counter storage has good spatial locality. This is because all counters of a memory page are stored in one counter memory line. It uses a shared major counter (M) for a page and 64 minor counters ($m_1, m_2, ..., m_{64}$) each for a memory line in the 4KB page. The major counter is 64 bits and each minor counter is 7 bits. Moreover, data and log writes usually exhibit good spatial locality, since a log is stored in a contiguous region and programs usually allocate a contiguous memory region.

Based on the locality existing in counter storage, log and data writes, SuperMem leverages *a locality-aware counter write coalescing (CWC) scheme* in SuperMem to reduce the number of counter writes. When a new counter cache line evicted from the counter cache reaches the write queue, we check whether a counter cache line in the write queue has the same physical address as the new one. If yes, we merge these cache lines with the same physical address. By using CWC, SuperMem reduces almost half of memory writes, as shown in Figure 3.



(a) The write queue without CWC (b) The write queue with CWC

Figure 3: The write queue with/without CWC.

2.4 Experimental Results

We have implemented and evaluated SuperMem in a full system simulator gem5 with NVMain. Experimental results show the CWC scheme reduces up to 50% of write requests in the encrypted PM with a write-through counter cache, and the XBank scheme improves the system performance by up to 2×. Thus SuperMem achieves the performance comparable to an ideal secure PM that presents the optimal performance of an encrypted PM.

3 Conclusion

This paper proposes SuperMem to achieve both security and persistence in non-volatile main memory. SuperMem leverages a write-through counter cache scheme with a register to guarantee crash consistency in the encrypted NVM. Moreover, a counter write coalescing scheme is introduced to reduce the number of write requests and a cross-bank counter storage scheme is employed to reduce the processing time of write requests. These schemes are implemented with slight modifications only on the hardware layer, which are transparent for programmers and applications. Thus programs and applications running on an un-encrypted NVM can be directly executed on an encrypted NVM with SuperMem. Experimental results show that SuperMem achieves the performance comparable to an ideal secure NVM exhibiting the optimal performance of an encrypted NVM.

References

[1] Awad, A., Manadhata, P., Haber, S., Solihin, Y., and Horne, W. Silent Shredder: Zero-cost shredding for secure non-volatile main memory controllers. In *ASPLOS* (2016).

[2] Lipmaa, B. H., Rogaway, P., and Wagner, D. Ctr-mode encryption, comments to nist concerning aes modes of operations. In *NIST Workshop on Modes of Operation* (2000).

[3] Liu, S., Kolli, A., Ren, J., and Khan, S. Crash consistency in encrypted non-volatile main memory systems. In *HPCA* (2018).

[4] Ye, M., Hughes, C., and Awad, A. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories. In *MICRO* (2018).

[5] Young, V., Nair, P. J., and Qureshi, M. K. DEUCE: Write-efficient encryption for non-volatile memories. In *ASPLOS* (2015).

[6] Zuo, P., Hua, Y., Zhao, M., Zhou, W., and Guo, Y. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *MICRO* (2018).