SecPM: a Secure and Persistent Memory System for Non-volatile Memory

Pengfei Zuo, Yu Hua✉
Wuhan National Laboratory for Optoelectronics
School of Computer, Huazhong University of Science and Technology, Wuhan, China
✉*Corresponding author: Yu Hua (csyhua@hust.edu.cn)*

Abstract

In the non-volatile memory, ensuring the security and correctness of persistent data is fundamental. However, the security and persistence issues are usually studied independently in existing work. To achieve both data security and persistence, simply combining existing persistence schemes with memory encryption is inefficient due to crash inconsistency and significant performance degradation. To bridge the gap between security and persistence, this paper proposes **SecPM**, a **Sec**ure and **P**ersistent **M**emory system, which consists of a counter cache write-through (CWT) scheme and a locality-aware counter write reduction (CWR) scheme. Specifically, SecPM leverages the CWT scheme to guarantee the crash consistency via ensuring both the data and its counter are durable before the data flush completes, and leverages the CWR scheme to improve the performance via exploiting the spatial locality of counter storage, log and data writes. Preliminary experimental results demonstrate that SecPM significantly reduces the number of write requests and improves the transaction throughput by using the CWR scheme.

1 Introduction

As DRAM suffers from limited scalability and high power leakage [19, 29], non-volatile memories (NVM), such as PCM [32], ReRAM [5], STT-RAM [6] and 3D XPoint [1], become promising candidates of the next-generation main memory. NVM has the advantages of high scalability, high density, and near-zero standby power. However, two fundamental challenges need to be addressed in order to effectively use NVM in memory systems, i.e., data persistence and security.

• ***Persistence.*** The non-volatility of NVM enables data to be persistently stored into NVM for instantaneous failure recovery. In order to achieve the correctness of persistent data, the crash consistency guarantee is fundamental [20, 31], which needs to ensure the correct recovery of persistent data in case of a system failure, e.g., power failure and system crash. Specifically, NVM systems typically contain volatile storage components, e.g., CPU caches. If a system failure occurs when a data structure in NVM is being updated, the data structure may be left in a corrupted state. Moreover, modern processor and memory controller usually reorder memory writes. The partial update and reordering cause the crash inconsistency in NVM [15, 35]. Hence, the cache line flushes, memory barriers, and log-based mechanisms are used to ensure the crash consistency [17, 33].

• ***Security.*** The non-volatility of NVM also causes the security problem of data remanence vulnerability [36, 7], since NVM still retains data after systems power down. In general, when using encryption to protect the data security, the encrypted data are stored in disks, while raw data are retained in main memory [12]. In the legacy DRAM-based memory, if a DRAM DIMM is stolen, data are quickly lost due to the volatility. Unlike it, if an NVM DIMM is stolen, an attacker can directly stream out the data from the DIMM. Hence, memory encryption becomes important to ensure the data security in NVM. Counter mode encryption [16, 36] is usually used in secure NVM, due to the low decryption latency.

However, existing schemes addressing the persistence issue [20, 31, 13, 33, 25] usually fail to efficiently use memory encryption in NVM systems. On the other hand, existing schemes addressing the security issue [36, 7, 28, 9] are inefficient to guarantee the data consistency. To achieve a secure persistent memory, simply combining existing persistence schemes with memory encryption does not work due to the following challenges.

• ***Consistency challenge.*** In order to guarantee the crash consistency, it is essential to use the cache line flush and memory barrier instructions to persist data into NVM with correct ordering. In the counter mode encryption, each memory line is encrypted/decrypted using a counter, and the counter increases one on each write [16]. Thus each write in the secure NVM has to persist two data including the data itself and the counter. The data is evicted from the CPU caches and the counter is evicted from the counter cache managed by the memory controller. However, the current cache line flush and memory barrier instructions only ensure the data from the CPU caches to be correctly persisted into NVM, which fail to operate the counter cache and hence cannot ensure the consistency of counters. As a result, the persisted data cannot be decrypted without correct counters during the recovery from a system failure, thus resulting in an inconsistent state.

• ***Performance challenge.*** Each write in the secure NVM generates two NVM write requests, which significantly degrades the system performance. Because the writes to NVM usually incur much higher latency than reads [23, 37, 38], and are in the critical path of application execution due to persistence requirements [30, 31]. Moreover, more write requests also increase the latency of read requests. When a write request is served by an NVM bank, the following read/write requests to the same bank are blocked and wait until the current write request is completed [22].

To bridge the gap between security and persistence, this paper proposes a secure and persistent memory system, called SecPM. SecPM proposes to use a simple yet efficient counter cache write-through (CWT) scheme to ensure the crash consistency of both data and counters. CWT appends the counter of the data in the write queue during encrypting the data, which ensures the counter is durable before the data flush completes. Moreover, SecPM leverages a locality-aware counter write reduction (CWR) scheme to improve the system performance. CWR explores and exploits the spatial locality of counter storage, log and data writes to merge the write requests from counters in the write queue, thus reducing the NVM writes. Our evaluation demonstrates that SecPM reduces up to half of write requests, and improves the transaction throughput by 1.4 $\sim$ 2.1 times, by using the CWR scheme.

2 Backgrounds and Motivations

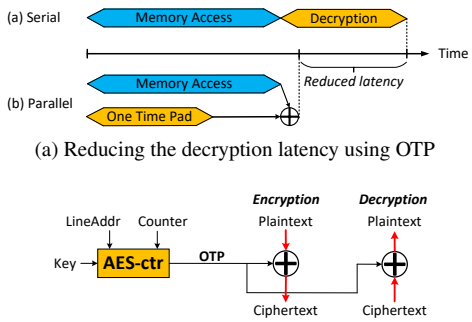
2.1 Consistency Guarantee for Persistence

In order to correctly persist data in NVM, it is important to ensure the crash consistency. Durable transaction [13, 17] is a commonly used solution for crash consistency guarantee, which enables a group of memory updates to be performed in an atomic manner. Figure 1 shows the steps implementing a durable transaction with undo logging, which include three stages, i.e., prepare, mutate and commit. Whichever stage a system failure occurs, the application can be recoverable in a consistent state since at least one of the log and data are consistent.

As modern CPU and memory controller usually reorder memory writes, the durable transaction uses cache line flush and memory barrier instructions to enforce write ordering [13, 20, 17]. The flush instructions including `clflush`, `clflushopt`, and `clwb` explicitly flush a dirty CPU cache line into the write queue of the memory controller. The memory barrier instructions including `mfence` and `sfence` order the memory operations via blocking the memory operations after the fence, until those before the fence complete. The `pcommit` instruction was initially used to force the write requests in the write queue into NVM but was deprecated later [2], due to the use of asynchronous DRAM refresh (ADR) [3, 27, 18]. ADR is able to persist the write requests in the write queue into NVM in case of a system failure via the battery backup. Therefore, the cache lines reaching the write queue can be considered durable.

```
TX_BEGIN
    do some computation;
    // Prepare stage: backing up the data in log
    write undo log;
    flush log;
    memory_barrier();
    // Mutate stage: updating the data in place
    write data;
    flush data;
    memory_barrier();
    // Commit stage: invalidating the log
    log->valid = false;
    flush log->valid;
    memory_barrier();
TX_END
```

Figure 1: Steps implementing an undo-log transaction.



(a) Reducing the decryption latency using OTP

(b) The encryption/decryption process

Figure 2: The counter mode encryption.

2.2 Memory Encryption for Security

Since NVM still retains data after systems power down, an attacker can easily stream out the data stored in the NVM after stealing the NVM DIMM. Hence, memory encryption is important to ensure the data security in NVM. However, memory encryption causes the high decryption latency in the critical path of memory reads, thus degrading the system performance. Counter mode encryption [16] is proposed to mitigate the decryption latency from the critical path of memory reads via leveraging the One Time Pad (OTP) technique, and hence has been widely used in the NVM systems [36, 7, 9, 28]. The main idea is to compute an OTP in parallel with a memory read, and then XOR the OTP with the ciphertext data, thus hiding the decryption latency in the memory access latency, as shown in Figure 2a.

The security of counter mode encryption is based on the premise that each OTP is never reused for data encryption [16, 36, 28]. To ensure this, the counter mode encryption uses a secret key, the line address and the per-line counter to generate the OTP through the AES circuit, as shown in Figure 2b. Therefore, data stored at different addresses are encrypted by different OTPs. Moreover, the per-line counter increases on each write to generate different OTPs for data rewrites of the same line. Hence, the OTPs are never reused. To reduce the generation

Table 1: The recoverability when a system failure occurs in the different stages of a transaction.

Stage	Log content	Log counter	Data content	Data counter	Recoverable ?
Prepare	Wrong	Wrong	Correct	Correct	Yes
Mutate	Correct	Unknown	Wrong	Wrong	No
Commit	Correct	Unknown	Correct	Unknown	No

time of OTPs, counters are buffered in an on-chip counter cache [36, 28] managed by the memory controller.

2.3 The Gap between Persistence and Security

Each cache line flushed from CPU caches in the encrypted NVM produces two write requests: one for the data and the other for the counter. This characteristic not only degrades the system performance since NVM needs to deal with more write requests, but also incurs the crash inconsistency problem. The reason is that the data is evicted from the CPU caches but its counter is evicted from the counter cache. However, the cache line flush instruction only flushes the data in CPU caches, but fails to operate the counter cache. The memory barrier instruction only works for the CPU cache line flushes, but fails to ensure the ordering of counter writes.

We analyze the recoverability when a system failure occurs in the different stages of a transaction executed in an encrypted NVM, as shown in Table 1. We observe when a system failure occurs in the mutate and commit stages, the data is unrecoverable. Specifically, when a system failure occurs in the prepare stage, the contents and counters of the data are unmodified and correct, which are in a consistent state. However, when a system failure occurs in the mutate stage, the data are not updated completely and become wrong. The contents of the log are correct due to the use of log flushes and memory barriers, but whether the counters of the log are correctly persisted is unknown, since the cache line flush and memory barrier instructions fail to operate the counters stored in the counter cache. Hence, during a recovery, the log cannot be decrypted due to no correct counters and fails to recover the logged data. For the same reasons, when a system failure occurs in the commit stage, the correctness of both log and data counters are unknown, and hence the data is unrecoverable.

To address the inconsistency problem, Liu et al. [18] proposed the concept of the selective counter-atomicity which indicates that either both data and its associated counter have persisted or neither data nor its associated counter has persisted. However, to implement the counter-atomicity, significant modifications are made for both software and hardware layers [18]. First, two new primitives including `CounterAtomic` variable and `counter_cache_writeback()` function, are added in the programming language. Second, the compiler have to be modified to support the new primitives. Third, an extra counter write queue is added into the memory controller. As a result, the programs initially running on a system with the un-encrypted NVM cannot directly run on a system with the secure NVM. Our paper proposes SecPM to guarantee the crash consistency in secure NVM without the needs of the modifications on programming language and compiler.

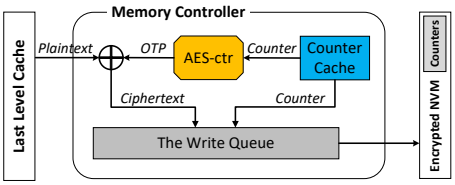


Figure 3: The hardware architecture of SecPM.

3 The SecPM Design

We propose SecPM, a **Sec**ure **P**ersistent **M**emory system, which leverages a simple yet efficient counter cache write-through scheme (§ 3.1) to guarantee the crash consistency, and a counter write reduction scheme (§ 3.2) to significantly reduce the number of write requests for improving the system performance.

3.1 Consistency Guarantee

The hardware architecture of SecPM is shown in Figure 3. When CPU issues a flush instruction, the corresponding cache line is evicted from the last level cache to the memory controller. The memory controller encrypts the cache line using a counter and then appends the encrypted cache line in the write queue. SecPM employs a counter cache write-through (CWT) scheme in the counter cache, which writes each dirty counter in the counter cache, and simultaneously writes the counter copy in the write queue. We further show how to use the CWT scheme to ensure the crash consistency.

Figure 4 shows the sequence diagram that the memory controller deals with a cache line flush. When the CPU flushes a cache line A (*Flu(A)*), the memory controller reads the counter of A from the counter cache (*Read(Ac)*), and adds the counter by 1 (*Ac++*). The updated counter is used to encrypt A (*Enc(A)*). During the encryption, the updated counter is written back to the counter cache, and simultaneously appended in the write queue (*App(Ac)*) via the CWT scheme.

After the encrypted A is appended in the write queue (*App(A)*), the memory controller sends an ack (*Ack(A)*) to the CPU, and the flush is retired (*Ret(A)*). From Figure 4, we observe that the counter encrypting a CPU cache line

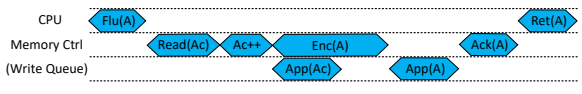


Figure 4: The sequence that CPU flushes a cache line.

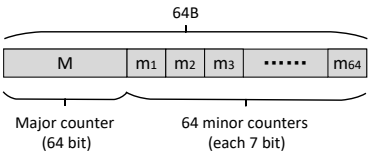


Figure 5: The counter storage of memory lines in a physical page.

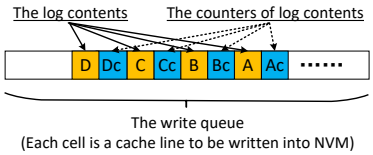


Figure 6: An illustration of the write queue when writing a log.

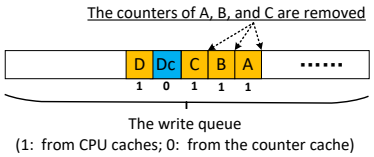


Figure 7: The write queue when writing a log via the CWR scheme.

Table 2: The recoverability of a transaction in SecPM.

Stage	Log content	Log counter	Data content	Data counter	Recoverable ?
Prepare	Wrong	Wrong	Correct	Correct	Yes
Mutate	Correct	Correct	Wrong	Wrong	Yes
Commit	Correct	Correct	Correct	Correct	Yes

has been already appended in the write queue before the cache line flush completes via the CWT scheme. Hence, if the contents of log/data have been persisted and become correct, their counters have also been persisted. Table 2 shows the recoverability when a system failure occurs in the different stages of a transaction in SecPM. We observe that at least one of log and data is correct whichever stage a system failure occurs in, and hence the system can be recoverable in a consistent state.

Recovery: We present the recovery scheme of transactions in SecPM after a system failure occurs. During the recovery, we scan the undo log region, and check each log entry. For each log entry, we check whether the log is complete via the log-end tag of a transaction. If the log is incomplete, it means the system crashes in the prepare stage of the transaction, in which the original data are correct. We directly abandon the incomplete log. If the log is complete, it means the system crashes in the mutate or commit stage, in which the log is correct. We undo the data via the log. Therefore, the system can be recovered in a consistent state in SecPM.

3.2 Counter Write Reduction

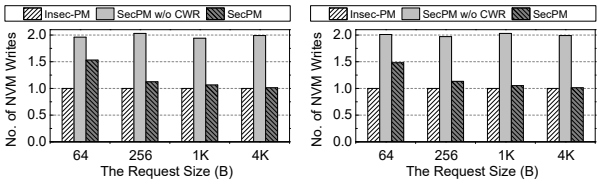
Each CPU cache line flush appends two write requests in the write queue as shown in Figure 4, which doubles the number of write requests, compared with an unencrypted NVM. Thus the performance of the memory system would significantly decrease. SecPM proposes a locality-aware counter write reduction (CWR) scheme to improve the system performance via leveraging the spatial locality of counter storage, log and data writes.

The spatial locality of counter storage: In order to reduce the storage overhead of counters, the counter mode encryption uses a shared major counter (M) for an entire page and 64 minor counters (m_1, m_2, ..., m_{64}) each for a memory line in the 4KB page [34, 7], as shown in Figure 5. The counters of all memory lines in a page are 64B and stored in one memory line, exhibiting good spatial locality. Each memory line is encrypted by the major counter concatenated with a minor counter. When a memory line is rewritten, its minor counter increases by 1. Although updating a counter only modifies several bits, persisting the counter has to write the entire memory line into NVM since a memory line is the basic unit of memory writes.

The spatial locality of log and data writes: Since a log is stored in a contiguous region in NVM, the log writes of a transaction flush multiple cache lines which have the contiguous physical addresses, thus having good spatial locality. Moreover, the data writes of a transaction usually have good spatial locality, since programs usually allocate a contiguous memory region for a transaction. Hence, the cache lines flushed into the contiguous region have the contiguous physical addresses. For example, a transaction inserts a 1KB key-value item into a key-value store maintained in NVM, which will flush 16 cache lines with the contiguous physical addresses.

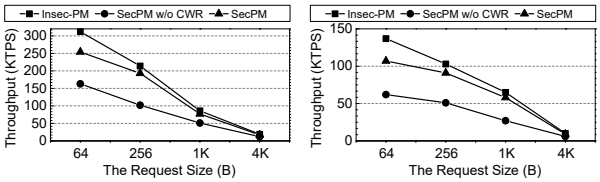
Based on these localities, we show the write queue during flushing the log of a transaction in Figure 6. The log contents contain multiple cache lines (i.e., A, B, C, and D) with contiguous physical addresses in the same physical page. Since the counters of a page are contained in one memory line as shown in Figure 2a, the counters of the log contents, i.e., A_c, B_c, C_c, and D_c, will be written to the same memory line. Moreover, since these counter cache lines are evicted from the write-through counter cache, the latter cache lines contain the updated contents of the former ones with the same address. For example, the cache line A_c only contains the updated counter of A, and the cache line B_c contains the updated counters of A and B, and the cache line C_c contains the updated counters of A, B, and C.

Based on above observations and insights, we present a counter write reduction (CWR) scheme. Specifically, when a new cache line evicted from the counter cache reaches the write queue, we check whether an exiting cache line in the write queue has the same physical address as the new one. If yes, we remove this existing cache line without causing any loss of data, since the new cache line contains the updated contents of the removed one as mentioned above. To reduce the latency of checking the cache lines with the same address, we add a one-bit flag for each cache line in the write queue. The flag is used to distinguish whether a cache line is from CPU caches or the counter cache. Thus we can check the cache lines only from the counter cache based on the flag. By performing the CWR scheme, the new write queue is shown in Figure 7. We observe that the number of write requests is significantly reduced.



(a) Hash table based KV store (b) B-tree based KV store

Figure 8: The number of NVM writes normalized to the insecure persistent memory.



(a) Hash table based KV store (b) B-tree based KV store

Figure 9: The transaction throughput for the two key-value stores.

4 Performance Evaluation

1) Methodology. We model the proposed SecPM in the gem5 [8] with NVMain [21]. NVMain is a cycle-accurate main memory simulator for emerging NVM technologies. The simulated system consists of x86-64 processors run at 2 GHz, 32KB L1 data and instruction caches, 2MB L2 cache, and 8MB L3 cache. The counter cache is 1MB. Without loss of generality, we model PCM technologies [10] with the read/write latency of 150ns/450ns and 16GB capacity. The encryption and decryption latency of AES circuit are 40ns [26]. We revise the gem5 with NVMain to support the simulation of persistent memory by implementing the `clwb` and `sfence` instructions. We evaluate SecPM using two storage benchmarks including a hash table based key-value store and a B-tree based key-value store, like existing work [24]. We compare the proposed *SecPM* with an insecure persistent memory without memory encryption (*Insec-PM*) and the *SecPM w/o CWR* which indicates the SecPM without the proposed CWR scheme.

2) Results. We vary the transaction request sizes sent to the two key-value stores from 64B to 4KB to evaluate the performance in terms of the number of NVM writes and transaction throughput.

• ***The number of NVM write requests.*** Figure 8 shows the number of write requests to NVM normalized to that of Insec-PM in the two key-value stores. We observe the SecPM w/o CWR achieves the security and consistency but incurs about 2 times writes compared with Insec-PM, whatever the transaction request size is. Compared with the SecPM w/o CWR, SecPM significantly reduces the NVM writes. When the request size is 64B, SecPM reduces about 26% of writes, since the transaction data writes have no locality while the log writes have locality. When the request size is larger than 256B, the locality of data writes increase, and SecPM reduces about half of NVM writes. Compared with Insec-PM, SecPM only causes 13%, 5%, and 2% more writes when the request size is 256B, 1KB, and 4KB, respectively.

• ***Transaction throughput.*** Figure 9 shows the transaction throughput for the hash table and B-tree based key-value stores. Compared with InsecPM, SecPM incurs a little throughput reduction, due to the more NVM writes and the latency overhead of data encryption. Compared with the SecPM w/o CWR, SecPM significantly increases the throughput by 1.4 $\sim$ 2.1 times, due to the reduction in the number of write requests.

5 Related Work

• ***Secure NVM.*** Existing schemes on the secure NVM including DEUCE [36], BLE [14], SECRET [28], and Silent Shredder [7], mainly focus on reducing the bit writes of encrypted data to NVM, which do not consider the crash consistency in secure NVM.

• ***Crash consistency in NVM.*** To achieve data persistence, various durable transaction systems, such as, NV-Heaps [11], Mnemosyne [31], DudeTM [17], NVML [4], and DCT [13], are proposed to manage persistent data with crash consistency. All these schemes are built on the un-encrypted NVM, without considering the memory encryption on NVM.

The most related work comes from Liu et al. [18] that focuses on the crash consistency of secure NVM, which proposes the selective counter-atomicity to ensure that data and its counter are persisted in an atomic manner. However, significant software and hardware modifications are needed to implement selective counter-atomicity. Unlike it, SecPM performs only slight modifications on the memory controller to achieve crash consistency which are transparent for programmers. Thus programs running on an un-encrypted NVM can be directly executed on a secure NVM with SecPM.

6 Conclusion

This paper proposes SecPM to achieve both the security and persistence in non-volatile main memory. In SecPM, a counter cache write-through scheme and a counter write reduction scheme are respectively introduced to ensure the crash consistency and improve the system performance with slight modifications only on the hardware layer. Our evaluation shows that SecPM reduces up to half of write requests, and improves the transaction throughput by 1.4 $\sim$ 2.1 times, via the CWR scheme.

Acknowledgements

This work is supported by National Natural Science Foundation of China (NSFC) under Grant No. 61772212.

References

[1] Introducing Intel Optane Technology - Bringing 3D XPoint Memory to Storage and Memory Products. https://newsroom.intel.com/press-kits/introducing-intel-optane-technology-bringing-3d-xpoint-memory-to-storage-and-memory-products/, 2015.

[2] Deprecating the PCOMMIT Instruction. https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction, 2016.

[3] Intel® Architecture Instruction Set Extensions and Future Features Programming Reference. https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf, 2017.

[4] Intel Corporation-Persistent memory programming. http://pmem.io/, 2018.

[5] AKINAGA, H., AND SHIMA, H. Resistive random access memory (ReRAM) based on metal oxides. *Proceedings of the IEEE 98*, 12 (2010), 2237–2251.

[6] APALKOV, D., KHVALKOVSKIY, A., WATTS, S., NIKITIN, V., TANG, X., LOTTIS, D., MOON, K., LUO, X., CHEN, E., ONG, A., ET AL. Spin-transfer torque magnetic random access memory (STT-MRAM). *ACM Journal on Emerging Technologies in Computing Systems (JETC) 9*, 2 (2013), 13.

[7] AWAD, A., MANADHATA, P., HABER, S., SOLIHIN, Y., AND HORNE, W. Silent Shredder: Zero-cost shredding for secure non-volatile main memory controllers. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2016).

[8] BINKERT, N., BECKMANN, B., BLACK, G., REINHARDT, S. K., SAIDI, A., BASU, A., HESTNESS, J., HOWER, D. R., KRISHNA, T., SARDASHTI, S., ET AL. The gem5 simulator. *ACM SIGARCH Computer Architecture News 39*, 2 (2011), 1–7.

[9] CHHABRA, S., AND SOLIHIN, Y. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)* (2011).

[10] CHOI, Y., SONG, I., PARK, M.-H., CHUNG, H., CHANG, S., CHO, B., KIM, J., OH, Y., KWON, D., SUNWOO, J., ET AL. A 20nm 1.8 v 8gb pram with 40mb/s program bandwidth. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)* (2012).

[11] COBURN, J., CAULFIELD, A. M., AKEL, A., GRUPP, L. M., GUPTA, R. K., JHALA, R., AND SWANSON, S. 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)* (2011).

[12] DAVID, K., JEREMY, P., AND TOM, W. AMD Memory Encryption. *AMD White Paper* (2016).

[13] KOLLI, A., PELLEY, S., SAIDI, A., CHEN, P. M., AND WENISCH, T. F. High-performance transactions for persistent memories. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2016).

[14] KONG, J., AND ZHOU, H. Improving privacy and lifetime of pcm-based main memory. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2010).

[15] LEE, S. K., LIM, K. H., SONG, H., NAM, B., AND NOH, S. H. WORT: Write Optimal Radix Tree for Persistent Memory Storage Systems. In *Proceeding of the USENIX Conference on File and Storage Technologies (FAST)* (2017).

[16] 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).

[17] LIU, M., ZHANG, M., CHEN, K., QIAN, X., WU, Y., ZHENG, W., AND REN, J. DUDETM: Building Durable Transactions with Decoupling for Persistent Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2017).

[18] LIU, S., KOLLI, A., REN, J., AND KHAN, S. Crash consistency in encrypted non-volatile main memory systems. In *Proceedings of the IEEE 24th International Symposium on High-Performance Computer Architecture (HPCA)* (2018).

[19] MUELLER, W., AICHMAYR, G., BERGNER, W., ERBEN, E., HECHT, T., KAPTEYN, C., KERSCH, A., KUDELKA, S., LAU, F., LUETZEN, J., ET AL. Challenges for the dram cell scaling to 40nm. In *IEEE International Electron Devices Meeting (IEDM)* (2005).

[20] PELLEY, S., CHEN, P. M., AND WENISCH, T. F. Memory persistency. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)* (2014).

[21] POREMBA, M., ZHANG, T., AND XIE, Y. Nvmain 2.0: A user-friendly memory simulator to model (non-) volatile memory systems. *IEEE Computer Architecture Letters 14*, 2 (2015), 140–143.

[22] QURESHI, M. K., FRANCESCHINI, M. M., AND LASTRAS-MONTANO, L. A. Improving read performance of phase change memories via write cancellation and write pausing. In *Proceedings of the IEEE 16th International Symposium on High-Performance Computer Architecture (HPCA)* (2010).

[23] QURESHI, M. K., KARIDIS, J., FRANCESCHINI, M., SRINIVASAN, V., LASTRAS, L., AND ABALI, B. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2009).

[24] REN, J., ZHAO, J., KHAN, S., CHOI, J., WU, Y., AND MUTLU, O. Thynvm: Enabling software-transparent crash consistency in persistent memory systems. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)* (2015).

[25] SEO, J., KIM, W.-H., BAEK, W., NAM, B., AND NOH, S. H. Failure-atomic slotted paging for persistent memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2017).

[26] SHI, W., LEE, H.-H. S., GHOSH, M., LU, C., AND BOLDYREVA, A. High efficiency counter mode security architecture via prediction and precomputation. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)* (2005).

[27] SHIN, S., TIRUKKOVALLURI, S. K., TUCK, J., AND SOLIHIN, Y. Proteus: a flexible and fast software supported hardware logging approach for nvm. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2017).

[28] SWAMI, S., RAKSHIT, J., AND MOHANRAM, K. SECRET: smartly encrypted energy efficient non-volatile memories. In *Proceedings of the 53rd Annual Design Automation Conference (DAC)* (2016).

[29] THOZIYOOR, S., AHN, J. H., MONCHIERO, M., BROCKMAN, J. B., AND JOUPPI, N. P. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *International Symposium on Computer Architecture (ISCA)* (2008).

[30] VOLOS, H., MAGALHAES, G., CHERKASOVA, L., AND LI, J. Quartz: A lightweight performance emulator for persistent memory software. In *Proceedings of the 16th Annual Middleware Conference (Middleware)* (2015).

[31] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2011).

[32] WONG, H.-S. P., RAOUX, S., KIM, S., LIANG, J., REIFENBERG, J. P., RAJENDRAN, B., ASHEGHI, M., AND GOODSON, K. E. Phase change memory. *Proceedings of the IEEE 98*, 12 (2010), 2201–2227.

[33] XU, J., AND SWANSON, S. 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 (FAST)* (2016).

[34] YAN, C., ENGLENDER, D., PRVULOVIC, M., ROGERS, B., AND SOLIHIN, Y. Improving cost, performance, and security of memory encryption and authentication. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)* (2006).

[35] YANG, J., WEI, Q., CHEN, C., WANG, C., YONG, K. L., AND HE, B. NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems. In *Proceeding of the USENIX Conference on File and Storage Technologies (FAST)* (2015).

[36] YOUNG, V., NAIR, P. J., AND QURESHI, M. K. DEUCE: Write-efficient encryption for non-volatile memories. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2015).

[37] ZHOU, P., ZHAO, B., YANG, J., AND ZHANG, Y. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)* (2009).

[38] ZUO, P., AND HUA, Y. A write-friendly hashing scheme for non-volatile memory systems. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST)* (2017).