COVERT: Counter OVErflow ReducTion for Efficient Encryption of Non-Volatile Memories

Shivam Swami and Kartik Mohanram
Department of Electrical and Computer Engineering, University of Pittsburgh, PA
shs173@pitt.edu kartik.mohanram@gmail.com

Abstract

Security vulnerabilities arising from data persistence in emerging non-volatile memories (NVMs) necessitate memory encryption to ensure data security. Whereas counter mode encryption (CME) is a stop-gap practical approach to address this concern, it suffers from frequent memory re-encryption (system freeze) for small-sized counters and poor system performance for large-sized counters. CME thus imposes heavy overheads on memory, system performance, and system availability in practice. We propose Counter OVErflow ReducTion (COVERT), a CME-based memory encryption solution that performs on-demand memory allocation to reduce the memory encryption frequency of fast growing counters, while also retaining the area/performance benefits of small-sized counters. Our full-system simulations of a phase change memory (PCM) architecture across SPEC CPU2006 benchmarks show that for equivalent overhead and no impact to performance, COVERT simultaneously reduces the full memory re-encryption frequency from 6 minutes to 25 hours and doubles memory lifetime in comparison to state-of-the-art CME techniques.

1. Introduction

The high power consumption and poor potential for technology scaling of DRAM below 22nm [1] has spurred research in emerging non-volatile memories (NVMs) such as phase change memory (PCM) and resistive RAM (RRAM) [2–5]. Whereas PCM/RRAM NVMs have emerged as promising DRAM replacements, data persistence in PCM/RRAM NVMs poses many security vulnerabilities that must be addressed prior to commercialization of these advanced NVM technologies [6–10].

A broad class of memory encryption techniques [6–14] have been proposed in the literature to address memory security. State-of-the-art encryption techniques [7–10] employ counter mode encryption (CME), which associates a counter with each cache line and uses this counter along with the cache line's memory address and a secret key to generate a one-time pad (OTP). The OTP is bitwise XORed with the cache line's data (plaintext) to generate encrypted data (ciphertext), which is written to the memory. The counter for the cache line being written is incremented on every write to ensure temporal exclusivity of the OTP. During decryption, the same OTP is XORed with the ciphertext to generate the plaintext. CME is integrated inside the processor-side memory controller (assuming a secure processor) to ensure security against memory attacks like the stolen DIMM attack and the bus snooping attack [8–10, 12–14]. Although CME is secure against these attacks, it incurs the memory overhead of counters and negatively impacts system performance due to high OTP generation latency.

To address the high OTP generation latency and improve the system performance of CME, a counter cache is integrated inside the processor-side memory controller to cache frequently accessed counters [8, 12, 14]. During a read, a counter cache hit enables parallel generation of the OTP with the read of the ciphertext from the memory, whereas a counter cache miss precipitates a read request to fetch the counter and delays OTP generation [13, 14]. For a fixed size counter cache, the counter size should be small to improve the hit rate of the counter cache. Whereas small counters incur low memory overhead and improve system performance, they can quickly overflow in the presence of high write traffic. On a counter overflow, CME requires the entire memory to be re-encrypted with a new secret key, causing the system to freeze for the duration of full memory re-encryption. CME thus imposes heavy overheads on memory, system performance, and system availability in practice.

This paper proposes Counter OVErflow ReducTion (COVERT), a practical solution to realize low memory overhead CME without compromising system availability and/or performance. At its core, COVERT employs dynamic counters (DYNAMO henceforth) to reduce frequent full memory re-encryption due to small-sized counters. DYNAMO leverages the fact that a significant fraction of memory provisioned for error correction remains unutilized till very late in memory lifetime [15, 16]. DYNAMO repurposes unused error correction memory cells to the overflowing counters, thereby delaying the mandatory full memory re-encryption on a counter overflow and improving system availability. It is important to note that COVERT is a drop-in replacement for classical CME, and does not compromise the security of the underlying CME (i.e., COVERT preserves CME requirements of (i) encryption inside a secure processor and (ii) spatial/temporal exclusivity of the OTP).

We evaluate COVERT on a phase change random access memory (PCRAM) architecture [17] using the MARSS full-system simulator [18] on both integer and floating-point workloads from the SPEC CPU2006 benchmark suite [19]. Our implementation of COVERT employs 16-bit CME as the underlying encryption technique[1]. Our results show that for equivalent overhead and no impact to performance, COVERT simultaneously reduces the full memory re-encryption frequency from 6 mins to 25 hours and doubles memory lifetime in comparison to state-of-the-art 16-bit CME (e.g., memory encryption control unit (MECU) [7]).

2. Background and motivation

This section covers security issues in NVMs and explains how CME is an effective means to safeguard NVMs from various attacks. Furthermore, we discuss various trade-offs involved in CME and build motivation for COVERT.

2.1 Attack models and security of NVMs

The most common security vulnerabilities in NVMs take the form of stolen DIMM attacks and bus snooping attacks [6, 8–10]. In the stolen DIMM attack, the attacker has physical access to the NVM DIMM, enabling them to stream data from the DIMM. Data persistence of NVMs exposes data in the plaintext to attackers on power down. In the bus snooping attack, the attacker can acquire data by monitoring unsecured off-chip communications. It is widely accepted that such attacks can be thwarted by implementing data encryption in the secure processor.

This research was supported by NSF Award CCF-1217738.

[1] In this work, n-bit CME refers to a CME architecture that employs an n-bit counter per cache line. It does not refer to the length of the secret key used for encryption.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

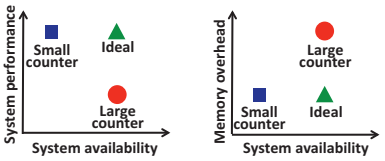

Figure 1: **This figure contrasts the tradeoffs of CME. Whereas high system performance and low memory overhead require small counters, high system availability (reduced counter overflow) requires large counters.**

Although data encryption in NVMs can be achieved by applying a block cipher to the plaintext to transform it into the ciphertext using a secret key, this direct encryption is latency intensive. Recent work [8–10, 12, 14] has advocated the use of counter mode encryption (CME) as a highly-secure low-latency main memory encryption technique. COVERT leverages and extends the capabilities of CME to ensure data security without incurring the overheads (discussed in Sec. 2.2) of conventional CME.

2.2 Counter mode encryption (CME)

In CME, a block cipher is used to encrypt a seed with a secret key (stored on the processor) to produce a one-time pad (OTP). This OTP is bitwise XORed with the plaintext to generate the ciphertext. During decryption, the same OTP is XORed with the ciphertext to obtain the plaintext. The spatial and temporal exclusivity of the OTP is critical for the security of CME [20], which implies that the OTPs should be unique for different memory blocks, and also for a particular block over time. These unique OTPs are generated from unique seeds that have two components: (i) the line address of the memory block (to ensure spatial exclusivity) and (ii) a counter which is incremented on each write (to ensure temporal exclusivity). Previous work [20] has shown that the counters can be stored in the plaintext in memory because the attacker still needs the secret key to regenerate the OTP.

To reduce CME latency, a counter cache is integrated inside the processor-side memory controller [12]. The counter cache stores counters belonging to frequently accessed cache lines. On a read, if the counter for the requested cache line is available in the counter cache, CME overlaps the OTP generation with off-chip data fetch, thereby hiding the decryption latency of the cache line. However, if the counter is not available in the counter cache, the OTP generation is delayed until the counter is fetched on-chip, which increases the total time for cache line decryption. Similarly, during a write, a counter cache hit reduces the encryption latency by eliminating the off-chip counter fetch.

To increase the counter cache hit rate, small counters are preferred in practice, since a fixed size counter cache can store more counters if each counter is small. However, small counters can quickly overflow in the presence of high memory write traffic. Conventionally, a counter overflow is handled by changing the secret key to prevent reuse of OTPs [12–14, 21]. However, since the same secret key is shared by every cache line, a change of secret key requires the entire memory to be re-encrypted, causing the system to freeze for the duration of full memory re-encryption. Re-encryption involves reading, decrypting, encrypting, and writing back every cache line in the memory. For the latest PCM prototype (read = 75ns, write = 150ns) [17], re-encryption of 16GB memory takes $\approx$ 1 minute (256 million 64-byte cache lines $\times$ 225ns re-encryption per cache line). To reduce counter overflow and increase system availability, large counters are used for encryption, since they do not overflow frequently. However, large counters increase the memory overhead of CME and result in poor system performance due to frequent counter cache misses. CME thus imposes heavy overheads on memory, performance, and system availability in practice, as shown in Fig. 1.

3. COVERT

The goal of this work is to enable low-overhead secure NVMs, without compromising system performance/availability. In this section, we describe COVERT, a CME-based memory encryption solution that performs on-demand memory allocation to reduce the memory encryption frequency of fast growing counters, while also retaining the area/performance benefits of small-sized counters. Although we consider PCM as the representative NVM in this work, the solutions developed here are applicable to other NVM technologies such as RRAM and spin-transfer torque RAM (STT-RAM).

3.1 COVERT: Dynamic counter (DYNAMO)

COVERT introduces DYNAMO, a technique to reduce full memory re-encryption rate resulting from counter overflow in conventional CME. To improve system availability without incurring area and performance penalty of large counters, DYNAMO performs on-demand memory allocation to the overflowing counters. By providing extra memory to an overflowing counter, DYNAMO allows continued operation based on the overflowed counter, thereby avoiding counter reset, and hence delaying full memory re-encryption.

Observation: Emerging NVMs suffer from low write endurance, resulting in early cell failures (PCM cells typically fail after 10^{8-10} writes [2–5]) that result in hard errors. To improve NVM reliability, architects employ error-correcting pointers (ECP) [15, 16], which uses a pointer to point to a failed cell and also stores the correct value for that cell. The standard ECP implementation reserves six ECPs in memory for every 512-bit memory block (i.e., per cache line), enabling each cache line to recover from up to 6 hard errors (ECP-6 henceforth). In terms of the pages maintained by the operating system, this translates to 384 ECPs per 4kB page (6 ECPs per 512-bit cache line $\times$ 64 cache lines). However, uniform ECP allocation across the memory leads to under-utilization of ECPs, since different cache lines in a page have different error correction requirements. For example, in [16], it is reported that very few cache lines ($<$1%) in a page use all the 6 available ECPs at the time of page failure[2]. **Thus, a significant portion of the memory provisioned for ECPs remains unutilized on a failed page and can be repurposed till such time they are necessary for error correction.** Without loss of generality, we consider ECP-6 as the underlying error correction technique; under-utilization of error correction resources is observed in other error correction techniques also.

DYNAMO design: DYNAMO leverages unused ECPs available in memory to extend the overflowing counters. When DYNAMO detects a counter overflow on a write to memory, it determines if unused ECPs are available for the memory block where the cache line is to be written. Each 512-bit cache line maps to 512 single-level cells (SLCs) in the memory. To detect unused ECPs, DYNAMO leverages the ECP-full flag, which is already provisioned in the standard ECP-6 design to indicate if a memory block has consumed all 6 ECPs. A reset/set ECP-full flag indicates available/exhausted ECPs. If unused ECPs are available, DYNAMO allocates one unused ECP to the overflowing counter and sets a DYNAMO flag (DYN-flag) to indicate counter extension for the cache line. This is shown in Fig. 2, where the last ECP, i.e., ECP_6 is allocated to the overflowing counter and the DYN-flag is set to 1. DYN-flag indicates number of ECPs available per 512-bit memory block; DYN-flag = 1 implies 5 ECPs, DYN-flag = 0 implies 6

[2]Techniques like PAYG [16] and Zombie memory [22] improve ECP utilization at the cost of system performance, and still result in more than 80% of cache lines possessing unused ECPs on a failed page. Since encryption is itself latency intensive, we do not expect latency-intensive error correction techniques like PAYG/Zombie memory to be integrated with NVM encryption in immediate future.

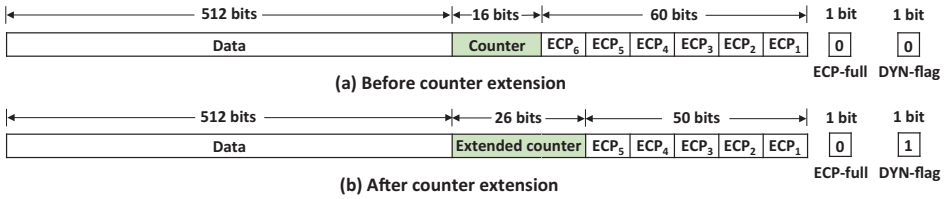

Figure 2: **Illustration of counter extension using DYNAMO. An overflowing counter is allocated an unused ECP and a DYN-flag is set to indicate the counter extension. This reduces the total available ECPs for that memory address to 5.**

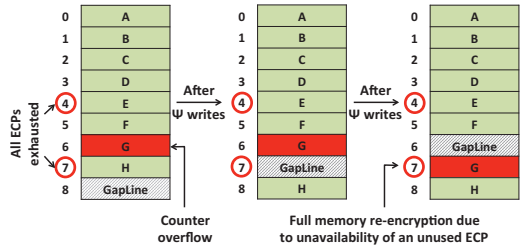

Figure 3: **A snapshot of memory in the presence of Start-Gap wear leveling [23] and DYNAMO. Start-Gap wear leveling maps a cache line to different physical addresses to achieve uniform wear across memory. Note that full memory re-encryption is required only when a cache line with an overflowing counter gets mapped to a physical address with no unused ECP. Following [23], we assume ψ = 100.**

ECPs. Assigning one unused ECP increases the size of an overflowing counter by 10 bits (we use only 8 bits and reserve 2 bits for future use). Hence, the effective counter size after extension is 24 bits and it delays counter overflow by 16 million (2^{24} - 2^{16}) writes.

Leveraging wear leveling for DYNAMO: It is common to perform wear leveling to achieve a uniform distribution of writes across NVMs, which prevents the more frequently written memory cells from failing early in comparison to other cells. Thus, wear leveling becomes necessary for NVMs that suffer from low endurance. Start-Gap [23] is a state-of-the-art wear leveling technique that incurs insignificant memory overhead (one spare cache line termed the GapLine) to achieve $\approx 97\%$ wear leveling. This work assumes that Start-Gap wear leveling is implemented in memory.

The frequently written cache lines, i.e., the cache lines with fast-growing counters experience more cell failures and are quite likely to exhaust their available ECPs, rendering DYNAMO ineffective. However, in the presence of Start-Gap wear leveling, a cache line gets mapped to multiple memory locations, which increases the probability of finding unused ECPs for frequently written cache lines. The full memory re-encryption occurs only when a cache line with an overflowing counter gets mapped to a memory address with no unused ECPs. As shown in Fig. 3, memory addresses 4 and 7 do not possess unused ECPs and cache line 'G' has an overflowing counter. When 'G' gets mapped to address 7, due to the unavailability of an unused ECP, full memory re-encryption is required. Hence, the probability of full memory re-encryption due to unavailability of ECPs is pq, where p is the probability of a memory address exhausting all 6 ECPs and q is the probability of a counter overflow. The value of p is $\approx 1\%$ [16] and q is $\approx 30\%$ (based on our simulations on SPEC CPU2006 benchmarks, which closely resemble real-world applications [19]). This implies that the probability of finding unused ECPs for counter extension is greater than 99% $(1 - pq)$ in practice.

Memory operation: On a memory write, if a counter overflows, DYNAMO is invoked to perform counter extension and the DYN-flag for that memory address is set to 1. However, if counter extension is not feasible, full memory re-encryption is performed using a new secret key and all the counters are reset to 0. Full memory re-encryption involves reading, decrypting, encrypting, and writing back every cache line in the memory, causing the system to freeze for the duration of full memory re-encryption.

Security: The security of CME against bus snooping and stolen DIMM attacks requires that (i) the encryption is performed inside a secure processor and (ii) a given OTP is never reused. The modification (DYNAMO) introduced by COVERT does not compromise these requirements, making COVERT as secure as CME.

4. Evaluation and Results

We evaluate COVERT on a phase change random access memory (PCRAM) architecture using the MARSS full-system simulator [18] on both integer and floating point workloads from the SPEC CPU2006 benchmark suite [19]. We employ 16-bit CME as the underlying encryption technique in COVERT.

Simulation framework: MARSS is configured to simulate a standard 4-core out-of-order system running at 3GHz. Each core is assigned a private L1 instruction/data cache of 32kB (latency = 2ns) and a private L2 cache of 128kB (latency = 5ns). Finally, L3 is a single, shared write-back cache of 8MB (latency = 20ns). The main memory is modeled as a 16GB, 8 banks, single channel PCRAM with read latency = 75ns and write latency = 150ns [9,17]. Furthermore, we implement 8-bit, 16-bit, 24-bit, 32-bit, and 64-bit CME using a fully pipelined 128-bit advanced encryption standard (AES) crypto-engine (OTP generation latency = 72ns [13]). We integrate a 512kB 32-way set-associative counter cache [13] inside the memory controller for all the techniques evaluated in this work. Note that we statically split the counter cache for COVERT into two parts: (i) 384kB 16-bit counter cache for storing a majority of non-overflowing 16-bit counters and (ii) 128kB 24-bit counter cache for storing a small number of overflowing counters that are extended from 16 bits to 24 bits using DYNAMO. Although it is possible to realize a split counter cache architecture that dynamically varies the counter cache block size in response to dynamic counter extension [24, 25], the implementation details and results for that architecture are not discussed here for brevity.

4.1 Summary of results

Table 1 summarizes the system performance (measured in instructions per cycle (IPC)) and re-encryption frequency of COVERT and other CME implementations. IPC is normalized to ideal CME, i.e., CME with zero latency overhead for OTP generation. We compare COVERT with state-of-the-art MECU [7] and DEUCE [9]. MECU (DEUCE) is designed with a 16-bit (32-bit) counter per 512-bit cache line, incurring a memory overhead of 3.12% (6.25%). To establish bounds on encryption rate and system performance of CME, we also evaluate 8-bit and 64-bit CME. **Our results show that COVERT requires a full system re-encryption only once in 25.7 hours (i.e., 250$\times$ improvement over 16-bit CME like MECU) with no penalty on system performance (i.e., COVERT and MECU have the same IPC).**

Technique	Counter size	IPC	Re-encryption rate	Memory overhead
8-bit CME	8 bits	0.84	1.39 (sec)	1.56%
64-bit CME	64 bits	0.77	millennia	12.5%
MECU [7]	16 bits	0.82	6.0 (min)	3.12%
DEUCE [9]	32 bits	0.79	274 (days)	6.25%
COVERT	**16 bits**	**0.82**	**25.7 (hr)**	**3.32%**

Table 1: **Summary of IPC, re-encryption rate, and memory overhead of COVERT and other CME techniques. COVERT provides the optimal trade-off between performance (IPC), system availability, and memory overhead.**

4.2 COVERT: Re-encryption rate

In this section, we evaluate the re-encryption rate of COVERT for applications from the SPEC CPU2006 [19] benchmark suite. Following [14], we track the growth rate of the fastest growing counter for each application and use these growth rates to estimate the re-encryption frequency in long-running applications. Furthermore, we assumed a constant write-back rate of 40MB/s. Note that although the data transfer rate of DDR3 is 6400MB/s, the PCRAM prototype available has a maximum write throughput of only 40MB/s [17].

In Fig. 4, we report the re-encryption rates for different CME techniques. Whereas the re-encryption rate reduces with an increase in the counter size (1.39 seconds for 8-bit CME versus 1000 millennia for 64-bit CME), it comes at the cost of high memory overhead (1.5% for 8-bit CME versus 12.5% for 64-bit CME) and poor system performance (IPC of 64-bit CME is 10% lower than 8-bit CME). By employing DYNAMO for counter extension, COVERT reduces the re-encryption frequency of a 16-bit CME (e.g., MECU) from every 6 minutes to every 25 hours (i.e., 250× improvement), for an additional memory overhead of only 0.19% (1 bit DYN-flag per 512-bit cache line). Note that scheduling a minute-long full memory re-encryption once in 25 hours (≈ 99.93% system availability) is feasible for a majority of systems in practice. However, for systems that are expected to provide higher availability (e.g., web/database servers), COVERT can be implemented with 24-bit CME to achieve 99.999% system availability (using DYNAMO) with minimal impact to system performance and only 4.8% total memory overhead. In contrast, DEUCE [9] achieves equivalent system availability for 20% increase in counter memory (800MB for COVERT versus 1GB for DEUCE) and 1-2% reduction in system performance over COVERT.

4.3 COVERT: Lifetime improvements

COVERT improves memory lifetime by reducing the memory wear attributed to full memory re-encryption. To compute the reduction in memory wear, we computed the total number of writes (due to write-back and full memory re-encryption) experienced by the memory in one re-encryption interval. When the re-encryption interval is large, writes due to full memory re-encryption are reduced, which in turn improves endurance. In our computations, we assumed a constant write-back rate of 40MB/s and total writes on re-encryption as 256 million (since 16GB memory = 256 million cache lines). Our results show that COVERT improves memory lifetime by 1.9× over 16-bit CME (e.g., MECU).

5. Conclusions

Memory encryption is required to address security vulnerabilities in NVM-based main memories. Counter mode encryption (CME), a state-of-the-art main memory encryption technique, imposes overheads on memory, system performance, and system availability. COVERT is the first paper to target these problems of CME for NVMs. COVERT employs DYNAMO, which utilizes unused error correction resources to reduce memory re-encryption frequency for insignificant memory overhead and no impact on system performance of the underlying CME architecture.

Benchmark	8-bit CME (seconds)	64-bit CME (millennia)	MECU [7] (minutes)	DEUCE [9] (days)	COVERT (hours)
astar	4.79	> 1000	20.4	931	87.4
bzip2	1.76	> 1000	7.52	342	32.1
gcc	3.33	> 1000	14.2	646	60.6
namd	0.29	> 1000	1.27	57.8	5.42
omnetpp	0.38	> 1000	1.65	75.3	7.06
xalancbmk	0.12	> 1000	0.54	24.8	2.32
soplex	4.55	> 1000	19.4	886	83.1
perlbench	0.33	> 1000	1.42	65	6.08
GemsFDTD	35.3	> 1000	150	6858	643
Geometric mean	**1.39**	**> 1000**	**6.0**	**274**	**25.7**

Figure 4: **Full memory re-encryption frequency for different CME techniques. Large counters reduce the full memory re-encryption rate at the cost of increased memory overhead and poor system performance.**

References

[1] *International Technology Roadmap for Semiconductors*, 2011.

[2] B. C. Lee *et al.*, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. Intl. Symposium on Computer Architecture*, 2009.

[3] B. C. Lee *et al.*, "Phase change technology and the future of main memory," *IEEE Micro*, 2010.

[4] B. Govoreanu *et al.*, "10× 10nm 2 Hf/HfO x crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *Proc. Intl. Electron Devices Meeting*, 2011.

[5] C. Xu *et al.*, "Understanding the trade-offs in multi-level cell ReRAM memory design," in *Proc. Design Automation Conference*, 2013.

[6] S. Chhabra *et al.*, "i-NVMM: a secure non-volatile main memory system with incremental encryption," in *Intl. Symposium on Computer Architecture*, 2011.

[7] W. Enck *et al.*, "Defending against attacks on main memory persistence," in *Annual Computer Security Applications Conference*, 2008.

[8] J. Kong *et al.*, "Improving privacy and lifetime of PCM-based main memory," in *Proc. Intl. Conference on Dependable Systems and Networks*, 2010.

[9] V. Young *et al.*, "DEUCE: Write-efficient encryption for non-volatile memories," in *Proc. Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[10] S. Swami *et al.*, "SECRET: Smartly encrypted energy efficient non-volatile memories," in *Proc. Design Automation Conference*, 2016.

[11] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC press, 1996.

[12] J. Yang *et al.*, "Improving memory encryption performance in secure processors," *IEEE Trans. Computers*, 2005.

[13] W. Shi *et al.*, "High efficiency counter mode security architecture via prediction and precomputation," in *Proc. Intl. Symposium on Computer Architecture*, 2005.

[14] C. Yan *et al.*, "Improving cost, performance, and security of memory encryption and authentication," in *Proc. Intl. Symposium on Computer Architecture*, 2006.

[15] S. Schechter *et al.*, "Use ECP, not ECC, for hard failures in resistive memories," in *Proc. Intl. Symposium Computer Architecture*, 2010.

[16] M. K. Qureshi, "Pay-As-You-Go: Low-overhead hard error correction for phase change memories," in *Proc. Intl. Symposium Microarchitecture*, 2011.

[17] Y. Choi *et al.*, "A 20nm 1.8 V 8Gb PRAM with 40MB/s program bandwidth," in *Proc. Intl. Solid-State Circuits Conference*, 2012.

[18] A. Patel, F. Afram, and K. Ghose, "MARSS: A full system simulator for multi-core x86 CPUs," in *Proc. Design Automation Conference*, 2011.

[19] J. L. Henning, "SPEC CPU2006 benchmark descriptions," in *ACM SIGARCH Computer Architecture News*, 2006.

[20] H. Lipmaa *et al.*, "CTR-mode encryption," in *NIST Workshop on Modes of Operation*, 2000.

[21] W. Shi *et al.*, "Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems," in *Proc. Intl. Conference on Parallel Architectures and Compilation Techniques*, 2004.

[22] R. Azevedo *et al.*, "Zombie Memory: Extending memory lifetime by reviving dead blocks," in *Proc. Intl. Symposium on Computer Architecture*, 2013.

[23] M. K. Qureshi *et al.*, "Enhancing lifetime and security of PCM-based main memory with Start-Gap wear leveling," in *Proc. Intl. Symposium on Microarchitecture*, 2009.

[24] C. Dubnicki and T. J. LeBlanc, "Adjustable block size coherent caches," in *Proc. Intl. Symposium on Computer Architecture*, 1992.

[25] K. Inoue, K. Kai, and K. Murakami, "Dynamically variable line-size cache exploiting high on-chip memory bandwidth of merged DRAM/logic LSIs," in *Proc. Intl. Symposium on High-Performance Computer Architecture*, 1999.