# Improving Privacy and Lifetime of PCM-based Main Memory

*Jingfei Kong*
*School of Computer Science*
*University of Central Florida*
*jfkong@eecs.ucf.edu*

*Huiyang Zhou*
*Dept of Electrical and Computer Engineering*
*North Carolina State University*
*hzhou@ncsu.edu*

## Abstract

*Phase change memory (PCM) is a promising technology for computer memory systems. However, the non-volatile nature of PCM poses serious threats to computer privacy. The low programming endurance of PCM devices also limits the lifetime of PCM-based main memory (PRAM). In this paper, we first adopt counter-mode encryption for privacy protection and show that encryption significantly reduces the effectiveness of some previously proposed wear-leveling techniques for PRAM. To mitigate such adverse impact, we propose simple, yet effective extensions to the encryption scheme. In addition, we propose to reuse the encryption counters as age counters and to dynamically adjust the strength of error correction code (ECC) to extend the lifetime of PRAM. Our experiments show that our mechanisms effectively achieve privacy protection and lifetime extension for PRAM with very low performance overhead.*

## 1. Introduction

Phase change memory (PCM) is an emerging memory technology, which features low access latency, byte-addressability, high integration density, and low leakage energy consumption. Recently, there have been strong interests in using PCM as main memory (PRAM) in computer systems [13][20][27] .

Despite of having the aforementioned strengths, PCM raises new challenges for computer system design. One key property of PCM is its non-volatility: the information stored in PRAM is persistent without power supply and may last for more than 10 years [18]. Although non-volatility is a fundamental reason for power efficiency, it makes PRAM much more vulnerable to malicious security attacks than volatile DRAM. Another important issue of PCM is its unreliability, particularly due to its limited write endurance[18], and various wear-leveling techniques have been proposed to extend the PRAM lifetime [4][13][19][20][25][26][27].

In this paper, we adopt a hardware encryption scheme, counter-mode encryption, to protect the privacy of PRAM, given its proven security and high performance. However, due to the diffusion characteristics of encryption algorithms, values in encrypted data blocks are randomized. This negates the effectiveness of some previously proposed wear-leveling techniques, redundant bit-write removal [27] and partial writes [13] in particular.

To mitigate the encryption impact upon wear-leveling techniques, we propose two new schemes. First, we propose simple yet effective extensions to the encryption scheme to revive partial writes. Second, we propose to use encryption counters as age counters and to dynamically adjust error protection strengths. In this work, we use error correction code (ECC) and design an efficient way to manage ECC protection. Our experiments using a cycle-accurate timing simulator show that the performance overhead introduced by encryption and ECC is small given the long latency of PRAM accesses.

Our main contributions include (1) to our knowledge, this is the first work to investigate the impact of encryption on wear-leveling techniques for PRAM; (2) we propose a new encryption counter scheme to revive partial writes to reduce write traffic; (3) we propose to leverage encryption counters as age counters and design an efficient way for ECC management to extend PRAM lifetime.

The remainder of the paper is organized as follows. In Section 2, we present background and discuss related work. In Section 3, we analyze the encryption impact on wear-leveling techniques and we propose a new encryption counter scheme to mitigate it. In Section 4, we present our design to leverage encryption counters as age counters for efficient ECC management. In Section 5, the overall architecture is presented and the interaction between encryption and ECC is discussed. Detailed experimental results are presented in Section 6. Section 7 concludes the paper.

## 2. Background and related work

In PCM, a memory cell can be transformed between a low-resistivity state and a high-resistivity state by atomic arrangements. Compare to FLASH memory which has the block-erase requirement, PCM is typically byte-addressable. Besides, PCM has better write endurance than FLASH and is projected to be more scalable, more cost-effective and of higher performance [20][27]. In this section, we discuss the challenges for using PCM as main memory (PRAM) and some can be applied to the FLASH technology as well.

## 2.1 Privacy

The non-volatility nature of PRAM aggravates the privacy concerns over the contents residing in main memory. It was reported that secret keys for disk encryption can still be retrieved from volatile DRAM even minutes after a computer is powered off [10]. Attackers may simply dump the plaintext image of PRAM to extract critical information. Such one-time storage dump is also a major security concern for disk storage systems and disk encryption is often used for security protection [8]. In a more strict security model, it is assumed that more complex security attacks exist and attackers have the abilities to monitor the dynamic data read from and stored to main memory. Protection against such advanced security attacks is addressed with secure processor architectures [14]. The security of those secure architectures relies on the assumption that processor cores are unbreakable. Secret keys involved are generated or sealed inside processors. Given the privacy issues with PRAM, we opt to use the counter mode encryption for its proved security and high performance [3] and assume that the secret keys are inside processors.

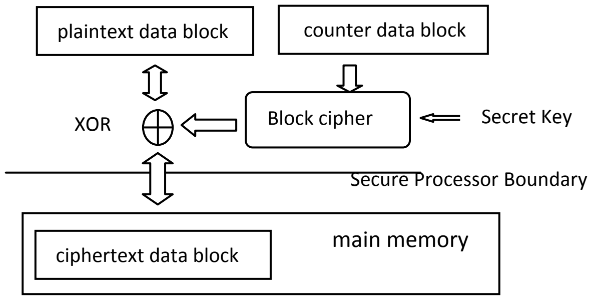

**Figure 1. Counter-mode encryption for secure processors**

Figure 1 shows the counter-mode encryption proposed for secure processors [3]. The counter data block is composed of a counter, two address offsets and a logical page identifier (LPID). The counter is associated with a cache line and is incremented by one every time when the cache line is written back to main memory. One address offset is the cache line offset within a page. The other is the plaintext data block offset within the cache line when the cache line size is larger than the size of an encryption/plaintext data block. The LPID is assigned uniquely for each allocated page in the main memory. The security strength of the counter-mode encryption comes from the fact that the value of the counter data block for each plaintext data block is unique both spatially and temporally. The LPID and address offsets provide spatial uniqueness while the counter ensures temporal uniqueness. For resource efficiency, counters of limited sizes are used. As a result, when a counter overflows, a new unique LPID is generated and the whole page would be re-encrypted to ensure uniqueness of counter data blocks [3]. The performance advantage of the counter-mode encryption is that the block cipher (i.e., encryption) latency can be overlapped with the latency of fetching the encrypted data block.

## 2.2 Wear-leveling techniques

Various wear-leveling techniques have been proposed to address the limited endurance of PRAM to extend its lifetime. The granularity of the approaches can be at the segment/page level, the cache-line level, and the individual bit level. They can be classified into two categories. One is using swapping/shifting/rotation to even the wear-out among hot (more write accessed) and cold (less write accessed) spots. For example, segment swapping [27] happens when one segment becomes so hot that it has to be swapped with some cold segment to even out wear-out. Similarly, in the start-gap scheme [19], each cache line would be rotated through the whole memory with the support from one spare cache line. The wear-leveling techniques for FLASH devices by manipulating the mapping between logical blocks and physical FLASH memory blocks also fall into this category. The other category is to reduce write traffic to PRAM. For example, line-level write back [20] only writes dirty cache lines instead of a whole page to PRAM. Similarly, partial writes [13] add dirty bits to track which words of a cache line are dirty. When the cache line is replaced, only the dirty words in the cache line have to be written back to PRAM. Redundant bit-write removal [27] first reads out and compares the existing content to the new content. Only those bits that differ are written back actually. Since many of the above approaches are orthogonal to each other, various schemes are often combined to achieve better lifetime improvement. The wear-leveling schemes based on reducing write traffic suit better for PRAM due to its byte-addressability than FLASH, in which the block-erase requirement makes it difficult to exploit bit-level or cache-line-level redundancy.

## 2.3 Reliability, lifetime and ECC

To store information, PCM material is heated to change state under electrical pulses. The repeated heat stress, however, could render PCM material unstable and unreliable. Data retention, endurance, program and read disturbs are some of the basic reliability aspects that have been investigated recently [12][18]. In this paper we assume that there exist failing mechanisms or process variations causing some PRAM cells fail earlier than others [9].

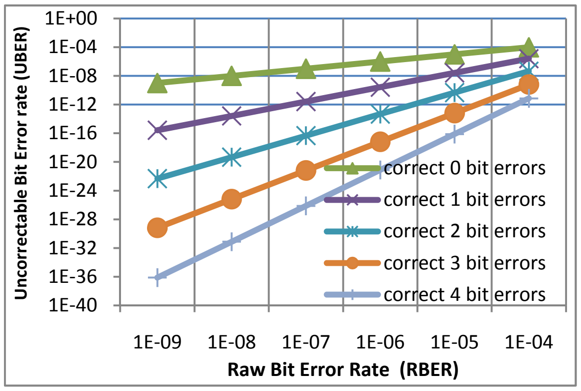

**Figure 2. The impact of ECC correctability on bit error rates (based on data block size of 64 bytes)**

Since PCM may have failures at some point, we need a scheme to detect and correct errors in PRAM. Error Correcting Code (ECC) is a common mechanism for such a purpose [11][16][24][26]. ECC achieves information redundancy by generating redundant bits based on the data to be protected. The more bit errors that need to be corrected, the more ECC bits are required. Figure 2 shows the bit error rate before (raw bit error rate) and after (uncorrectable bit error rate) ECC protection. The figure shows that ECC protection greatly reduces the bit error rate, often at a few orders of magnitude. The implication on PRAM is that the failure of PRAM would be postponed dramatically. For example, with the data traffic to PRAM at 4GB/second, without ECC protection, a PRAM with the raw bit error rate at $10^{-8}$ would have one bit error after around 3 milliseconds ($=1/(4G*8*10^{-8})$ seconds) of usage. With ECC capable of correcting 1 bit error, the uncorrectable bit error rate becomes $2.6*10^{-14}$, which means that the PRAM would not encounter one bit error until after almost 20 minutes ($=1/(4G*8*2.6*10^{-14})$ seconds) of usage.

## 3. Privacy Protection for PRAM

### 3.1 The impact of encryption on PRAM wear-leveling techniques

We argue that encryption is necessary for protecting privacy of non-volatile PRAM. However, to our knowledge, no prior work, especially the wear-leveling techniques, have investigated the impact of encryption. To analyze the impact, it is necessary to dissect certain characteristics of modern encryption algorithms.

Between two main classes of encryption algorithms, symmetric and asymmetric-key encryption, symmetric-key encryption is often chosen for its high-speed. Between stream ciphers and block ciphers in symmetric-key encryption, block ciphers are often used in memory/storage encryption. With the basic encryption unit being a data block, block ciphers produce an output block of the same length as an input block. The size of one block for encryption is often smaller than the size of a single cache line/block. For example, Advanced Encryption Standard (AES) [5]supports the encryption block size of 16 bytes while one last-level cache line/block may have 64 or more bytes. So, a cache line contains 4 or more encryption data blocks. To ensure security, there is one important principle for block cipher design – diffusion [15]. Diffusion tries to disperse the statistical characteristics of plaintexts into long spectrum of ciphertexts. In other words, each plaintext bit would affect many ciphertext bits. The result is that there is no relationship between two ciphertexts even there are just few different bits in their plaintexts.

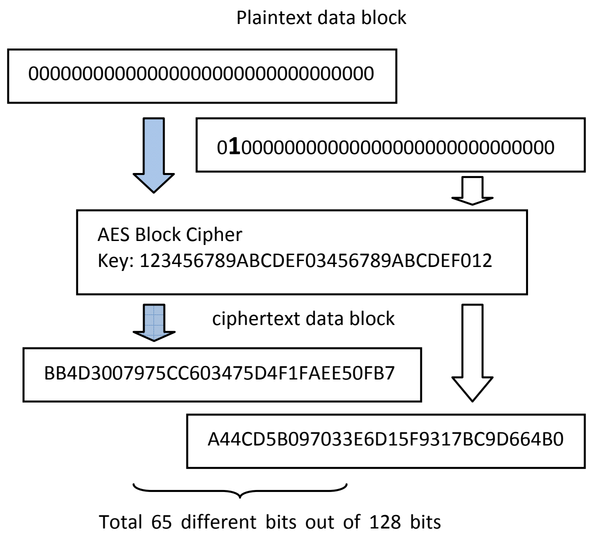
Plaintext data block
00000000000000000000000000000000
01000000000000000000000000000000
AES Block Cipher
Key: 123456789ABCDEF03456789ABCDEF012
ciphertext data block
BB4D3007975CC603475D4F1FAEE50FB7
A44CD5B097033E6D15F9317BC9D664B0
Total 65 different bits out of 128 bits

**Figure 3. Avalanche effect caused by encryption**

The diffusion characteristics have severe impacts on some previously proposed wear-leveling techniques. Since the encryption block size is less than the size of one cache line/block, encryption mainly affects wear-leveling techniques below the cache line granularity. Schemes on or above the cache line granularity such as segment swapping [27], the start-gap [19] and the line-level write-back [20] are not affected by encryption. For redundant bit-write removal [27], with encryption, the new ciphertext data values would be largely different from the old ciphertext data values. Figure 3 shows such an example with AES encryption. Even if the to-be-written-back plaintext data block is the same as the old one, the two ciphertext data blocks are largely different from each other due to the encryption counter update. Therefore the effectiveness of the redundant bit-write removal is greatly reduced. For the partial-write scheme [13], the problem with encryption is that the encryption counter for a cache line is incremented for every write back. As a result, the whole cache line needs to be re-encrypted and modified, thereby completely disabling partial writes even if there is only one dirty word in the cache line.

## 3.2 A new encryption counter scheme to mitigate the encryption impact

Based on the analysis in Section 3.1, we propose to extend the original encryption counter scheme to revive the partial-write wear leveling. Our extension includes additional counters at the encryption-block granularity. In other words, for each cache line, besides one cache-line-level counter, we add multiple block-level counters. Encryption for each data block is done using the combination of the cache-line-level counter and the block-level counter. Upon a write-back, only the block-level counters corresponding to dirty blocks are incremented and only the dirty blocks are re-encrypted. Other non-dirty blocks within the same cache line can therefore be spared from being written back to PRAM. When any block-level counter overflows, however, all block-level counters in the same cache line are reset to zero and the cache-line-level counter is incremented by one. In this case, the partial write scheme does not work as the whole cache line is re-encrypted. Note that since the basic encryption unit is an encryption block (typically 16 bytes), finer granularities (e.g., word size) for partial writes are not beneficial as the whole data block will be encrypted even if only one word in the block is updated.

In Section 6.2, we show the quantitative impact of encryption on the wear-leveling techniques. The effectiveness of our new encryption counter scheme is shown in Section 6.3. Note our new scheme revives partial writes to reduce write traffic to PRAM and it alone may not be sufficient to improve lifetime. Some rotation scheme is necessary to distribute write traffic evenly in order to take full advantage of the write traffic reduction.

## 4. Adaptive ECC management

A straightforward way to extend PRAM lifetime using ECC is allocating enough ECC storage to cover the maximum number of bit errors that are expected. However, during the most of the PRAM lifetime, the number of bit errors is much less than the expected maximum number. Therefore it is wasteful in space and potentially harmful to the performance. The reasons are (a) some memory space are allocated for unnecessary ECC storage and (b) the logic for correcting a high number of bit errors is slower than it for a small number of bit errors.

In this paper, we propose to dynamically manage ECC strength according to the reliability/wear-out status of PRAM. To keep track of the wear-out status of PRAM, we use the number of write-accesses to PRAM as a metric. We assume that the raw bit error rate increases as PRAM ages [9]. Therefore, we gradually increase the strength of ECC protection by allocating more ECC bits when PRAM is gradually worn out. Figure 4 shows one such example in which the number of bit errors to be corrected by ECC is

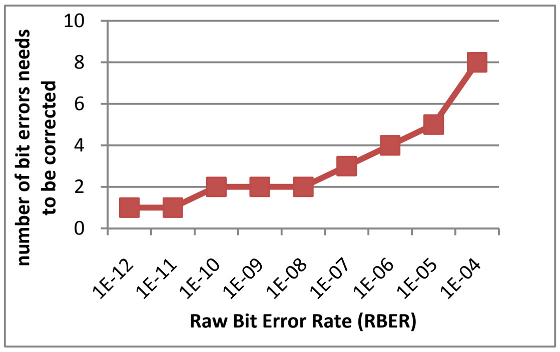

**Figure 4. Dynamic requirement for ECC to meet UBER of $10^{-19}$ as PRAM gets worn out**

increasing as the raw bit error rate is increasing. The target for the uncorrectable bit error rate (UBER) in Figure 4 is $10^{-19}$, which enables a PRAM with data traffic at 4GB/second to have just one bit error after around 10 years ($=1/(4G*8*10^{-19})$) of usage.

For efficient ECC management, two main challenges need to be addressed. The first is where to store the ECC bits as the size varies for different error correction requirements and how to access them accordingly. The second is to obtain the write-access counters as they are necessary for monitoring the wear-out status of PRAM.

## 4.1 The ECC space and address mapping

To accommodate dynamic ECC management, we propose a hardware-software integrated approach, shown in Figure 5.

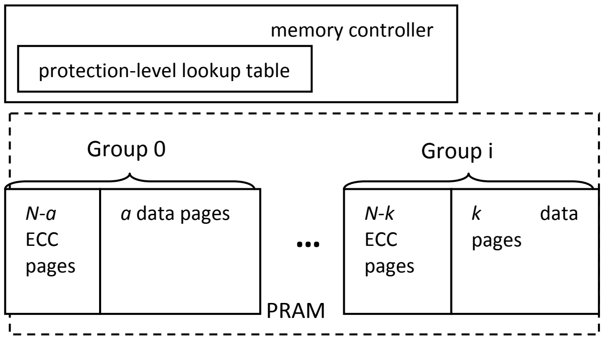

**Figure 5. The design of our proposed scheme for dynamic ECC management**

Instead of having separate memory chips for ECC, we propose to store both data and their ECC bits in a unified memory space (i.e., virtual memory space). The operation system (OS) will assist the dynamic allocation of ECC and data pages and maintain information of the ECC pages so that the data pages will not be mapped to the same places. The physical memory is partitioned into groups and each group contains multiple data pages and the corresponding ECC pages. All the data pages in the same group have the same level of ECC protection. The assumption is that some aforementioned rotation-based wear-leveling techniques

```
ECC_addr = ECC_base_addr + ECC_ offset_addr * ECC_size_for_one_cache_line
    ECC_base_addr =  data_cache_line_addr & group_size_mask  /*group_size_mask = ~(group_size – 1) */
    ECC_offset_addr =  [data_cache_line_addr & (~group_size_mask) – Data_base_addr_in_the_group] / (cache_line_size)
    ECC_size_for_one_cache_line = the number of ECC bits for the protection_level_for_the_group
   Data_base_add_in_the_group = number of ECC pages in the group * page size
```

**Figure 6. The address mapping of ECC protection bits**

are deployed and they introduce evenly or close to evenly distributed write traffic to PRAM. Our experiments are based on such an assumption. When some pages in a group are getting older and reach a threshold, more ECC space is required. In this case, an exception would happen and OS would intervene to reorganize the group to contain less data pages and more ECC pages. Note that since such exception event is rare, the introduced performance overhead would be negligible. Depending on workload write-traffic characteristics, the effectiveness of rotation-based wear-leveling to distribute the write traffic, and process variations, different groups may have different numbers of ECC pages according to their wear-out status. As shown in Figure 5, group 0 may have *a* data pages and (*N-a*) ECC pages while group *i* may have *k* data pages and (*N-k*) ECC pages. Eventually, the ratio of the number of ECC pages against the number of data pages in a group will reach the worst case, where the maximum number of bit errors is expected. In such a case, the group may be marked unreliable and the OS will not use it anymore.

Managing memory in groups simplifies address mapping to access the ECC bits. As shown in Figure 5, the memory controller has a protection-level look up table which contains the information of how many ECC pages exist in each group. Such information is then used to derive the number of ECC bits for a cache-line in the group. The space cost for the protection-level lookup table is small. For example, if each group contains 1K pages and the page size is 4KB, a 1K-entry table can manage a 4GB-PRAM. The process of determining the physical address of ECC bits of a data access is shown in Figure 6. First, the physical address of the data (*data_cache_line_addr* in Figure 6 as the unit of data operation is the cache line size) is used to decide which group contains the data (*ECC_base_addr* in Figure 6). Then, the offset within the group (*ECC_offset_addr*) is calculated to see where the data is located within the group at the cache line granularity. Based on how many ECC bits are allocated for each cache line, the address of the ECC bits is generated. Note that due to the use of the two's power numbers, the multiplication and division in Figure 6 can be implemented using simple shifts and adds. The group size is mainly dependent on the PRAM endurance characteristics. Since all the data in the same group have the same ECC, we essentially assume that the memory in a group will wear out at a similar rate. If the endurance variation is expected to be large, we prefer a small group size to avoid a small region to affect a large amount of memory. Note that although an ECC page is referenced more frequently than data pages (as each ECC page can accommodate multiple data pages), it is unlikely for ECC pages to become most worn out ones. The reasons are two-folds. First, as shown in Section 6.6, after encryption, both data and ECC code have similar bit-level redundancy and bit-level wear-leveling techniques have the similar effects. Second, inside an ECC page, each individual ECC block has the same number of writes as the corresponding data block (e.g., a cache line). So the lifetime of an ECC page will be the same as (or very close to) the data page which is most worn out.

Storing ECC bits in virtual memory space is independently proposed in a recent work [23]. An ECC page table is introduced in their scheme to locate the ECC pages.

## 4.2 Leveraging encryption counters as write-access (age) counters

There are several ways to track the number of write-accesses to memory pages in PRAM. One is to associate a local counter with each cache line. The maximum among all local counters in a page would be the age of the page. Another way is to have a single counter for one page and it records the total number of write-accesses to the page. The first approach incurs high space overhead (local counter size * number of lines in a page). The second approach has much less space overhead (just one counter) but less accurate since write-accesses to different lines in a page are accumulated, leading to a highly overestimated age. To reduce space overhead without losing much accuracy, we choose to use a two level counters scheme similar to the one used in [22].

In the counter-mode encryption described in [3], there already is a local counter (LC) for each line in a page. In addition, there is a 64-bit logical page identifier (LPID) assigned for each memory page when it is allocated. To account for a high number of write accesses, we add another global counter (GC) for each page. When a local counter overflows, GC would be incremented by one and all the local counters in the same page would be reset to zero. In this case, as discussed in Section 2.1, a new unique LPID is generated and the page would be re-encrypted to ensure security [3]. Such a combined two-level global and local counters are much more accurate than one global counter and have much less space overhead than the local counter scheme. Note as the overflow happens rarely, the associated performance overhead is negligible [22].

## 5. Overall Architecture

The overall architecture for improving privacy and lifetime is shown in Figure 7.

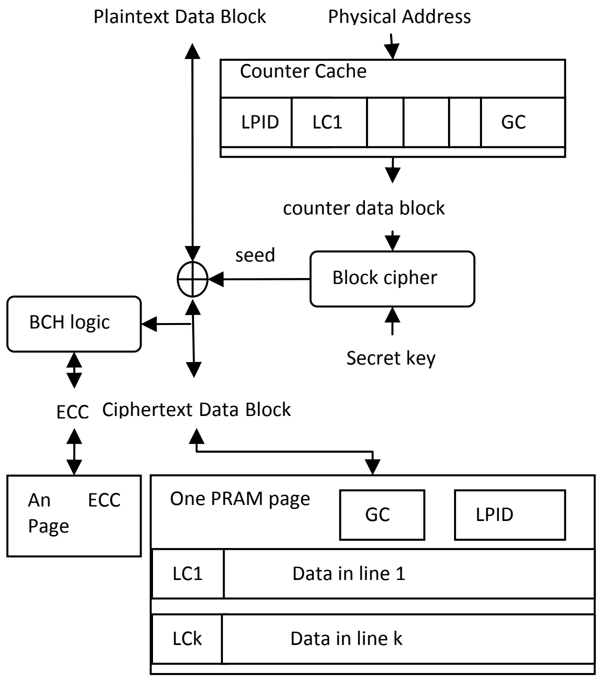

**Figure 7. A logical view on the architecture of the proposed scheme for privacy protection and lifetime improvement.**

With the proposed architecture, a memory access proceeds as follows. When an encrypted cache line is to be fetched from PRAM, its memory address is used to locate the corresponding counters and generate the seed for the block cipher. As the counters are stored along with the ciphertext data in the PRAM, directly accessing them will postpone the seed generation process and expose the block cipher latency. To overcome this performance issue and overlap this latency with PRAM access latency, a counter cache is included as in the previous work [3]. Here, note that if the counter scheme proposed in Section 3.2 is used, each local counter (LC) will be appended with a few small block-level counters. The encrypted data are stored in a data page in PRAM. The ECC bits are stored in an ECC page in the same group as the data page. In our scheme, ECC and data pages have the same organization and the counters in either type of pages are updated when there is a write access. The only difference is that the counters in the ECC page are only used to track the age and not for decryption.

There is an interesting interaction between encryption and ECC generation. Two options exist: the ECC bits can be computed either based on plaintext data or ciphertext data. If ECC is computed using plaintext data, the write access time can be reduced as the ECC computation and encryption can be performed in parallel. On the other hand, the read access latency is affected as the ECC check has to wait after the ciphertext data is decrypted. In comparison, computing ECC bits on ciphertext data reduces the read latency and increases the write latency. Since read operations are more performance critical than writes, we choose to compute ECC based on ciphertext data, as shown in Figure 7.

In an alternative memory organization, which uses PCM as main memory and DRAM as another level of cache [20], the counter cache can reside in the DRAM cache. This DRAM cache can also be used to store the uncompressed data when memory compression technologies [1][7] are employed. Since encryption makes data less compressible, encryption should happen after the compression stage. In this case, the plaintext data shown in Figure 7 is a compressed data block. The impact of memory compression is evaluated in Section 6.4.

The storage overhead for counters, which are used for both encryption and age estimates for ECC management, is small. If we assume the cache line size as 128 bytes in the last level cache, a 4KB page size, a 64-bit LPID, a 2-bit counter per encryption block, a 13-bit LC per line and a 32-bit GC, the overhead is around 3%. The selection of the counter sizes is to make sure that the age counters can record the number of writes that is beyond the endurance of PCM device ($10^8$ to $10^9$) [19].

## 6. Experimental results

### 6.1 Methodology

Our experiments are conducted using a cycle-accurate timing simulator developed upon the SimpleScalar toolset [2]. The underlying processor model is MIPS R10000 and the default configuration is listed in Table 1. The L2 cache size is set to 1MB to increase the memory traffic. The PRAM access latency is 1024 cycles [19]. The block cipher engine is a 128-bit AES cipher and the encryption latency is assumed to be 80 cycles [3]. Each cache line in the counter cache stores the counters for one page. It is composed of a 64-bit LPID [3], multiple local counters and one global counter (GC). The error correction code used is a binary cyclic code (BCH) [17]. The BCH latency depends on the number of bit errors that can be corrected and the maximum latency is 120 cycles for correcting 8 bit errors [11][26]. For each message data of $k$ bits, a BCH codeword (containing both data and the redundant ECC bits) with a length of $n$ bits can be constructed to correct up to $t$ bit errors out of the entire codeword. The length of the codeword $n$ should satisfy $2^{(m-1)}-1<n<=2^m-1$ and $m*t<=n-k$, where m is the minimum number of redundant ECC bits required for every bit error correction. In our experiments, 4 BCH codes are interleaved to protect the data at the granularity of the last-level cache line size (256 bytes). For

**Table 1. Default processor configuration**

| | |
|---|---|
| Branch Predictor | 64K-entry g-share, 4K-entry direct mapped Branch Target Buffer (BTB) |
| Superscalar Core | CPU frequency: 4GHz |
| | 7-stage pipeline: Fetch/Dispatch/Issue/RegisterRead/EXE /WriteBack/Retire, Pipeline bandwidth:4 |
| | Fully-symmetric Function Units: 4 |
| | Reorder Buffer (ROB) size: 128<br>Issue Queue (IQ) size: 64<br>Load Store Queue (LSQ) size: 64 |
| Execution Latencies | Address Generation: 1 cycle<br>Memory Access: 2 cycles (hit in data cache)<br>Integer ALU ops: 1 cycle<br>Complex ops:MIPS R10000 latencies |
| Instruction Cache (private) | 32KB 2-way, Block/line size 64B<br>10-cycle miss penalty |
| L1 Data Cache (private) | 32KB 2-way, Block/line Size 64B<br>10-cycle miss penalty<br>8 Miss Status Handling Registers (MSHRs) |
| L2 Unified Cache (shared) | 1MB 16-way Block/line size: 256B<br>1024-cycle miss penalty |
| Counter Cache (shared) | 32K 16-way Block Size: 64B<br>1024-cycle miss penalty |
| Cipher Engine | 128-bit AES with 80-cycle latency |
| ECC | BCH code, correct up to 8 bit errors, with up to 120-cycle decode-latency |
| PRAM | 4GB, 1024-cycle latency, endurance $10^8$ |

each BCH codeword, k = 512 bits (64 bytes) and m = 10. Therefore n-512 >= 10*t must be satisfied. It indicates that each additional bit error correction would need an additional 10 redundant ECC bits.

Memory-intensive benchmarks from SPEC2000 and SPEC2006 with high cache miss rates are used in the experiments. For lifetime analysis, an in-order processor model is used for simulation speed and the benchmarks are simulated to run for a hundred-billion instructions or upon completion.

## 6.2 Impact of encryption on wear-leveling techniques

In this section, we show the impact of encryption on two wear-leveling techniques: redundant bit-write removal and partial writes.

First, we analyze the write traffic of the SPEC benchmarks. We collect the total number of bit-write traffic to PRAM under three scenarios. The baseline is the one without either redundant bit-write removal or encryption. The other two are redundant bit-write removal with and without encryption, respectively. Assuming uniform bit writes across PRAM, Figure 8 shows the impact of encryption. Without encryption, there are lots of bit-write redundancies in the benchmarks. The highest one, *mcf*, has around 99.9% of its total bit writes redundant. Even for the lowest one, *equake*, the redundant bit writes are around 68%. In contrast, with encryption, every benchmark has only around 50% of total bit writes redundant. On average using the geometric mean (Gmean),

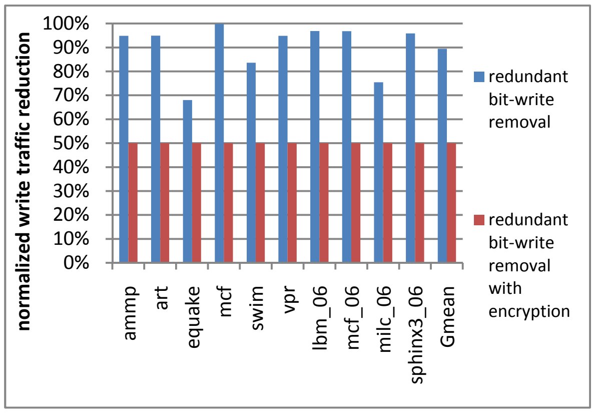

**Figure 8. Impact of encryption on redundant bit-write removal**

redundant bit-write removal can save around 90% of the bit-write traffic to PRAM. With encryption, however, it would only save half of bit-write traffic to PRAM.

As discussed in Section 3.1, encryption completely removes the benefit of partial-write. In this experiment, we first confirm the benefits of partial writes in reducing write traffic when encryption is not used. The traffic reductions normalized to the baseline, in which every replacement of a dirty cache line writes the whole cache line to PRAM, are shown in Figure 9. It can be seen that partial writes at word granularity (4 bytes) or encryption block granularity (16 bytes) can reduce around 45% or 35% of the write traffic on Gmean. Note here we conduct experiments in a conservative way as we do not simulate memory buffer organization. With coalescing effects under memory buffers, the results may be better [13]. With encryption, however, partial writes fail to reduce any write traffic.

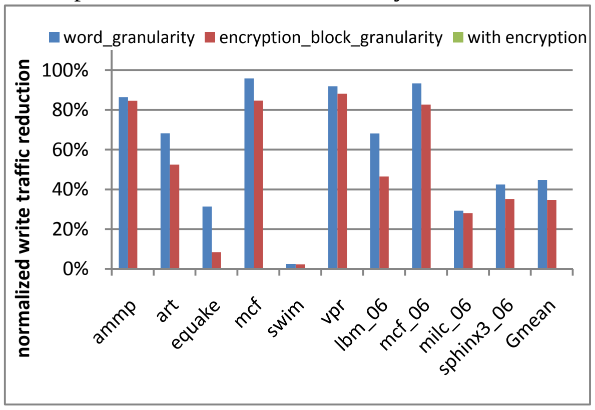

**Figure 9. Impact of encryption on partial writes**

## 6.3 Effect of the new proposed encryption counter scheme on partial writes

Figure 10 shows the effect of our proposed new encryption counter scheme (Section 3.2) to revive partial writes to reduce write-traffic to PRAM. The baseline is the one in which the whole dirty cache line is written back to PRAM upon replacement. In our experiment, we examined block-level encryption counters of different sizes, 1-bit, 2-

bit, 3-bit, and 4-bits. We also compared them to the ideal case when the block-level counters can be arbitrarily large (labeled '*MAX-bit counter*' in Figure 10).

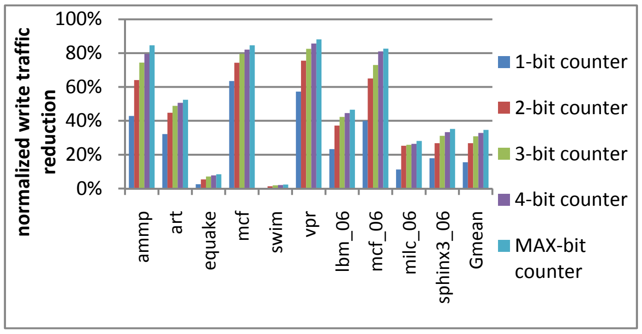

**Figure 10. Impact of our new encryption counter scheme on partial writes**

Several observations can be made from Figure 10. First, our new encryption counter scheme effectively revives partial writes to reduce write traffic. Second, as the size of block-level encryption counters increases, more write traffic can be reduced. The reason is that large counters reduce the number of overflows, which in turn spares more non-dirty blocks from being written back to PRAM. The increased counter size, however, incurs higher space overhead. Therefore, we choose 2-bit block-level counters, which achieve 26.8% write traffic reduction on average using Gmean.

## 6.4 Impact of memory compression on wear-leveling techniques

In this section, we examine the impact of memory compression. It can be used as another wear-leveling technique to reduce write traffic. However, it also varies the bit-level redundancy compared to uncompressed data. In our experiments, Frequent Pattern Compression (FPC) [1] and LZSS [21] are evaluated. FPC exploits the observation that certain value patterns such as zero-values are frequent in main memory while LZSS is a dictionary encoding scheme. Data are compressed at the cache line granularity and the compressed data are stored in-place into the original cache line location in main memory. Note this is an optimistic way of evaluating the compression effect on write traffic reduction. Storing the compressed data in compact may result in underflow/overflow when the size of a new compressed cache line is larger or smaller than the size of the old compressed cache line. The underflow/overflow may result in moving some data around, which increases write traffic.

Figure 11 shows the impact of FPC on redundant bit-write removal (LZSS shows similar performance). Such similarity is also reported in [7]. Note that it is not practical to combine compression with partial writes as compression may change the data layout in the compressed data. There are several observations. First, there are two

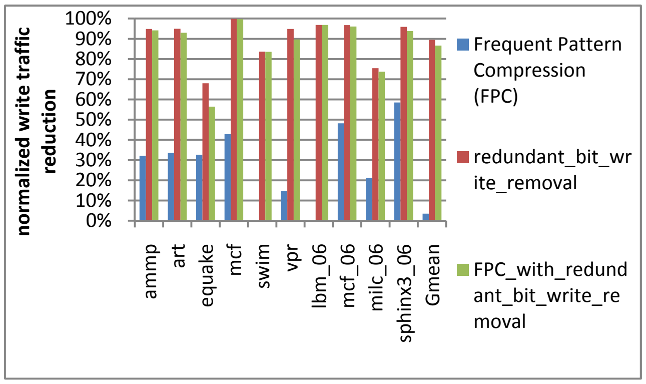

**Figure 11. Impact of Frequent Pattern Compression on redundant bit-write removal**

benchmarks which exhibit very low compressibility (*lbm* and *swim)* for FPC. Because of that, write traffic reduction from using FPC reaches 3.5% using Gmean and 28.4% on arithmetic mean. Second, compression reduces the bit-level redundancy. Redundant bit-write removal removes 82.3% traffic on Gmean of compressed write-back data, down from 89.5% on uncompressed write-back data. The significant bit-level redundancy under compression is due to abundant non-dirty data as shown in Figure 9 and high bit-level redundancy as shown in Figure 8. Overall, compression combined with bit-write removal achieves similar write-traffic reduction (87.7% on Gmean) to redundant bit-write removal only (89.5% on Gmean).

## 6.5 PRAM lifetime comparison

In this experiment, we map write traffic to PRAM lifetime estimate. We assume that write traffic to PRAM is uniformly distributed across the memory footprints of each benchmark. In other words, we assume that an optimal cache-line-level swapping/rotating/shifting scheme is already in place to extend the PRAM lifetime. With such idealistic assumption, we estimate the upper bound of the PRAM lifetime and the results are shown in Figure 12. The baseline is the one without partial writes, without bit-redundancy removal, encryption or compression.

From the figure, it shows that in the baseline model, the PRAM has relatively short lifetime for benchmarks *ammp* and *art*. This is because the benchmarks have high write-traffic density on their memory footprints. On average, the lifetime of the baseline is around 1.3 years using the arithmetic mean (Amean). With redundant bit-write removal, all benchmarks show large improvement on lifetime as a result of the high bit-write redundancy as shown in Figure 9. With this wear-leveling technique, the lifetime of PRAM reaches to around 51.3 years, more than 39x improvement over the baseline. When encryption is used, however, redundant bit-write removal can only achieve 2x improvement over the baseline, reaching the lifetime of around 2.6 years. For partial writes at word granularity, the average lifetime of PRAM across the

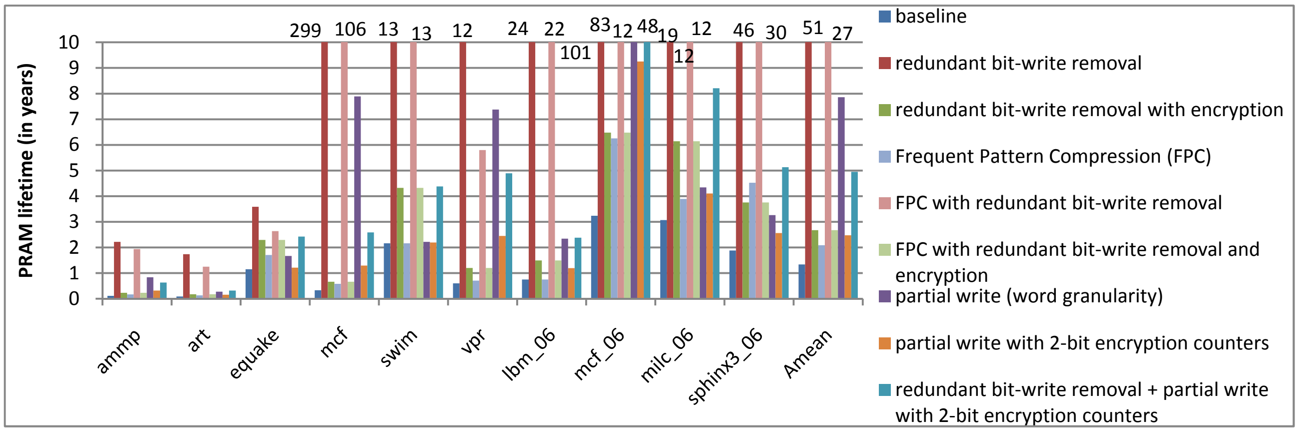

**Figure 12. PRAM lifetime with various schemes (assuming uniform writes)**

benchmarks is around 7.9 years. For partial writes with encryption, the original encryption counter scheme disables partial writes completely, thereby no lifetime improvement over the baseline. For partial writes with our proposed new encryption counter scheme (2-bit block-level encryption counters), the achieved PRAM lifetime is around 2.5 years on average, almost 2x improvement over the baseline. When we combine both wear-leveling techniques and enable encryption protection, we can achieve almost 5.0 years of lifetime, around 4x improvement over the baseline. Since the results in Figure 12 are already based on an idealistic assumption of uniform write traffic distribution, we believe that even with all the wear-leveling techniques, PRAM may fail in a limited time, which necessitates the use of ECC.

For compression, we can see FPC alone improves lifetime from 1.3 years to 2.1 years on average because of write traffic reduction. But it reduces the effect of redundant bit-write removal (lifetime is reduced from 51.3 years to 27.5 years). Such lifetime reduction despite the similar write traffic reduction in Figure 11 is due to the fact that a small difference in write traffic reduction may result in a large difference in lifetime. For mcf, 99.89% and 99.69% traffic reduction results in 299 years and 106 years of lifetime, respectively. With encryption, such bit-write redundancy is dropped to 50%, the same as without compression due to the diffusion effect of encryption algorithms.

## 6.6 ECC space overhead and wear out

Without dynamic ECC management, the fixed ECC space overhead for correcting up to 8 bit errors per 64 bytes is 10 bytes. In other words, 15.6% of the total PRAM capacity would have to be reserved for ECC. With dynamic ECC management, we leverage the fact that we only need to correct a much less number of bit errors when the PRAM is young (i.e., have not been written many times). To correct 1 bit error per 64 bytes, we need 10 ECC bits. Therefore, only 2.0% of the total PRAM capacity is necessary for ECC storage. If a group consists of 1024 pages, only 20 ECC pages is required to protect 1004 pages. For 2 bit errors, it becomes 3.9% and so on. This is much more efficient compared to the fixed ECC allocation scheme, which means the space otherwise reserved for ECC can be allocated for program use.

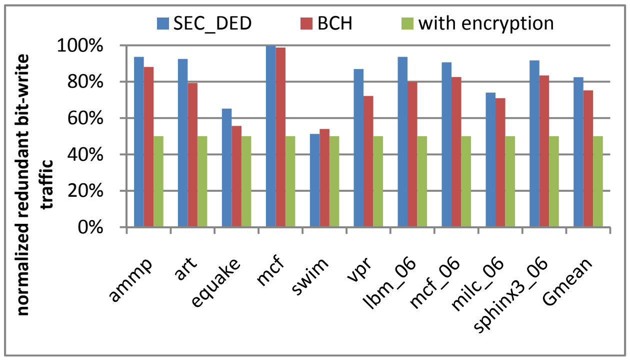

**Figure 13. Impact of encryption on redundant bit-write behavior of ECC**

In the next experiment, we examine redundant bit writes in ECC with and without encryption. Two ECC schemes, SEC-DED [24] and BCH [26], are used in this experiment and the results are shown in Figure 13. From the figure, it shows that without encryption, there are a high number of redundant bit writes, which can be eliminated with the redundant bit-write removal technique. However, with encryption, for both SEC-DED and BCH, the ratio of redundant bit writes becomes 50%, indicating the same behavior as the bit-write traffic in regular data.

## 6.7 Performance overhead of encryption and ECC

As discussed in Section 5, we choose to compute ECC based on encrypted data. To examine the overall performance impact, we model the proposed scheme in our timing simulator. For each benchmark, we skip the first one billion instructions and execute the next three-hundred million instructions. The performance results, which are normalized to the baseline without encryption or ECC, are shown in Figure 14.

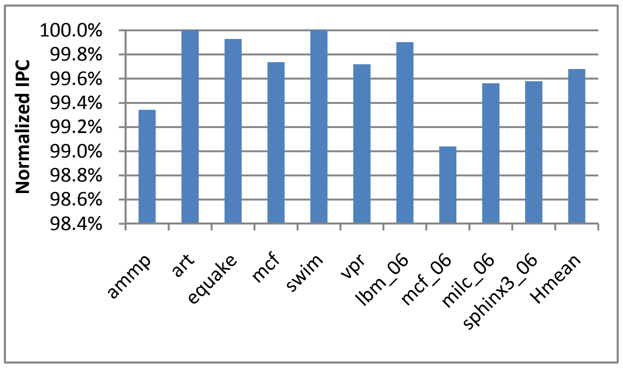

**Figure 14. Performance overhead of encryption and ECC**

From Figure 14, it shows that adding encryption and ECC has very small performance impact (0.3% on average). The benchmark, *mcf_06*, has the worst performance degradation (1%) due to its high memory traffic. The reason for such small performance overhead is that the PRAM access latency is large enough to dominate the overall performance.

### 6.8 Summary

In summary, without encryption, redundant bit-write removal and partial writes can achieve 51.3 years and 7.9 years of PRAM lifetime, respectively. With encryption, those two combined can only achieve 2.6 years of lifetime. Our proposed new encryption scheme can improve it to around 5.0 years at 1.6% space cost. Redundant bit-write removal combined with compression can achieve 27.5 years without encryption and 2.6 years with encryption. The encryption counters can be leveraged to monitor PRAM wear out status and the adaptive ECC management can be deployed with an increment of 2.0% space cost and dozen-cycle latency for each additional bit error to be corrected.

## 7. Conclusion

Phase change memory is a promising technology for computer systems. In this paper, we investigate the largely overlooked privacy issue of PRAM due to its non-volatility. We show that if encryption is used for privacy protection, some of previously proposed wear-leveling schemes will be severely affected. To mitigate the adverse impact of encryption, we propose to extend the counter-mode encryption to revive a wear-leveling technique: partial writes. We also investigate memory compression and show that it reduces memory traffic but hurts bit-level redundancy. Then we study error correction code (ECC) as an essential mechanism for PRAM lifetime extension. We propose a dynamic ECC management scheme to vary ECC protection strength according to the age of PRAM, which is conveniently provided from the encryption counters. Our experimental results show that the performance overhead for achieving privacy protection and lifetime improvement is minimal.

## 8. Acknowledgements

We thank the anonymous reviewers and Professor Christof Fetzer for their insightful comments. This work is supported by an NSF grant CNS-0905223 and an NSF CAREER award CCF-0968667.

## 9. References

[1] A.R. Alameldeen, Using Compression to Improve Chip Multiprocessor Performance, Ph.D. dissertation, CS Dept, University of Wisconsin-Madison, 2006.
[2] D. Burger et.al, The Simplescalar Tool Set Version 2.0. Technical Report, Computer Science Department, University of Wisconsin-Madison, 1997.
[3] S. Chhabra et.al, Making Secure Processors OS- and Performance-Friendly, ACM Transactions on Architecture and Code Optimization, 2009.
[4] S. Cho et.al, Flip-N-Write: A Simple Deterministic Technique to Improve PRAM Write Performance, Energy and Endurance, MICRO 2009.
[5] J. Daemen et.al, The design of Rijndael: AES - the advanced encryption standard. Springer-Verlag, 2002.
[6] C. Dirik et.al, The Performance of PC Solid-State Disks (SSDs) as a Function of Bandwidth, Concurrency, Device Architecture, and System Organization, ISCA 2009.
[7] M. Ekman et. al, A Robust Main Memory Compression Scheme, ISCA 2005.
[8] N. Ferguson, AES-CBC + Elephant diffuser: A disk encryption algorithm for Windows Vista. http://www.microsoft.com/, Aug. 2006
[9] R. Gleixner, Reliability Characterization of Phase Change Memory, 10th Annual Non-Volatile Memory Technology Symposium, 2009.
[10] J.A. Halderman et.al, Lest We Remember: Cold Boot Attacks on Encryption Keys, USENIX Security 2008.
[11] T. Kgil et.al, Improving NAND Flash Based Disk Caches, ISCA 2008.
[12] K. Kim et.al, Reliability investigations for manufacturable high density PRAM, 43rd Annual IEEE International Reliability Physics Symposium, 2005.
[13] B. C. Lee et.al, Architecting phase change memory as a scalable dram alternative. ISCA 2009.
[14] D. Lie et.al, Architectural Support for Copy and Tamper Resistant Software, ASPLOS 2000.
[15] A.J. Menezes et.al, Handbook of Applied Cryptography, CRC Press, 1996.
[16] N. Mielke et.al, Bit Error Rate in NAND Flash Memories, International Reliability Physics Symposium, 2008.
[17] R. H. Morelos-Zaragoza, The Art of Error Correcting Coding, Second Edition, John Wiley & Sons, 2006.
[18] A. Pirovano et.al, Reliability study of phase-change nonvolatile memories, IEEE Transactions on Device and Materials Reliability, 2004.
[19] M. K. Qureshi et.al, Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. MICRO 2009.
[20] M. K. Qureshi et.al, Scalable high performance main memory system using phase change memory technology. ISCA 2009.
[21] J. A. Storer et.al, Data Compression Via Textual Substitution, Journal of the ACM, 1982.
[22] C. Yan et.al, Improving Cost, Performance, and Security of Memory Encryption and Authentication, ISCA 2006.
[23] D.H. Yoon et. al, Virtualized and Flexible ECC for Main Memory, ASPLOS 2010.
[24] D.H. Yoon et.al, Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches, ISCA 2009.
[25] W. Zhang et.al, Characterizing and Mitigating the Impact of Process Variations on Phase Change based Memory Systems, MICRO 2009.
[26] W. Zhang et.al, Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures, PACT 2009
[27] P. Zhou, et.al, A durable and energy efficient main memory using phase change memory technology. ISCA 2009.