

Defining Anomalous Behavior for Phase Change Memory

Siddhartha Chhabra and Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
{schhabr, solihin}@ncsu.edu

Abstract

Traditional memory systems based on memory technologies such as DRAM are fast approaching their cost and power limits. Alternative memory technologies such as Phase Change Memory (PCM) are being widely researched as a scalable, cost- and power-efficient alternative for DRAM. However, a PCM memory cell has a limited endurance of $10^7 - 10^8$ writes, resulting in an ideal lifetime of only a few years which can be reduced further due to the non-uniform distribution of write traffic to the main memory. Wear leveling algorithms have been proposed to ensure uniform distribution of traffic to the main memory. However, this limited write endurance poses an easy to exploit loophole for attackers to bring the system down. A malicious application can perform multiple writes to the same memory location (PCM cell) and quickly cross the endurance limit for the cell. We show that wear leveling algorithms, while meeting their objective for non-malicious applications, do not help to protect the system from this exploit. If PCM based systems are to be adopted for future systems, it is critically important that this security hole is addressed. In this paper, we define what constitutes as anomalous behavior for PCM and, in general, for memories with limited write endurance. Once an application is established to exhibit anomalous behavior, solutions can be built on top of it to prevent any reduction in the lifetime of the system due to the main memory reaching its write endurance limit.

1 Introduction

In recent years, the number of cores used for a system have been increasing. Most commonly, there are multiple cores integrated on a single chip and there can be multiple such chips present in a system. The increased number of cores allows multiple applications to run concurrently increasing the overall throughput of the system. In addition, the working sets of applications have been increasing steadily. The combined effect of these factors results in an increasing need to have a larger main memory system, to effectively hide the long disk latency. However, traditional memory technologies like DRAM face serious problems in terms of cost, energy consumption, and scalability [1, 4]. Hence, it has become imperative to research on new memory technologies which can offer an increased capacity to meet the increasing memory requirements and at the same time are cost and energy efficient. Phase-Change Memory (PCM) presents one such promising alternative. PCM is $4\times$ denser than DRAM, providing an increased capacity while staying within the cost and energy budget. Additionally, PCM prototypes have been demonstrated to scale much better than DRAM [1, 9].

While PCM scales much better compared to DRAM, it suffers from several limitations. Firstly, PCM has higher latencies compared to DRAM, making the memory accesses slower and affecting

the overall system performance. Secondly, a PCM cell has a limited write endurance of $10^7 - 10^8$ writes, after which the cell becomes unusable, giving PCM an ideal lifetime of 3.2 years [7]. Recent research has focussed on bridging the gap between DRAM and PCM access latencies [3, 7, 8]. For example, Qureshi et al. [7] propose a hybrid main memory system where PCM is combined with a DRAM buffer, acting as a cache. The DRAM buffer provides reduced latency and the PCM provides the increased capacity. There has also been recent work on increasing the lifetime of a PCM based system by reducing the write-traffic to the main memory. For example, Qureshi et al. [7] employ a lazy-write architecture, wherein the PCM is written to only conditionally. Partial writes [3, 7] have also been exploited to reduce the write traffic to PCM. However, the limited write endurance of PCM poses a critical security threat which has largely been ignored by the research community. An attacker can write an attack application which repeatedly writes to the same memory location resulting in specific PCM cells becoming unusable, thereby, rendering the system unusable. Figure 1(a) presents one simple attack application which works by thrashing the cache, resulting in a writeback in every loop iteration.

```
//Assuming a system with 8-way, 1MB L2 cache
int *a;
int i = 0;
a = (int*)malloc(2359296*sizeof(int)); //Allocate 9 arrays of size 1MB
while(1)
{
    for (i = 0 ; i < 9 ; i++)
    {
        a[*262144]++; //Write to first element of the arrays
                        //to thrash the cache
    }
}
```

(a) Attack on PCM only system

```
int *a;
//Get DRAM size from the proc file system (/proc/meminfo)
long long j = 0;
a = (int*)malloc(262144*256*4); //Assuming a 256MB DRAM
while(1)
{
    for(j = 0 ; j < 262144*64*4; j+=1024)
    {
        a[j]=1; //Write to the first block on every page
    }
}
```

(b) Attack on Hybrid PCM system

Figure 1. Attacks on PCM based Memory Systems.

The application in Figure 1(a) assumes a main memory system consisting of PCM alone. Hence, all writebacks will directly go to the PCM. Qureshi et al. [6] showed that such an attack can significantly reduce the lifetime of the system, with the *Time to Failure*

(*TTF*) for the system calculated as:

$$TTF = \frac{Cell\ Endurance \cdot Cycles\ per\ Write}{Cycles\ in\ one\ Second}$$

Assuming a 4GHz system, with a write latency of 2^{12} cycles to the PCM, the above attack succeeds in 32 seconds. One might assume that a hybrid memory system with a DRAM buffer will prevent such attacks. However, Figure 1(b) presents an attack that can be used to attack a hybrid main memory system. The attacks works by thrashing the DRAM cache and causing repeated writebacks to the PCM. In the most conservative case, the attacker needs to write to all the pages in the DRAM and one additional page to force a writeback to the PCM in every iteration. Hence the worst case *TTF* for this attack on a hybrid memory system can be calculated as:

$$TTF = Time\ to\ fill\ DRAM\ Cache + \frac{Cell\ Endurance \times Cycles\ per\ Write}{Cycles\ in\ one\ Second}$$

where:

$$Cycles\ per\ Write = \frac{Number\ of\ Pages\ in\ DRAM\ Cache \times Cycles\ to\ Write\ to\ PCM}{Cycles\ to\ Write\ to\ PCM}$$

Assuming a 256MB DRAM buffer, the above attack succeeds in 24 days, which is less than 2.5% of the ideal lifetime of PCM. Even for regular applications, the writes to main memory are not uniform. Even though, the traffic is much more uniform compared to the attack application, the non-uniformity present can easily result in endurance failure for one cell earlier than the others. To this end, researchers have proposed using wear leveling algorithms [6, 7, 11] that attempt to distribute the writes equally to all the cells in memory over the lifetime of the system. The proposed wear leveling algorithms are shown to work well for non-malicious applications, however, for the attack applications, we presented above, the proposed wear leveling algorithms do not help. We discuss two of the most recently proposed wear leveling algorithms, *Fine Grained Wear-Leveling (FGWL)* [7] and *Start-Gap Wear-Leveling* [6] and show that the attacks discussed above can still proceed without being detected.

The attacks discussed above are a severe form of Denial of Service (DoS) attacks where the system is not only unavailable for servicing memory requests but is incapable of doing so. As indicated by the cost, power, and scalability needs for main memory, a PCM based system is likely to be deployed in future systems. However, if PCM based systems are to become a reality, it is critically important, that a solution for detecting and preventing against such attacks is incorporated in PCM based systems.

Contributions: In order to make sure that such an attack does not succeed, we need to make sure that it is detected that the system is under attack. To come up with an attack detection mechanism, we first need to define what constitutes as an *anomalous behavior* in terms of write traffic to the main memory. As can be seen from the example applications above, for an attack application to succeed in incapacitating the system, it needs to perform a large number of writes to cross the endurance limit for a PCM cell. Hence, a direct metric to detect anomalous behavior could be the writes performed per instruction. For a potentially malicious application, this factor

should be very high compared to regular applications. As the attack application needs to write to the same address over and over again to break the system, another metric could be the distribution of writes performed to the memory pages. A high concentration of writes to a particular block in a page will indicate a potentially malicious behavior. However, we show that these simple metrics can be bypassed by an attacker. For defining anomalous behavior, the metric should incorporate the write traffic *and* the addresses to which this write traffic is directed *together*. We propose, *Write Traffic Per Page (WTPP)*, and show that it is a complete metric that can be used to detect anomalous write behavior which can potentially harm a PCM based system. As a second component, we need to ensure that the system can reliably collect these statistics and the attacker cannot, in anyway, subvert this process. Once we can identify anomalous behavior reliably, we can build solutions on top of it, to prevent such an attack from succeeding.

Overall, we make the following contributions:

- We define what constitutes as anomalous behavior in terms of write traffic for a PCM based system, and, in general for any memory technology with limited write endurance.
- We propose a hardware based solution to reliably collect statistics required for establishing anomalous behavior.
- We show that a complete replacement of PCM, while beneficial for scalability, is not possible when we consider the dependability and security of a system. We need a hybrid main memory system to reliably benefit from the increased memory capacity provided by PCM.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes our experimental setup. Section 4 describes our anomaly detection mechanism. Section 5 presents a hardware implementation for our anomaly detection mechanism. Section 6 presents a brief discussion on why PCM cannot replace DRAM completely and we conclude in section 7.

2 Related Work

Recent months have seen a surge in research on alternate memory technologies due to the fundamental limitations of DRAM in terms of cost, energy, and scalability. The work thus far on PCM can be categorized into three primary areas:

Bridging Latency gap: PCM has significantly higher read and write latency, when compared to DRAM. This higher latency of PCM has a direct impact on overall performance of the system. Qureshi et al. [7] propose a hierarchical main memory system with a smaller DRAM (latency benefits) acting as a cache for a large PCM main memory (capacity benefits). Lee et al. [3] propose buffer reorganizations to bridge the latency gap. In a recent work Qureshi et al. [8] proposed write cancellation and pausing to further improve PCM's read performance. Cho et al [2] propose Flip-N-Write, an architectural technique to reduce the write latency of PCM.

Bridging the Energy gap: PCM array read and write energies are $2.1\times$ and $43.1\times$ greater than those for DRAM [3]. [2, 3, 12] propose architectural mechanisms like buffer reorganization and reducing write traffic which in turn results in reduced write energy.

Increasing the write endurance: Qureshi et al. [7] propose lazy write organization to write to PCM from the DRAM cache only conditionally. If a page is present in the PCM and has not been modified

while in the DRAM, it is not written back. This avoids many additional writes that can occur in the base PCM system. [3, 7] propose line level writebacks or partial writes to main memory to minimize the number of bits written to. [2] propose Flip-N-Write which once again reduces the number of bits written to by examining the new data with the original data word. This technique increases the write endurance of PCM by a factor of almost 2. In addition, the wear leveling algorithms proposed for Flash memory can be used to ensure uniform distribution of write traffic to the memory. [3, 7, 11] propose various wear leveling algorithms for a PCM main memory system. [6] proposes *start-gap* wear leveling algorithm and tries to provide security through the algorithm. However, we show in a later section that wear leveling algorithms, including start-gap, will not help in preventing against this critical reliability hole.

3 Experimental Setup

We use Simics [5], a full system, cycle-accurate simulator, and build a virtual memory simulator to obtain the results shown in this paper. We model a 4GHz, in-order processor with split L1 data and instruction caches. Both caches are 32KB in size, 2-way set associative and have a 2-cycle round-trip hit latency. The L2 cache is unified and has a size of 1MB, 8-way set-associative, and 10-cycle round-trip hit latency. All caches have 64B blocks and use LRU replacement. We use SPEC 2006 benchmarks for our experiments [10]. We skip 5 billion instructions for each benchmark, and then simulate for 200 million instructions.

4 Anomaly Detection Mechanism

In this section, we first show why wear leveling algorithms cannot work to prevent against such attacks. We then present a discussion towards the definition of anomalous behavior for limited write endurance PCM based systems.

4.1 Wear Leveling Algorithms: Can they defend against malicious behavior?

Wear-leveling algorithms have been defined and widely used for memory systems with limited write endurance, for example, flash memory. Wear-leveling algorithms attempt to make the write traffic distribution more uniform to ensure that the cells wearout as close to each other as possible. We assume that a PCM based system will also employ some sort of wear-leveling to ensure uniform distribution of writes and prevent one cell from failing earlier than the others. It has been shown in earlier works [3, 7] that wear-leveling algorithms work well for (*non-malicious*) applications that the proposed system was evaluated for. However, we take two of the most recently proposed wear-leveling algorithms for PCM and show that the DoS attacks can still proceed to bring the system down.

Fine Grained Wear-Leveling (FGWL) [7] is one of the earliest wear-leveling algorithms proposed for PCM. In FGWL, the lines in each page are stored in a rotated manner. For example, a 4KB page with 64-byte line size will have 64 lines. These lines are stored in a rotated manner with a shift amount of 0-63. If the shift amount is 0, the lines are stored un-rotated. For a shift amount of 1, Line 0 is stored where Line 1 would be stored on a traditional system and so on, with Line 63 stored in Line 0's place. The rotate amount stays the same as long as the page is resident in memory. On a page fault, a Pseudo-Random Number Generator (PRNG) is consulted to give a shift amount for the page and this stays associated with the page until it is swapped out to the swap space. Now, if we consider

an attack application, as described in Figure 1(a), it has a working set of 9 pages in all. An Operating System (OS) carries out a page swap when a currently running application experiences a page fault and there is not enough space to bring in the page to main memory without a replacement. If we have a lightly loaded system, for example, a server receiving a low activity phase, the pages belonging to an attack application might never be replaced and stay resident in memory. The scenario becomes even more realistic considering that PCM memories are likely to be much bigger than traditional DRAM memories, giving the pages of an attack application, a much higher likelihood to stay resident in memory for a much longer time. FGWL works only when a physical page is replaced and a new page is brought in to the main memory. Hence, the security afforded by FGWL is contingent on the system experiencing regular page faults and in the scenario we just discussed, it will not be able to detect the attack and the system will be brought down in 32 seconds.

Secondly, the most recent wear-leveling algorithm proposed in [6], the start-gap wear-leveling algorithm, discusses the attack scenario briefly and suggests that the wear-leveling algorithm can actually defend against such an attack. We first introduce the algorithm and then show that it can be easily surpassed by an attacker. For Start-gap algorithm, the entire memory is viewed as a set of lines. For our discussion, we assume memory consisting of four lines (lines 0-3). One line of the main memory is empty (*a gapline*) and is pointed to by the *gap* register (line 4). A *start* register is used to keep track of the first line in the memory, and initially points to line 0. The lines in memory can be visualized as forming a circular buffer. After ψ writes to the main memory, the gap register moves up one place, to line 3, and, line 3 is copied to the original location of the gap register (line 4). When the gap register reaches line 0, all lines in memory have been shifted down by one place, hence, start register is incremented to point to line 1. It is easy to see that a line can be located in the main memory using the start and gap register. [6] also proposes a variant of the start-gap algorithm, the *Region based Start-Gap (RBSG)* algorithm to prevent the attack from being successful. The key insight is to have multiple regions in memory, with each region having its own start and gap register. The region size is chosen such that it is ensured that a line will shift before reaching its endurance limit. The security afforded by this algorithm can be trivially surpassed by an attacker armed with any knowledge of the wear-leveling algorithm. Figure 2 explains how this attack proceeds.

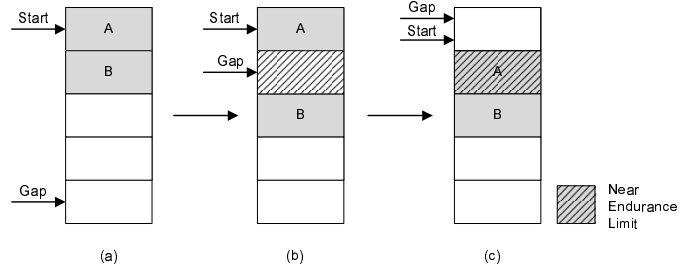


Figure 2. Start-gap Attack.

Figure 2(a) shows the initial memory state. The attacker has allocated lines A and B. The attacker writes to line B repeatedly and RBSG algorithm will make sure that the line B moves before reaching its endurance limit(Figure 2(b)). Note that, the block where line B originally was, is already very close to its endurance limit

as RBSG is designed to move the line just before it reaches its endurance limit. Another write to the line will bring the system down. Now, the attacker starts writing to line A, after ψ writes the gap will move and line A will now be in the position where line B originally was (Figure 2(c)). Another write to line A would make the PCM cells cross their endurance limit for this line, hence, defeating the protection provided by RBSG. We can clearly see that RBSG does not do anything to prevent the attack from happening. The security provided is contingent on the fact that the attacker does not know about the wear-leveling algorithm, which is a very weak assumption to make.

The fundamental reason why wear-leveling algorithms cannot defend against these attacks is the fact that: *Wear-leveling algorithms are designed to ensure uniform distribution of writes to the main memory assuming non-malicious applications. They are not designed to handle a system under attack.* Hence, it is necessary to come up with a mechanism that can define what constitutes as an anomalous behavior and build a solution on top of it to prevent such attacks.

4.2 Defining Anomalous Behavior

In this section, we go through metrics that can be used to define anomalous behavior that can harm a PCM based system. We show that some of the obvious metrics, while looking to serve the purpose, if used for defining anomalous behavior, will result in false negatives by giving the attacker an opportunity to bypass the anomaly detection. We then propose, *Writeback Traffic Per Page (WTPP)* and show that it is a complete metric and can be used to define anomalous behavior for PCM based systems.

4.2.1. Writebacks Per Instruction (WPI). It is obvious that the attacker needs to force multiple writebacks to the main memory in order to break a PCM based system. The number of writebacks performed by an attack application should be significantly more than regular applications. Hence, one direct metric to detect anomalous behavior could be *Writebacks Per Instruction (WPI)*. Figure 3 shows the WPI for all SPEC 2006 applications as well as the attack application discussed previously (Figure 1(a)).

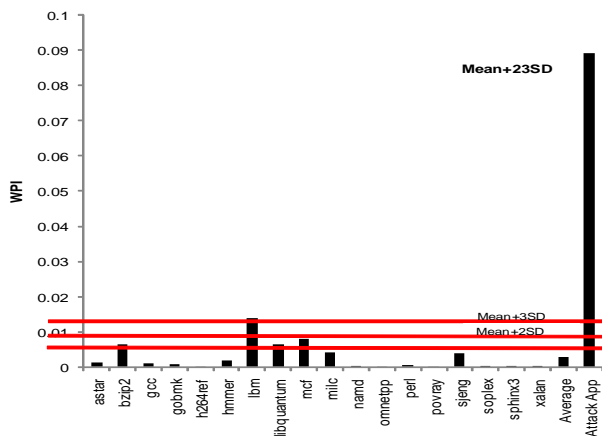


Figure 3. Writebacks Per Instruction.

As we can clearly see from the figure, the WPI for the attack application is much higher (23 standard deviation higher than the mean of the WPIs) than any of the SPEC 2006 applications. If we assume a normal distribution of writeback traffic, then, more than

99.99% of the regular applications will fall within three standard deviation of the mean. For the workloads we consider, all the applications are within three standard deviation of the mean of the WPIs. Hence, we can set the *system WPI* as the mean of the WPIs of regular benchmarked applications plus three standard deviation. The anomalous behavior can then be defined as: *A WPI of more than the system WPI indicates potentially anomalous behavior.* When an application is detected to exhibit this behavior, it can be marked as potentially harmful for the lifetime of the system and a protection mechanism can work to protect the system.

This seemingly simple metric seems to be enough to detect potentially malicious activity. However, an attacker armed with the knowledge of the detection mechanism, can break this definition of anomaly and prevent triggering anomalous behavior. For example, the attacker can trivially change the attack application discussed in Figure 1(a) as shown in Figure 4 below:

```
while(1)
{
    for (i = 0 ; i < 9 ; i++)
    {
        a[*262144]++; //Write to first element of the arrays
                //to thrash the cache
        __asm {
            NOP //Series of 10 NOPs
            .
            .
        }
    }
}
```

Figure 4. Bypassing WPI as a metric.

The above attack will still be writing to the memory in every iteration. In effect, the cycles to write to memory will increase from 2^{12} (time to write to PCM) to $2^{12} + 10$ cycles (one cycle for each NOP instruction). Plugging in this number in the equation for TTF, we get the TTF as 32.78 seconds, which is marginally higher than the base attack application itself. However, the insertion of NOPs reduces the WPI by almost a factor of 10, getting it well within the system WPI. Hence, a seemingly useful metric, WPI, cannot be used to define anomalous behavior as attackers can trivially bring the WPI down to result in false negatives.

4.2.2. Write Traffic Distribution (WTD). Now, for the attack application to succeed in a short time, it has to repeatedly force writebacks to the same address, which in turn implies, that it needs to force stores to the same location over and over again. A plausible metric to define anomalous behavior could then be the Write Traffic Distribution (WTD). A high concentration of writes to a few lines on the page could potentially indicate malicious activity. Figure 6(b) shows the distribution of writes to the top ten most frequently written to pages.¹

As can be seen, the distribution of writes for regular applications is much more uniform than that for the attack application. Hence, we can formulate a heuristic based on this observation to define anomalous behavior. One possible heuristic could be: *If the distribution favors one set of lines by more than, say $\alpha\%$, then it could indicate possible malicious behavior.* The value of α could be fixed based on characterization of the workloads that the system is expected to run.

However, WTD, similar to WPI, can be broken by an attacker with some knowledge of this definition of anomalous behavior. As

¹For brevity reasons, we just show the write traffic distribution averaged across all SPEC 2006 benchmarks.

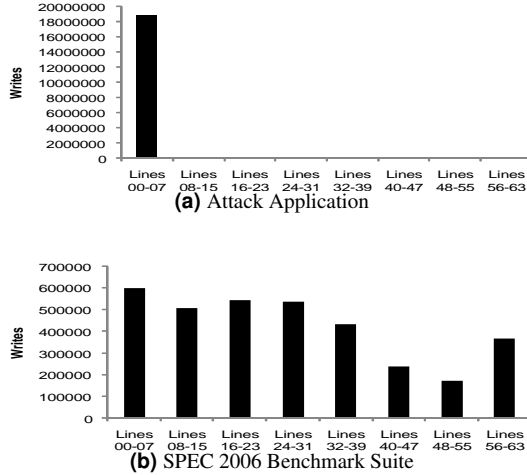


Figure 5. Write Traffic distribution to the Most Frequently Written to Page.

an example, the attacker can write an application which distributes writes equally to all lines of the page. Assuming a 4KB page size, and 64 bytes line size, this attack will take $64 \times$ more time than the base attack, resulting in a TTF of 34 minutes (64×32 seconds). Hence, once again, a seemingly useful metric, WTD, cannot be used to define anomalous behavior.

4.2.3. Writeback Traffic Per Page (WTPP). In order to define anomalous behavior and to come up with a metric to represent the same, we need to consider two factors. One, the attack application will make a large number of writebacks, and, two, the writebacks will be concentrated on a fixed set of addresses. However, if we consider the number of writebacks alone (WPI) or the set of addresses to which the writes are made alone (WTD), then the attacker can surpass the anomaly detection mechanism. Hence, to come up with a foolproof mechanism, that does not result in any false negatives and cannot be surpassed by the attackers, we need to consider these two factors together.

Lets assume a 16GB PCM main memory system with an ideal lifetime of 3 years [7]. In order to get this ideal lifetime, the writeback traffic should be uniform. Assuming, uniform distribution of writeback traffic, we can get the following equation for the lifetime of a PCM system:

$$Lifetime = \frac{Size\ of\ PCM}{Writeback\ Traffic(GBPS)} \times Write\ Endurance$$

It will take $\frac{Size\ of\ PCM}{Writeback\ Traffic(GBPS)}$ seconds to write to the entire memory once. Hence, the lifetime can simply be calculated by multiplying with the cell endurance as shown above. Now since the endurance of a PCM cell is 2^{32} and there are 2^{32} seconds in one year, the above equation simply reduces giving the lifetime as $\frac{Size\ of\ PCM}{Writeback\ Traffic(GBPS)}$ years. Plugging in the numbers for ideal lifetime and assumed size of PCM, we obtain a write traffic of 5.3 GBPS to keep the system at its ideal lifetime. There are 4194304 pages in a 16GB memory assuming a page size of 4KB, which gives us a writeback traffic of 1.2 KBPS per page, to keep ideal lifetime.

A writeback traffic of 1.2 KBPS to a page ensures that the cells on this page will not reach the endurance limit before the ideal lifetime of the system, even if we assume that it keeps receiving this traffic throughout. Hence, we can define anomalous behavior as: *A page receiving a WTPP more than 1.2 KBPS.*

If WTPP is taken as a metric, the attacker cannot conduct a successful attack without lowering the WTPP of the malicious application's pages to less than 1.2 KBPS. One might argue that a malicious application will not be flagged as exhibiting anomalous behavior if it lowers its WTPP to less than 1.2 KBPS. This is true, but in this case, even though the application's intent is to destroy the system, it will not succeed in doing so before the ideal lifetime of the system. Hence, the malicious application's behavior in this case is *not* anomalous.

One might argue that the metric is too conservative and can result in flagging anomalous behavior for regular applications as well. The mechanism works at a finer granularity of pages and irrespective of whether the page belongs to a regular or attack application, a WTPP greater than 1.2 KBPS will shorten the lifetime of the system and must be flagged as anomalous behavior, so prevention mechanisms can come into action to prevent this from happening. It is important to note that anomalous behavior does not necessarily mean that the application is malicious, instead, it simply indicates behavior that can potentially reduce the lifetime of the system. Hence, the conservative definition of anomalous behavior is *required* if the system is to retain its ideal lifetime. We collected statistics for the most modified pages for SPEC 2006 applications and the attack application. Figure 6 presents these results.

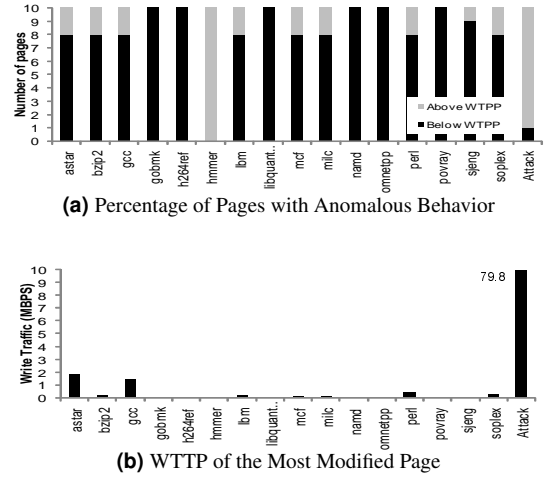


Figure 6. WTPP Analysis.

Figure 6(a) shows the percentage of ten top most modified pages that are below a WTPP of 1.2 KBPS and those that are above a WTPP of 1.2 KBPS. As can be seen from the figure, for most of the regular applications (except *hammer*), a maximum of two pages exhibit anomalous behavior by having a WTPP of more than 1.2 KBPS and as expected, all data pages of the attack application exhibit anomalous behavior. Figure 6(b) presents the WTPP for the most frequently written to page. The maximum WTPP for regular applications is less than 2 MBPS whereas the attack application has a WTPP of nearly 80 MBPS. It is also interesting to note that *hammer*, while having all of its top ten modified pages exhibit a WTPP

of more than 1.2 KBPS, has a maximum WTPP of only 11 KBPS. However, our anomaly detection mechanism rightly flags them as exhibiting anomalous behavior as even a WTPP of 11 KBPS can reduce the lifetime of the system by $10\times$.

5 Hardware Implementation of Anomaly detector

We have established WTPP as a complete metric that cannot be manipulated by an attacker without resulting in a non-anomalous behavior (a WTPP of less than 1.2 KBPS). However, it is critically important that the system can collect the WTPP statistics reliably. We propose a hardware implementation for the anomaly detector wherein an on-chip cache, called the *page counter cache*, is added to keep track of WTPP for all pages accessed by the application. It is important for the system to keep track of all the pages that are written to by an application, as we are tracking anomalous behavior at the granularity of pages. More importantly, if a fixed number of pages is tracked, the attacker can prevent reliable collection of statistics, by writing to one page more than the number of pages that are being monitored. Additionally, to keep the runtime overheads of the anomaly detection mechanism reasonable, we track anomalous behavior across context switches.

The page counter cache behaves like a regular cache, however, any overflows are stored in a dedicated portion of the main memory, called the *scratch memory*. Figure 7 explains the functioning of the page counter cache.

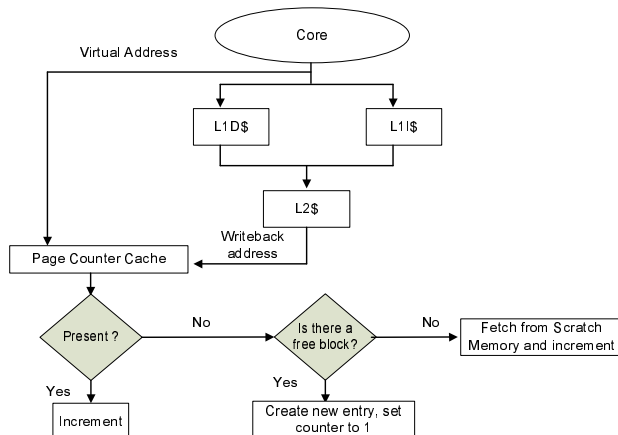


Figure 7. Anomaly Detection Implementation.

On a writeback, the page counter cache is accessed and the counter associated with the page receiving the writeback is incremented. If the counter is not present in the cache and there are no free entries available, it indicates that the cache has overflowed to the scratch memory and the counter must be fetched from the scratch memory and a replacement must be made. The page counter cache is scanned along with the scratch memory (if needed) across context switches, to calculate WTPP of all the pages written to in this time quantum and flag any anomalous behavior.

6 PCM: A complete replacement for DRAM?

As described in section 1, a PCM only system can be made unusable within 32 seconds by an attack application. We defined anomalous behavior and proposed an anomalous behavior detection mechanism. However, even after an anomalous behavior is

detected, a solution must be in place to protect the system against such attacks. Killing the application is not a practical option as it will result in a large number of regular applications being terminated as well. Hence, it is necessary to keep the application executing. In order to do so, the system must have some form of memory like DRAM, where these pages with anomalous behavior can be re-mapped to. In addition, current PCM prototypes have higher latencies than DRAM, which can result in a performance degradation, especially when the higher latency of PCM dominates the benefits from its increased capacity. Hence, having a DRAM portion is required from both reliability and performance perspective resulting in a hybrid main memory system, similar to that proposed by Qureshi et al. [7], as a practical replacement for a DRAM-only main memory system.

7 Conclusions

In this paper, we defined what constitutes as an anomalous behavior in terms of write traffic for a PCM based system. We showed that simple metrics like Writebacks per Instruction (WPI) and Write Traffic Distribution (WTD) can be bypassed by the attackers. We defined Writeback Traffic per Page (WTPP) as a complete metric that can be used to detect anomalous behavior. We then proposed a hardware based solution to reliably collect the WTPP statistics for the anomaly detection mechanism. Based on our observations, we concluded that a complete replacement of DRAM with PCM is not a feasible alternative from both performance and reliability perspective.

References

- [1] *International Technology Roadmap for Semiconductors, ITRS 2007.*
- [2] S. Cho. et al. Flip-N-Write: A simple deterministic technique to improve pram write performance, energy and endurance. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.
- [3] B. C. Lee. et al. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th International Symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM.
- [4] C. Lefurgy. et al. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.
- [5] P. S. Magnusson. et al. Simics: A Full System Simulation Platform. *IEEE Computer Society*, 35(2):50–58, 2002.
- [6] M. K. Qureshi. et al. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *Proc. of the 42nd Annual International Symposium on Microarchitecture*, 2009.
- [7] M. K. Qureshi. et al. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th International Symposium on Computer architecture*, pages 24–33. ACM, 2009.
- [8] M. K. Qureshi. et al. Improving read performance of phase change memories via write cancellation and write pausing. In *Proc of the 16th International Symposium on High Performance Computer Architecture (HPCA-16)*, 2010.
- [9] S. Raoux. et al. Phase-change random access memory: a scalable technology. *IBM J. Res. Dev.*, 52(4):465–479, 2008.
- [10] Standard Performance Evaluation Corporation. <http://www.spec.org>, 2004.
- [11] W. Zhang. et al. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proc of the 16th International Conference on Parallel Architectures and Compilation Techniques (PACT-18)*, 2009.
- [12] P. Zhou. et al. A durable and energy efficient main memory using phase change memory technology. In *Proceedings of the 36th annual International Symposium on Computer architecture*, pages 14–23. ACM, 2009.