Kingsguard: Write-Rationing Garbage Collection for Hybrid Memories

Shoaib Akram
Ghent University
Belgium

Jennifer B. Sartor
Vrije Universiteit Brussel and Ghent University
Belgium

Kathryn S. McKinley
Google
United States of America

Lieven Eeckhout
Ghent University
Belgium

Abstract

Phase Change Memory (PCM) offers higher capacity and energy efficiency than DRAM. It has two disadvantages: (1) write endurance is low, and (2) latency is high. Hybrid memory combines DRAM and PCM to promise low latency, higher capacity, energy efficiency and durability. Prior hardware and OS approaches spread writes out using *wear leveling*, and place frequently-written pages in DRAM. Unfortunately, prior *coarse-grained* approaches lead to impractical PCM lifetimes of 4 years or less for popular Java applications.

This work exploits garbage collection in managed language runtimes to make PCM a practical DRAM replacement leaving the programming model unchanged. We find that for 16 Java applications on average (1) 70% of writes occur to newly-allocated nursery objects, and (2) 2% of objects capture 81% of writes to mature objects. We introduce two *write-rationing garbage collectors* that improve PCM lifetime: (1) Kingsguard-nursery places nursery objects in DRAM and survivors in PCM, reducing PCM writes by 5× compared to PCM-only systems with wear-leveling. (2) Kingsguard-writers (KG-W) places nursery objects in DRAM and nursery survivors in a DRAM monitoring space. It then monitors writes to all mature objects and moves unwritten mature objects from DRAM to PCM. Because most mature objects are unwritten, KG-W exploits PCM capacity while increasing PCM lifetimes by 11×. This work demonstrates garbage collection as a promising avenue to manage hybrid memories.

1 Introduction and Motivation

Modern applications demand high main memory capacity. Unfortunately, DRAM scaling has slowed leaving researchers to look for alternative memory technologies. Phase Change Memory (PCM) offers high density, scalability (capacity), low standby power, and non-volatility, but has four shortcomings: high access latency, write latencies exceed read latencies, high write energy, and low endurance. Write endurance is challenging because each write changes the material form.

Researchers have combined DRAM and PCM to form hybrid memories, seeking the best of both memory technologies. Hardware wear-leveling then spreads writes out across the entire PCM capacity to improve PCM lifetime. Prior work also proposes OS approaches to eliminate PCM writes by migrating frequently-written pages to DRAM. Prior approaches have two drawbacks: (1) they operate at a coarse-grained page granularity, (2) they are reactive.

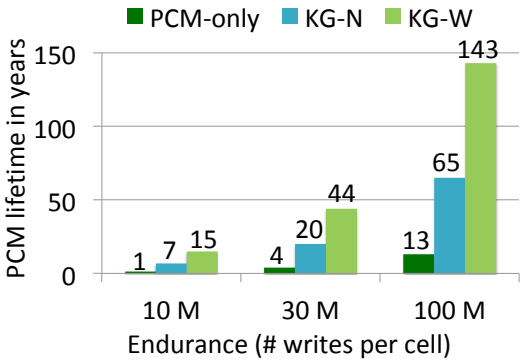

Figure 1. PCM-only is impractical. 32 GB lasts only 4 years on average with 30 M writes per cell and hardware line wear-leveling in simulation.

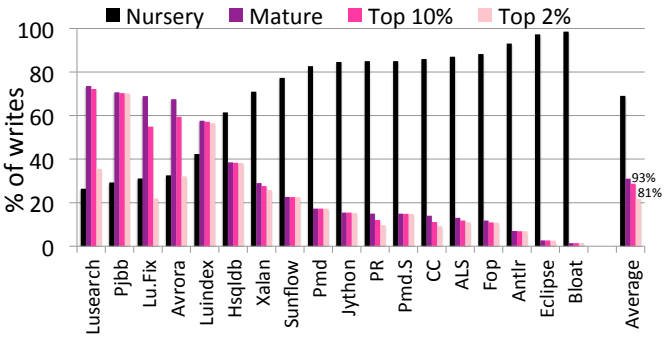

Figure 2. Nursery objects incur 70% of writes and mature objects incur 30% on average. The top 10% of written mature objects incur 93% of mature writes and the top 2% incur 81%.

Figure 1 shows empirical measurements of PCM lifetimes when executing Java applications. A 32-core PCM-only system with 32 GB of PCM main memory and an endurance of 30 M writes would wear out in 4 years, even with wear-leveling. Because lifetime is a linear function of memory capacity and writes, increasing endurance to 100 M would improve lifetime to 13 years. However, the worst-case lifetime,

i.e., running the Java application with the highest write rate, is less than 5 years. These results show that even with optimistic endurance levels, a pure PCM-only system is impractical. A recent paper describes our experimental methodology [2].

This work uses the managed runtime implementation for languages such as Java, C#, JavaScript, and Python to better utilize hybrid DRAM-PCM memories. We introduce a new class of *write-rationing* garbage collectors that reorganize objects to limit writes to PCM in hybrid main memories. Write-rationing garbage collection aims to keep frequently-written objects in DRAM to improve PCM lifetime, and keeps read-mostly objects in PCM to exploits PCM capacity.

A detailed analysis of write behaviors in Java applications motivates the design and implementation of write-rationing garbage collectors. Figure 2 presents write behaviors in an instrumented generational collector as a function of object age: young (nursery) versus mature space objects. 26% to 99% of writes occur to nursery objects, averaging 70%. Of the writes to mature objects, 2% of mature objects capture 81% of these writes. Thus, 93% of all writes fall in one of these two categories. Our write-rationing garbage collectors exploit the correlation between writes and object demographics (age), and place young and the small fraction of mature objects that incurs the most writes in DRAM.

In summary, we contribute the design and implementation of write-rationing garbage collectors to manage hybrid memories, minimizing PCM writes while maximizing the use of PCM capacity. Our work introduces a practical approach to exploit hybrid memories, and requires no changes to hardware and the programming model.

2 Write-Rationing Garbage Collection

We introduce two Kingsguard write-rationing collectors that protect PCM from writes and aim to exploit PCM capacity. The heap memory is organized as DRAM and PCM virtual memory, and the OS maps the heap memory to physical memory in DRAM or PCM. Our first collector, called *Kingsguard-nursery* (KG-N), allocates new objects in a DRAM nursery, since they incur between 26% and 99% of writes, and then promotes all nursery survivors to a PCM mature space.

Kingsguard-writers (KG-W) adds support for fine-grained monitoring of objects. It also uses a DRAM nursery, but promotes nursery survivors to a DRAM *observer* space. The Java Virtual Machine (JVM) tracks all mature object writes using a write-barrier. Observer space collections copy objects with zero writes from the observer space to the PCM mature space and copy any written objects to the DRAM mature space, using past writes to predict future writes. Kingsguard-writers promotes most observer space survivors (90%) to PCM memory, thus exploiting its capacity. When it detects written objects in PCM, it moves them back to DRAM during a full-heap collection. KG-W also includes optimizations for managing large objects and JVM meta data.

Large objects Most collectors manage large objects in a separate heap space due to high copying costs. KG-W places large objects in PCM to exploit its capacity. KG-W moves large objects to DRAM during a full-heap collection in case they get writes. KG-W smartly allocates some large objects in the nursery first to give them a chance to die.

Meta data Garbage collectors require metadata to track object liveness. GenImmix stores the mark state of objects in their headers. This results in PCM writes during a full-heap collection due to writes to objects' mark states. KG-W decouples the mark state meta-data from objects and stores it in DRAM to avoid PCM writes.

3 Results

Our first implementation of Kingsguard collectors uses cycle-level simulation because hybrid memory systems are not available [2]. We find that KG-N alone improves PCM lifetime by 5×, whereas KG-W improves lifetime by 11× over PCM-only systems (see Figure 1). For 16 Java applications, KG-W needs between 26 to 40 MB of DRAM to achieve these lifetimes. We implement and compare to state-of-the-art OS write partitioning (WP) [3]. OS WP consumes about the same amount of DRAM, but WP has over 3× more writes to PCM than KG-W.

PCM consumes no idle power. Kingsguard-nursery reduces the energy-delay product by 36% over a DRAM-only system and 33% over a PCM-only system. Kingsguard-writers adds 5% average overhead to monitor nursery survivors in the observer space and to copy objects between spaces. Kingsguard-writers reduces the energy-delay product: 32% on average over DRAM-only and 29% over a PCM-only system.

More recently, we have implemented Kingsguard collectors on a real machine [1]. Real hardware results validate the findings of our results in simulation [2].

4 Conclusions

This paper introduces write-rationing garbage collectors, which seek to maximize the use of PCM while improving its lifetime in hybrid DRAM-PCM memory systems. Our proposed Kingsguard collectors exploit object write behaviors in Java applications to manage hybrid memories. They improve over, and complement, prior hardware and OS approaches. This work opens up a new and promising direction to manage hybrid main memory systems.

References

[1] Shoaib Akram, Jennifer B. Sartor, Kathryn S. McKinley, and Lieven Eeckhout. 2018. Emulating Hybrid Memory on NUMA Hardware. In *Technical Report at arXiv.org*.

[2] Shoaib Akram, Jennifer B. Sartor, Kathryn S. McKinley, and Lieven Eeckhout. 2018. Write-rationing Garbage Collection for Hybrid Memories. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[3] Wangyuan Zhang and Tao Li. 2009. Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures *(PACT)*.