Understanding the Limiting Factors of Page Migration in Hybrid Main Memory

Santiago Bock Bruce R. Childers Rami Melhem Daniel Mossé
Department of Computer Science
University of Pittsburgh
{sab104, childers, melhem, mosse}@cs.pitt.edu

ABSTRACT

Combining DRAM and *non-volatile memory* (NVM) has been proposed to increase capacity and reliability, and to decrease energy consumption. *Software-managed hybrid memory* is a promising way to incorporate NVM in main memory due to its architectural simplicity. However, there are significant performance issues caused by interference in the memory system due to data migration between DRAM and NVM and a lack of effective migration policies.

We propose new analysis and simulation techniques to understand the behavior of software-managed hybrid memory. These techniques allow us to characterize the overhead experienced by requests in the memory hierarchy and identify the factors that limit performance in software-managed hybrid memory. We show that queuing delays at the NVM banks and NVM bus are the main limiting factors.

1. INTRODUCTION

Researchers have proposed using new non-volatile memory (NVM) technologies, such as Phase Change Memory (PCM) or Spin-Transfer Torque Memory (STT-RAM), either as full DRAM replacements or in *hybrid* memories, where a small and fast DRAM is backed by a larger and slower NVM [3, 4]. In *software-managed* hybrid memory, both DRAM and NVM are directly addressable, and the operating system (OS) *migrates data* between DRAM and NVM [1, 2, 5] to improve performance and reduce energy consumption.

Page migration allows the OS to move frequently used data to DRAM and keep colder data in NVM. If done correctly, page migration can have a significant effect on performance: Figure 1 shows average speedup of the state-of-the-art [5] and an ideal system that uses an oracle page migration policy and zero-interference migration (the baseline is a system that does no migration). The x-axis shows *migration rate*, which is the fraction of execution time during which the OS migrates pages. At low rates, the state-of-the-art system performs close to the baseline. At high rates, however, it does worse due to interference between application and migration memory requests. An ideal memory system with no interference and a "good" migration policy can achieve high performance, especially at high migration rates.

CF'15, May 18-21, 2015, Ischia, Italy
ACM 978-1-4503-3358-0/15/05.
http://dx.doi.org/10.1145/2742854.2742901.

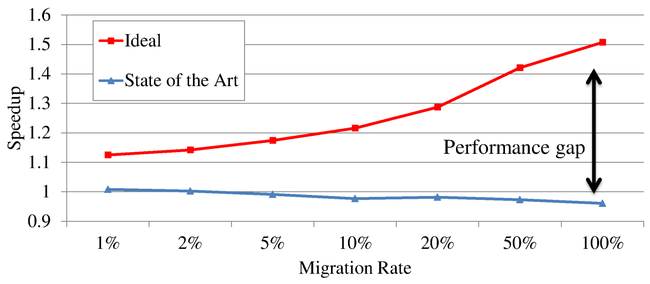

Figure 1: Average speedup over no-migration for different migration rates.

This paper presents analysis and simulation techniques to investigate the nature of this performance gap and understand how to improve software-manage hybrid memory. We characterize the overhead of page migration and study the delays that applications experience in the memory hierarchy.

2. LIMITING FACTORS

Migration can have a negative impact on performance because the latency of regular requests increases due to interference from migration memory traffic. The following are performance-limiting factors in hybrid memory.

Migration Policy. The OS does not always make the best decisions for page migration because it has coarse-grain information about application memory accesses and cannot fully analyze available data. The difference between a realistic policy and an "ideal" one is *migration policy overhead.*

Cache Flush. A migrated page needs to be flushed from the L1 and L2 caches due to the changed physical address. The flush from L1, and especially L2, can result in an increased cache miss rate for application requests.

Bank Contention. Page migration causes many requests to a single memory bank. Application requests to the same bank will wait longer in the queue, increasing overhead.

Bus Contention. Page migration increases bus utilization due to additional memory traffic. Application requests that share the bus with migration traffic will wait longer in the queue, increasing overhead.

Row Buffer Interference. Most applications access memory in regular patterns, which can be exploited to increase performance by keeping the bank row open. Migration requests to or from banks with high application access locality interfere with open row buffers, increasing the row buffer miss rate seen by application requests.

3. NEW TECHNIQUES

To design hardware mechanisms and migration policies for hybrid memory that perform close to an ideal system, we must first determine how much each of the factors presented in Section 2 contributes to the overhead of page migration. We propose techniques to analyze memory access latency and determine the importance of the limiting factors.

Zero-Interference Migration. We propose a hypothetical system that is not affected by memory traffic and other costs associated with migration. Migration still has the same latency as the realistic full-interference case. However, accesses to DRAM and NVM from migration do not interfere with regular application requests, and other side-effects, such as flushing the caches, are modified for zero-interference.

Offline Migration Policy. This migration policy (an oracle) uses future memory access counts to estimate the relative importance of pages. It migrates only pages that will benefit performance in the future.

Memory Latency Attribution Analysis (MLAA). This analysis determines the average number of cycles that read requests from the CPU spend in different memory structures (caches, queues, buses and banks). MLAA captures the relative importance of each component of the memory hierarchy in the total latency experienced by requests from the CPU, enabling comparison of different configurations.

4. RESULTS

We applied MLAA to a simulated hybrid memory system with DRAM and PCM running a subset of the SPEC CPU2006 benchmark suite. We compare two interference regimes: **Full-Interference** is a realistic memory and **Zero-Interference** is the scheme from Section 3. Hybrid memory is managed by the offline migration policy.

The MLAA graphs are shown in Figure 2. On average, the number of cycles can be reduced by 24% when going from Full-Interference to Zero-Interference. This reduction comes mostly from three sources: PCM bus queue (10%), PCM open (6%) and PCM bank queue (4%). In Full-Interference, the PCM bus and bank queues become saturated with migration requests, which delays regular requests. The reduction in PCM open time is due to a lower row buffer miss rate. These results suggest that, on average, most of the overhead of page migration is caused by interference of migration requests with application requests.

Other components of the memory do not have a significant reduction on cycle count even though they contribute considerably to the total cycle count. In particular, the L1 tag time is the same for both interference regimes because migration does not change how often the L1 tag is accessed. Flushing the L1 after migration does change the L2 access count, but to a very small degree due to the small size of the L1. The DRAM components of memory also exhibit little change in cycle count. This is because DRAM is faster and therefore has more idle time, allowing migration requests to proceed with less interference.

The behavior of individual benchmarks varies widely. Some benchmarks benefit little (cycle count reduction of less than 2%) from Zero-Interference migration. *454.calculix* and *456.hmmer-1* are dominated by L1 and L2 tag access cycles, so changes in other components of the system have no effect. Cycle counts for *401.bzip2-5* are more evenly distributed across the hierarchy, but do not change. In *403.gcc-7*, cycles

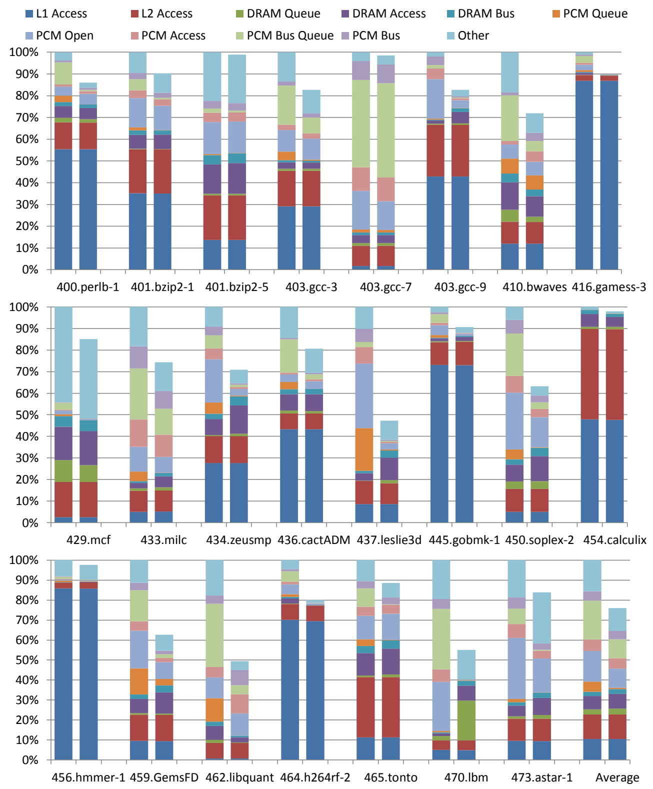

Figure 2: MLAA for Offline. For each workload, first bar is Full-Interference and second bar is Zero-Interference.

counts for PCM open decrease by the same amount that PCM bus queue cycles increase, resulting in similar performance. In this case, a decrease in row buffer miss rates (less cycles opening rows) causes a reduction in service time that is shifted to the queue.

5. CONCLUSION

Due to its simplicity, software-managed hybrid memory is a promising way to incorporate NVM in main memory. However, the lack of effective migration policies to managed DRAM and NVM, coupled with hardware overhead, leads to poor performance. This paper presents new analysis and simulation techniques to aid in the development of new policies and hardware mechanisms for low-cost migration.

6. ACKNOWLEDGMENT

This material is based upon work supported by the NSF under Grants No. CNS-1012070 and CCF-1422331.

7. REFERENCES

[1] S. Bock, B. R. Childers, R. Melhem, and D. Mosse. Concurrent page migration for mobile systems with OS-managed hybrid memory. In *Int'l. Conference on Computing Frontiers*, 2014.

[2] G. Dhiman, R. Ayoub, and T. Rosing. PDRAM: a hybrid PRAM and DRAM main memory system. In *Design Automation Conference*, pages 664–469, 2009.

[3] A. P. Ferreira, B. Childers, R. Melhem, D. Mossé and, and M. Yousif. Using PCM in next-generation embedded space applications. In *Real-Time and Embedded Technology and Application Symposium*, 2010.

[4] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, pages 24–33, 2009.

[5] L. Ramos, E. Gorvatov, and R. Bianchini. Page placement in hybrid memory systems. In *International Conference on Supercomputing*, 2011.