Migration Based Page Caching Algorithm for a Hybrid Main Memory of DRAM and PRAM

Hyunchul Seok
KAIST
Daejeon, Korea
hcseok@core.kaist.ac.kr

Youngwoo Park
KAIST
Daejeon, Korea
ywpark@core.kaist.ac.kr

Kyu Ho Park
KAIST
Deajeon, Korea
kpark@ee.kaist.ac.kr

ABSTRACT

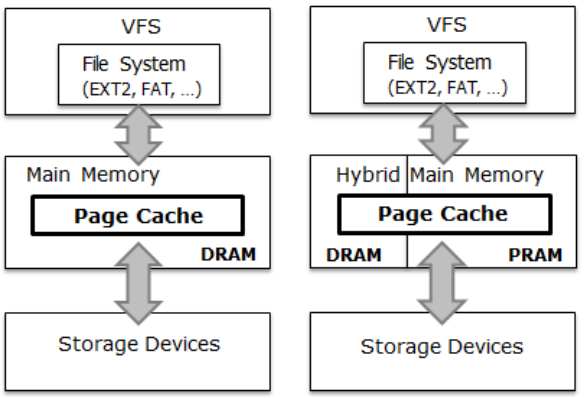
As the DRAM based main memory significantly increases the power and cost budget of a computer system, new memory technologies such as Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM) have been proposed to replace the DRAM. Among these memories, PRAM is the most promising candidate for large scale main memory because of its high density and low power consumption. In previous researches, a hybrid main memory approach of DRAM and PRAM is adopted to make up for the latency and endurance limits of PRAM. On the other hand, large amount of a main memory is used for page cache to hide disk access latency. Many page caching algorithms such as LRU, LIRS, and CLOCK-Pro are developed and show good performance, but these are only consider the main memory with uniform access latency and unlimited endurance. They cannot be directly adapted to the hybrid main memory architecture with PRAM.

In this paper, we propose a new page caching algorithm for the hybrid main memory. It is designed to overcome the long latency and low endurance of PRAM. On the basis of the LRU replacement algorithm, we propose page monitoring and migration schemes to keep read-bound access pages to PRAM. The experiment results show that our page caching algorithm minimize the write access of PRAM while maintaining cache hit ratio. The results show that we can maximally reduce the total write access count by 48.4%. Therefore, we can enhance the average page cache performance and reduce the endurance problem in the hybrid main memory.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; D.4.2 [**Operating Systems**]: Storage Management—*Allocation/deallocation strategies, Main memory*

SAC'11 March 21-25, 2011, TaiChung, Taiwan.



(a) On only DRAM (b) On DRAM and PRAM

Figure 1: Page Caching System Architecture

General Terms

Page cache algorithm

Keywords

Page Cache, Hybrid Main Memory, PRAM, Migration

1. INTRODUCTION

For several decades, DRAM has been the main memory of computer systems. Recent studies have shown that DRAM based main memory significantly increases the power and cost budget of a computer system [2]. Therefore, new memory technologies such as Phase-change RAM (PRAM), Ferroelectric RAM (FRAM), and Magnetic RAM (MRAM) have been proposed to replace the DRAM.

Among these memories, PRAM is the most promising candidate for large scale main memory because of its high density and low power consumption. While DRAM stores each bit of data in a separate capacitor, PRAM uses phase of material to represent each bit. PRAM density is expected to be much larger than DRAM (about four times). Also, PRAM is a non-volatile memory because the phase of material does not change after power-off. It has negligible leakage energy regardless of the size of memory. Table 1 shows the properties of PRAM compared to DRAM[5].

However, the access latency of PRAM is still not comparable to DRAM latency. It also has worn-out problem caused by limited write endurance. In previous researches,

Table 1: Properties of PRAM

Attributes	DRAM	PRAM
Non-volatility	No	Yes
Cost/TB	Highest	Low
Read Latency	50ns	50-100ns
Write Latency	20-50ns	~1us
Read Energy	~0.1nJ/b	~0.1nJ/b
Write Energy	~0.1nJ/b	~0.5nJ/b
Idle Power	~1.3W/GB	~0.05W
Endurance	∞	10^8 for write

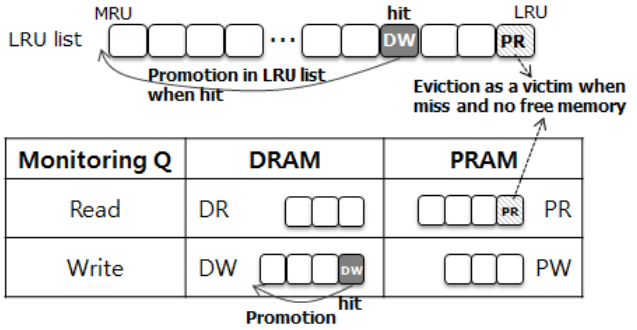


Figure 3: Overview of Page Caching Algorithm

a hybrid main memory approach of DRAM and PRAM is adopted to make up for the latency and endurance limits of PRAM [3][8][6]. They propose several hardware and software schemes to manage the hybrid main memory.

On the other hand, in modern computer system, large amount of a main memory is used for page cache to hide disk access latency, as shown in Figure 1(a). Many page caching algorithms such as LRU, LIRS [4], and CLOCK-Pro [7] are developed and show good performance for current DRAM based main memory. However, previous page caching algorithms only consider the main memory with uniform access latency and unlimited endurance. They cannot be directly adapted to the hybrid main memory architecture with DRAM and PRAM, as shown in Figure 1(b).

In this paper, we propose the new page caching algorithm for the hybrid main memory. It is designed to overcome the long latency and endurance problem of PRAM. On the basis of conventional cache replacement algorithms, we propose page monitoring and migration schemes to keep read-bound access pages to PRAM. It minimizes the write access of PRAM while maintaining cache hit ratio. Therefore, we can enhance the average page cache performance and reduce the endurance problem in the hybrid main memory.

The remainder of paper is organized as follows. In Section 2 we present the design of our algorithm and we describe page monitoring and migration strategy. The performance evaluation results are shown in Section 3, and we briefly discuss the future works in Section 4. Section 5 concludes this paper.

2. THE PAGE CACHING ALGORITHM FOR THE HYBRID MAIN MEMORY

In this section, we describe a design of a new page caching algorithm on the hybrid main memory, which consists of DRAM and PRAM. Although the conventional cache algorithms show good performance, they cannot be directly adapted to the hybrid main memory. Because the hybrid main memory uses two different types of memories, it is important to consider their properties to maximize a performance and efficiency. As shown in Table 1, PRAM has very long latency comparing to DRAM for writing a page. If the page cache algorithm causes a lot of writes to PRAM, the average page cache performance gets worse. In addition to the write latency, PRAM has low endurance comparing to DRAM. If there are many write accesses, PRAM will be worn-out quickly. Therefore, the page cache algorithm on the hybrid main memory should be designed to overcome the long latency and endurance problem of PRAM.

2.1 Overview

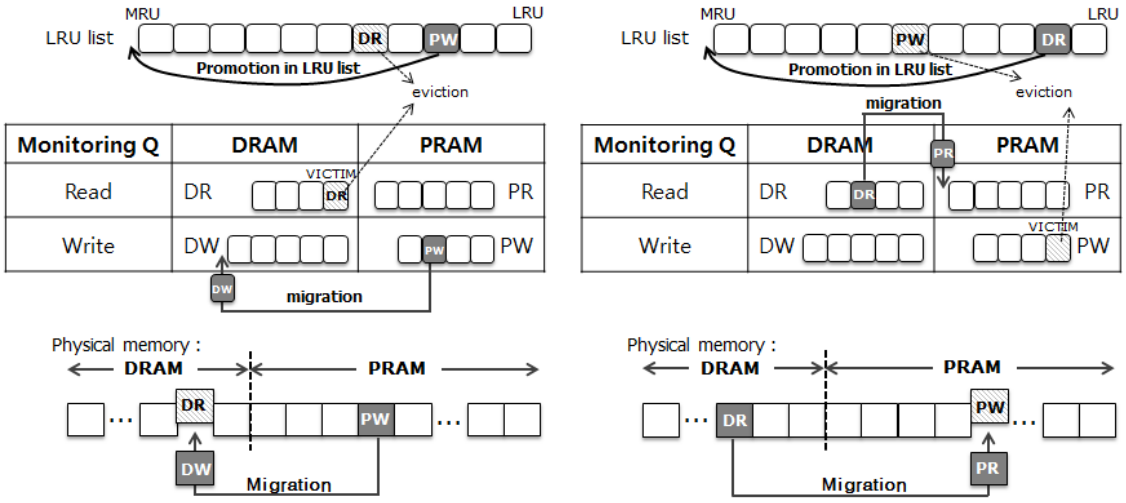
To satisfy the requirements, we designed the new page caching algorithm with the conventional cache algorithm. We add the page monitoring and migration schemes to keep read-bound access pages to PRAM and write-bound access pages to DRAM. Although any conventional cache replacement algorithms can be adapted, we choose the LRU replacement algorithm which is very simple but well operated. In order to implement page monitoring and migration schemes we use four monitoring queues, which consist of DRAM read queue, DRAM write queue, PRAM read queue, and PRAM write queue. When one page block will access, it contains to both the LRU list and one of the four queues by its access pattern and the memory type where it is located.

Figure 3 shows the basic operation of our algorithm. We use the LRU replacement algorithm for managing the page caches. When a page fault occurs, we put the page block into the MRU position in the LRU list. In addition, we put the page block to the one of the monitoring queues according to the page access type. For example, if the page block's access pattern is read and a selected memory page is on DRAM, we put the page block to the DRAM read queue. If there is no free memory when a miss occurs at page fault, the least recently used page block is selected as the victim page, which is the LRU replacement algorithm. At the same time we evict the page which related to the victim page. For example, if the victim page is write cache and resided in PRAM, we eliminate the page from PRAM write queue. When the page hits, the page in both the LRU list and the monitoring queue is promoted, as shown in Figure 3.

In order to monitor the behavior of pages, we manage the four monitoring queues. We put the information of read-bound pages on PRAM and DRAM into two read queues of PRAM and DRAM, respectively. Similarly, write queues contain the information of write-bound pages of PRAM and DRAM. When the pages are accessed, the page's access pattern is predicted by the access count and then we put the page into the monitoring queues by its access pattern.

2.2 Migration Strategy

In case of the write-bound pages on PRAM, they cause the performance degradation and worn-out problem because of PRAM's long latency and low endurance. To reduce the bad effects of writes on PRAM, we move the write-bound caches from PRAM to DRAM. Additionally, in case of read-



(a) Migration of the write-bound page

(b) Migration of the read-bound page

Figure 2: Migration of Cached Pages between DRAM and PRAM

bound caches, we move them from DRAM to PRAM because we can gather the read-bound caches to PRAM as much as possible. As a result, we can reduce the number of writes on PRAM and keep read-bound access pages to PRAM.

There are two migration cases as shown in Figure 2; one is the migration of write-bound caches from PRAM to DRAM and another is the migration of read-bound caches from DRAM to PRAM. To determine the migration, we roughly use a simple method. If the write access is hit and the accessed page is in the PRAM write queue, the migration is triggered, as shown in Figure 2(a). Similarly, if the read access is hit and the accessed page is in the DRAM read queue, the migration is started, as shown in Figure 2(b).

In Figure 2(a), it shows that the page is migrated to the DRAM write queue. If there is no free space in DRAM, we have to select a victim page on DRAM. In this case, we select the victim page from the bottom of the DRAM read queue. In case of the physical memory view, the selected victim page on DRAM must be removed, and the write-bound page on PRAM is moved to the DRAM where the victim page was located. In the DRAM write queue, this page is put into the top of the queue. If there is no element of the DRAM read queue when we find a victim page for migration, we choose a victim page from the bottom of the DRAM write queue, which means that the victim page is the least recently used. In case of the migration of the read-bound pages, it is similar to the case of the migration of the write-bound pages, as shown in Figure 2(b). For selecting a victim page, we select the bottom page of PRAM write queue first, and if there is no pages on PRAM write queue, we choose the bottom page of PRAM read queue. When we remove a victim page for migration, we just remove it from the memory, the LRU list, and the monitoring queue. The reason why we do not change the pages between the victim page for migration and the page which will migrate is that it causes the additional write on PRAM.

3. EXPERIMENT

In this section, we explain the evaluation environments

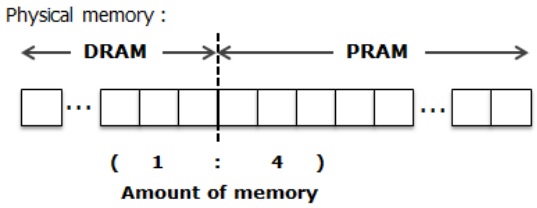


Figure 4: Composition of The Hybrid Main Memory

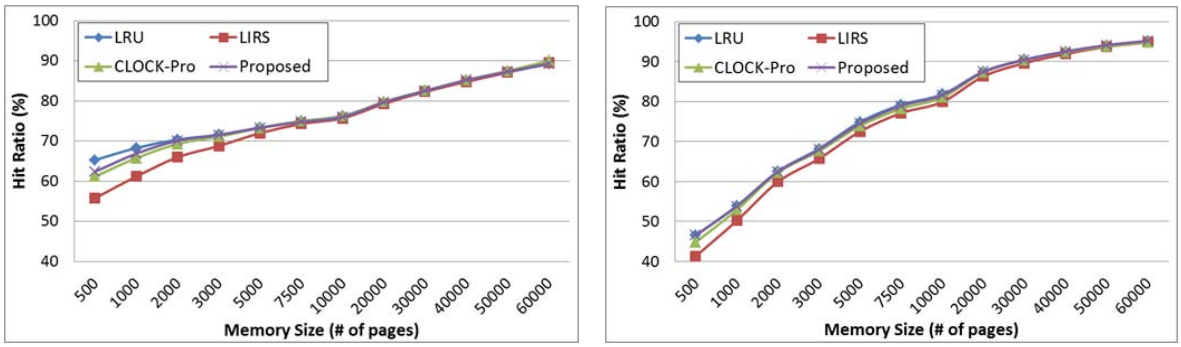
such as the design of the hybrid main memory and performance metrics. We use trace-driven simulation to evaluate our hybrid cache algorithm. We then summary its performance results which are hit ratio and write access count on PRAM compared with the conventional page caching algorithm such as LRU, LIRS, and CLOCK-Pro.

3.1 Experiment Setup

We implemented our proposed page cache algorithm based on the LRU trace simulator because our algorithm is based on the LRU replacement algorithm. In order to evaluate the performance of our page cache scheme, we have to define the hybrid main memory. We assume that the hybrid main memory consists of DRAM and PRAM, which are divided by a memory address. The memory area pointed by low memory address is for DRAM and the high section is allocated to PRAM as shown in Figure 4. Because PRAM density is expected to be four times larger than DRAM, the PRAM-to-DRAM ratio is four. As the workload for performance evaluation, we use two workload traces, which are extracted from OLTP application I/O running at two large financial institutions. These traces are made available courtesy of Ken Bates from HP, Bruce McNutt from IBM, and the Storage Performance Council [1].

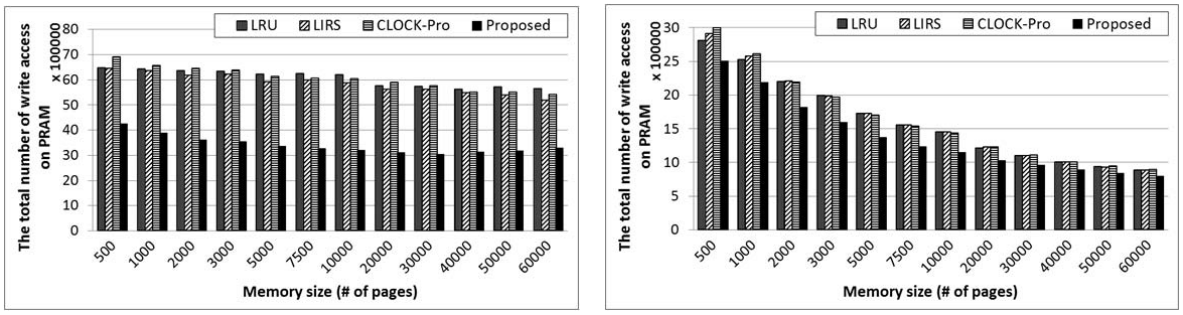
3.2 Evaluation Results

We give trace-based simulation results for showing the performance of our page caching algorithm. We compared



(a) *Financial1* (b) *Financial2*

Figure 5: Hit Ratio of Proposed Algorithm and Conventional Algorithms on workloads *financial*



(a) *Financial1* (b) *Financial2*

Figure 6: The total write access count of PRAM on workload *financial*

the experiment results with those of the conventional page caching algorithms such as LRU, LIRS, and CLOCK-Pro. The two traces of workloads *financial* are used to evaluate the performance.

3.2.1 Hit ratio

The first evaluated result is the hit ratio, which is important to determine the performance of the page caching algorithm. We compared the hit ratio of our page caching algorithm to the hit ratios of the conventional page caching algorithms. We tested it with many sizes of the main memory. The results are shown in Figure 5.

From the results of the hit ratio, the LRU replacement algorithm shows the highest hit ratio through the whole memory sizes. Although the hit ratio of our algorithm is lower than that of LRU, but the results show that the hit ratio is similar to LRU algorithm and higher than LIRS and CLOCK-Pro. Actually, we designed our algorithm with the basis of the LRU replacement algorithm and added the page monitoring and migration schemes. We expected the hit ratio of our algorithm is similar to that of the LRU replacement algorithm. However, because we designed the selected victim page for migration is just eliminated, explained in the section 2, it can cause the page faults more. As a result, the hit ratio is lower than the LRU. Although the migration scheme causes the degradation of the hit ratio, the hit ratio is still larger than that of LIRS and CLOCK-Pro at small sizes of the main memory. And at large size of the memory, four algorithms show the same hit ratio.

3.2.2 Write access count on PRAM

Write access count on PRAM is important because it is related of the total latency of page cache and the lifetime of PRAM. PRAM has very long latency compared to DRAM so that the performance of page caches is degraded if many write pages hit on PRAM. In addition, PRAM becomes worn-out more quickly because of its low endurance. Therefore, in order to use a page cache on the hybrid main memory of DRAM and PRAM, a page caching algorithm must be able to reduce the write access count on PRAM.

In this section, we evaluated the write access count on PRAM and compare it with the results of the conventional algorithms. During the simulation, we counted read and write accesses of DRAM and PRAM. Figure 6 shows the total number of write accesses on PRAM with workloads which are *financial1* and *financial2*. We measured the count number with several memory sizes. The memory size is the total size of DRAM and PRAM. In Figure 6, the number of accesses of four algorithms decreases as the memory size grows. When a page fault occurs, it increases the write access count even if a pattern of the page is read access. Because the number of faults is decreased by increasing the size of memory, the total number of write accesses on PRAM decreases when the memory size increases.

When using our algorithm, we can know the total number of write accesses is reduced compared to the conventional page caching algorithm. We can reduce the total write access count on PRAM by 34.4% on *financial1* and 10.8% on *financial2* when we use the hybrid main memory with 500-

Table 2: Average Write Access Counter of DRAM and PRAM on workload *financial1*

Type	LRU	LIRS	CLOCK-Pro	Proposed
DRAM	16309	16248	12839	54627
PRAM	16186	16130	13327	10631

Table 3: Average Write Access Counter of DRAM and PRAM on workload *financial2*

Type	LRU	LIRS	CLOCK-Pro	Proposed
DRAM	7006	7328	4254	10562
PRAM	7018	7286	4582	6828

page sizes. We can maximally reduce the total write access count by 48.3% on *financial1* and 20.9% on *financial2*.

Table 2 and Table 3 show the average of write access on DRAM and PRAM. Because we move the write-bound pages to DRAM and keep the read-bound pages on PRAM as much as possible, the average of write access count on DRAM is increased and that of PRAM is decreased in both two workloads. We compare the gain of the latency, approximately. If we compare the results of LRU and our proposed method on the workload *financial1*, the increased average number of writes on DRAM is 38318 and the decreased average number of writes on PRAM is 5555. From Table 1, we choose the latencies; write latency of DRAM is 50ns and write latency of PRAM is 1us. If we calculate the latencies with these values, the average increased write latency on DRAM is 1.9msec but the decreased write latency on PRAM is 5.6msec so that we can conclude that the case of our algorithm can reduce the total latency of page cache.

Actually, PRAM can increase the total amount of a main memory with the same area because of its high density, compared to the amount of a main memory made by only DRAM. It means that we can use the larger amount of the hybrid main memory than the main memory with only DRAM and we can easily get the higher hit ratio when using the hybrid main memory. With this advantage, we can increase the system performance by using our page caching algorithm. It can reduce the effect of PRAM's disadvantages such as long write latency and low endurance by keeping the write-bound pages to DRAM and the read-bound pages to PRAM as much as possible.

4. FUTURE WORKS

In the proposed migration scheme, we determined the page migration by the page monitoring with four monitoring queues. In this paper, we use very simple idea, described in section 2. It is simple but well operated, shown in the experiment results. However, the precise prediction to the page's pattern will be enhanced the performance more. Actually, page blocks may be experienced with many read and write access through the operations, and read and write accesses may be repeated. Therefore, the well-organized algorithm for predicting the page's pattern is needed.

As the next future work, we conducted the trace-based simulation of the performance. But in this case, we cannot exactly evaluate the effect of PRAM's properties. We have to experiment how much the total performance is increased and how much the latency is decreased when we use our page caching algorithm on the hybrid memory.

5. CONCLUSIONS

We propose a new page caching algorithm on the hybrid main memory. The hybrid main memory is organized by heterogeneous types of memories, which have different properties such as the access latency, density, and endurance. In modern computer system, large amount of a main memory is used for page cache to hide disk access latency. Because the conventional page caching algorithms only deal with the uniform access latency and endurance, a new page caching algorithm is needed. When using the PRAM for making the hybrid main memory, the considering things are the long write latency and low endurance of PRAM. Therefore, we proposed the page caching algorithm with page monitoring and migration schemes to keep read-bound access pages to PRAM and write-bound access pages to DRAM.

The experiment results show that out algorithm can reduce the total number of write accesses by 48.3%, maximally. In case of hit ratio, it shows the similar hit ratio to the conventional page caching algorithms such as LRU, LIRS, and CLOCK-Pro. It minimizes the write access of PRAM while maintaining cache hit ratio. Therefore, we can enhance the average page cache performance and reduce the endurance problem in the hybrid main memory.

6. ACKNOWLEDGMENTS

The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.

7. REFERENCES

[1] UMass TraceRepository, *http://traces.cs.umass.edu/index.php/Storage/Storage*.

[2] L. A. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[3] G. Dhiman, R. Ayoub, and T. Rosing. Pdram:a hybrid pram and dram main memory system. *In Proceedings of the 46th Annual Design Automation Conference*, July 2009.

[4] S. Jiang and X. Zhang. Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Marina Del Rey*, pages 31–42. ACM Press, 2002.

[5] K. H. Park, Y. Park, W. Hwang, and K.-W. Park. Mn-mate: Resource management of manycores with dram and nonvolatile memories. *12th IEEE International Conference on HPCC*, September 2010.

[6] Y. Park, S. K. Park, and K. H. Park. Linux kernel support to exploit phase change memory. *Linux Symposium*, July 2010.

[7] S. J. Performance and S. Jiang. Clock-pro: An effective improvement of the clock replacement. In *Proceedings of USENIX Annual Technical Conference*, 2005.

[8] M. K. Qureshi, V. Srinivassan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. *In Proceedings of the 36th Annual International Symposium on Computer Architecture*, June 2009.