# A Buffer Cache Algorithm for Hybrid Memory Architecture in Mobile Devices

Chansoo Oh†‡, Dong Hyun Kang†, Minho Lee†, and Young Ik Eom†

†College of Information and Communication Engineering, Sungkyunkwan University
Suwon, 440-746, Korea
‡Hanwha Techwin, Korea
{chansoo.oh,kkangsu,minhozx,yieom}@skku.edu

**Abstract.** In general computing environments including mobile devices, buffer cache algorithm is generally used to mitigate the performance gap between CPU and secondary storage. However, traditional DRAM-based buffer cache architecture reveals a power consumption problem in mobile devices, because it periodically performs the refresh operations to maintain data in DRAM. In addition, traditional buffer cache algorithms never consider the states of mobile applications (e.g., foreground and background state). In this paper, we propose a novel buffer cache algorithm, which efficiently addresses the above issues based on hybrid main memory architecture that is comprised of DRAM and PCM. Our algorithm is motivated by key observation that background applications on mobile device rarely issue I/O requests as well as they can degrade the performance of foreground applications because of the interferences among the I/O requests of applications. For evaluation, we implemented our algorithm and compared its performance against two other algorithms. Our experimental results show that our algorithm reduces the elapsed time of the foreground applications by 53% on average and the power consumption by 23% on average without any negative performance effects on background applications.

**Keywords:** hybrid memory system, buffer cache algorithm, mobile device, foreground application, background application

## 1 Introduction

Today's mobile devices (e.g., tablets and smartphones) require a significant amount of memory because many applications run simultaneously and they have to store their data in main memory. However, the traditional DRAM-based memory architecture suffers from the periodic refresh operations that consume the battery power of device to maintain data in DRAM (Dynamic Random Access Memory). In particular, traditional buffer cache algorithms have been designed without consideration on the dynamic state changes of mobile applications (e.g., foreground and background state). Therefore, many researchers focused on non-volatile memory, such as PCM (Phase Change Memory), to take the benefits of lower power

consumption over DRAM. However, read/write latency of PCM is slower than that of DRAM and PCM has limited lifecycle (Table 1) [1]. In order to mitigate these weaknesses of PCM, some researchers proposed buffer cache algorithms that efficiently reduce the number of PCM writes by using hybrid main memory architecture, which consists of DRAM and PCM [2], [3], [4]. However, previous studies have only focused on desktop applications. There is no prior work which considers the state of mobile applications, such as foreground and background state.

**Table 1.** Characteristics of DRAM and PCM

| | Read Latency | Write Latency | Read Energy | Write Energy | Static Energy | Endurance |
|---|---|---|---|---|---|---|
| DRAM | 50ns | ~20-50ns | ~0.1nJ/b | ~0.1nJ/b | ~W/GB | $\infty$ |
| PCM | 50~100ns | ~1us | ~0.1nJ/b | ~1nJ/b | <<0.1W | $10^8$ |

In this paper, we propose a novel buffer cache algorithm that saves power consumption and improves the performance of foreground applications based on hybrid main memory architecture comprised of DRAM and PCM. Our algorithm is motivated by two key observations: (1) background applications on mobile device rarely issue I/O requests and (2) they can degrade the performance of foreground applications because of interference in I/O requests between foreground and background applications. Based on these observations, we first classify I/O requests into foreground I/O and background I/O, and then place foreground I/O in DRAM and background I/O in PCM because background I/O has no direct impact on the user experience [5].

For evaluation, we implemented our algorithm and compared its performance against two representative algorithms, such as traditional LRU (Least Recently Used) and 2QLRU (2 Queue Least Recently Used) [6]. Our experimental results clearly show that our algorithm reduces the elapsed time of the foreground applications by 53% on average and the power consumption by 23% on average without any negative performance effects on background applications.

The remainder of the paper is organized as follows. Section 2 explains the related works, such as PCM and Android system. Then we present design of our algorithm in Section 3. Section 4 evaluates its performance while comparing it with previous buffer cache algorithms. We conclude our research and suggest future work in Section 5.

## 2 Background

Mobile device has a limitation in expanding hardware such as CPU, memory, and battery because it should have reasonable price, portability, and small size. Especially, high performance hardware needs much more energy to operate than old devices. Battery capacity is limited due to the fixed size of mobile device. So, various methods which try to reduce power consumption to preserve battery capacity have been studied so far [7], [8], [9], [10], [11]. Also, mobile device can run several

applications at the same time because it provides a multitasking environment. These applications could be classified as foreground one and background one depending on its state. Only a few applications can run as foreground applications and others should run as background applications. Foreground applications produce a large number of I/O requests while interacting with the user, whereas background applications rarely issue I/O requests, which are generated by push services or audio file playbacks. Generally, foreground applications run with high priority because users can perceive performance of the foreground applications as the performance of the mobile device. On the other hand, background applications are running with lower priority than foreground ones because they do not need fast response and do not affect directly to the user.

Android is the one of the most popular software platforms for mobile devices [12]. It has been evolved to be suitable with mobile environment. Especially, Android adopts various functions from Linux kernel, such as wakelocks [13], binder [14], and lowmemorykiller [15], to customize the kernel for mobile device. In Android, many applications, which are currently not being used by user, are maintained in the background state until being killed by LMK (lowmemorykiller) or changed to foreground state. LMK kills processes to get enough memory space to run new processes. It chooses a background process which has a lowest priority at that time. Android has a standard classification about the process states called 'importance hierarchy' which consist of foreground process, visible process, service process, background process, and empty process [5]. In this classification, background process and empty process do not affect any direct effect to the foreground applications.

## 3 Proposed Algorithm

In this paper, we propose a novel buffer cache algorithm that exploits the characteristics of mobile devices based on the hybrid memory architecture consisting of DRAM and PCM. We are focused on managing the buffer according to the states of applications. The proposed algorithm is based on traditional LRU policy, which is most popular and easy to understand in utilizing temporal locality. Foreground applications are allowed to run in DRAM whose access latency is faster than PCM. Therefore, write operation in PCM can be minimized and preserve lifecycle of PCM because most of I/O operations on mobile device are generated by foreground applications. On the other hand, background applications run in PCM preferentially whose access latency is slower than DRAM to guarantee the performance of foreground applications. Each page in the memory has a reference count which represents the number of times it is accessed, to determine migration between DRAM and PCM. Reference count is incremented when the page hit occurs and it is decremented when the page is selected as a candidate for eviction. If the reference count of a page on the memory space is zero, the page in DRAM moves to the MRU (Most Recently Used) position of PCM and the victim page in PCM is evicted to secondary storage.

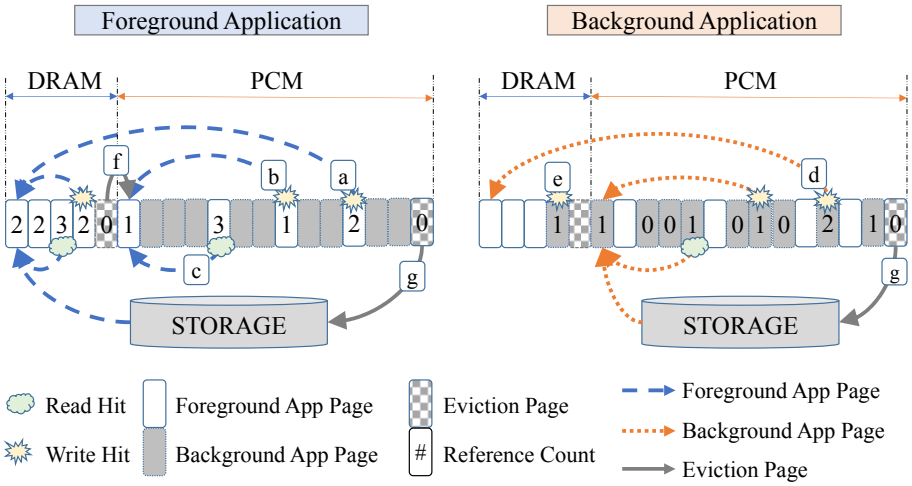


**Fig. 1.** Examples of page migration and eviction

Fig. 1 illustrates examples of page migration and eviction of the proposed algorithm. If the page, which is referred by foreground application, is not on the memory (page miss), it allocates a new page on the MRU position in DRAM. In case of page miss by background application, it allocates a new page on MRU position in PCM. When background application is switched to foreground state, only the page, which is being referred by the foreground application, is migrated to the MRU position in DRAM to minimize migration overhead. In contrast, when foreground application is changed to background state, the page in DRAM is not migrated to PCM and keeps in the same LRU position in DRAM. Because new foreground application generates lots of I/O requests and these pages extrude old pages to PCM, the pages of background applications are naturally migrated to PCM. Accordingly, there is no performance degradation by page migration.

In case of foreground application, new page is allocated on MRU position in DRAM and these pages are managed by LRU policy in DRAM. Cold pages that are not referred in DRAM will be migrated to PCM and they are managed by LRU policy in PCM. When page hit occurs in PCM, the page whose reference count is bigger than two is regarded as hot page and is migrated to the MRU position of DRAM (Fig. 1a). However, the pages whose reference count is lower than two are moved to the MRU position in PCM (Fig. 1b). This is for preventing performance degradation of foreground applications, which can be caused by operating hot pages in PCM which is slower than DRAM. Meanwhile, read-only pages in PCM, which are accessed by foreground application, are not migrated to DRAM but they are moved to the MRU position in PCM irrespective of its reference count (Fig. 1c). This is because read latency of PCM is not much slower than that of DRAM.

On the other hand, in case of background applications, new pages are allocated only on MRU position in PCM and these are managed by LRU policy to yield DRAM to foreground applications. When page hit occurs in PCM, every page owned by the background applications is moved to the MRU position in PCM because background

applications are not sensitive about the response time. However, pages referenced more than 2 times in PCM are migrated to the MRU position in DRAM (Fig. 1d), because these hot pages are operated frequently and it can cause performance degradation of the foreground application by occupying CPU and other hardware resources.

If hot pages owned by background applications are migrated to DRAM once, these pages are not moved to MRU position in DRAM again and hold its LRU position (Fig. 1e). This is to avoid performance degradation of the foreground applications by letting background applications occupy DRAM. If there is no empty space to allocate a new page in DRAM, the proposed algorithm migrates the page on the LRU position in DRAM to the MRU position in PCM (Fig. 1f). Also, if there is no free space for a new page in PCM, the page which is on the LRU position in PCM will be evicted to secondary storage (Fig. 1g).

## 4 Evaluation Results

In this section, we present the performance evaluation results and analyze the proposed buffer cache algorithm, comparing it with most popular buffer cache algorithm such as LRU and 2QLRU. To extract trace data for measuring performance of the proposed algorithm, we used Google Nexus7 that are using Android kitkat version 4.2.2 based on Linux kernel version 3.4.0. Also, we modified Android kernel to get the trace data while the applications make read/write requests into the buffer cache. Kernel has oom_adj value representing process priority which can be used to classify application state as foreground or background. Mobile devices check the oom_adj value to choose processes to kill when there is not enough memory space to run other processes. Normally, foreground application has zero value and background processes have values from 1 to 15.

Above all, we measure the footprint of each workload to set up experimental environment. For first trace, we collected trace data using Chrome web browser as a foreground application and using Hangout, Gmail, and mp3 player as background applications. The I/O request ratio of foreground and background application is 15:1. Also, we obtained second trace by using multiple applications changing each applications state between foreground and background. The request ratio of the application types for second trace is 3:1. During experiments, we configured the percentage of DRAM to 5% of total memory size to minimize power consumption at memory system and the rest percentage of memory to PCM. Our experimental results for the two types of traces are demonstrated in Fig. 2, Fig. 3, and Fig. 4. In the graphs, x-axis means memory size in each experiment and it is represented by the percentage of memory size to the total footprint. In each figure, graph (a) represents experimental results for the first trace and graph (b) represents experimental results for the second trace.

The hit ratio of our algorithm for two cases (all applications and foreground applications), our algorithm shows better performance results compared to LRU and 2QLRU policy in all range of memory sizes (Fig. 2, Fig. 3). Also, we can see that the

results for our algorithm are saturated earlier than the algorithms in the comparison group.

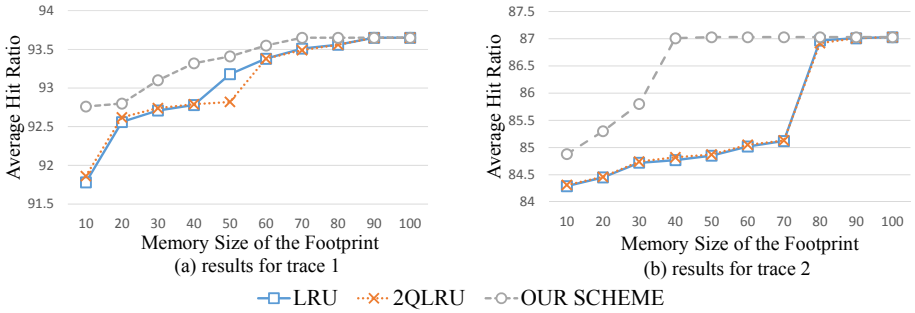


**Fig. 2.** Average hit ratio for all applications

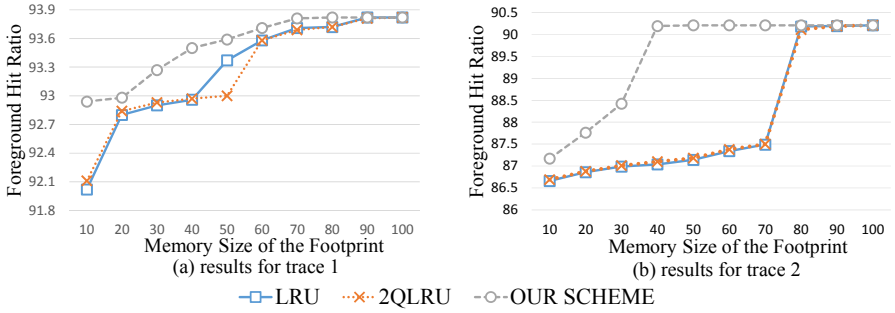


**Fig. 3.** Hit ratio for Foreground Application

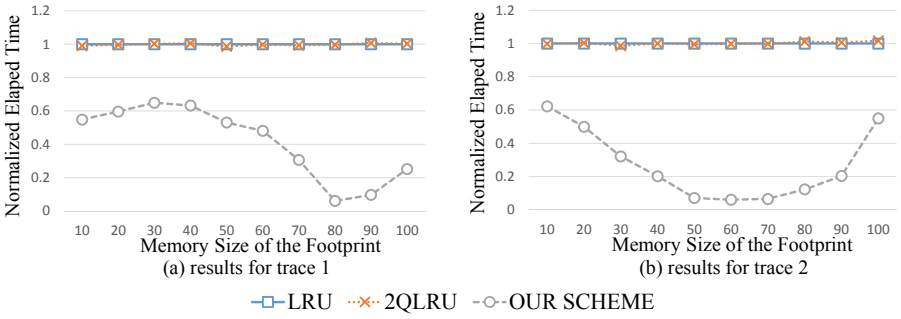


**Fig. 4.** Elapsed time for Foreground Application

Especially, Fig. 2b and Fig. 3b illustrate that hit ratio of the proposed algorithm is saturated definitely earlier than other algorithms in the range of over 40%. This means that the proposed algorithm can have better performance than other algorithms with small size of memory to run the same workload. Because every applications have been run in short interval in second workload, most of read and write requests are concentrated in small number of pages. However, experimental results for first trace

indicate that hit ratio increases gradually until memory size is 70% of the footprint (Fig. 2a, Fig. 3a). Nevertheless, the proposed algorithm has better performance than LRU and 2QLRU in whole range. In Fig. 3, hit ratio of the foreground application shows better performance than average hit ratio of all applications. Because foreground application has higher priority than background application in the proposed algorithm, many pages accessed by foreground application can be maintained in memory for a long time.

In order to confirm the performance of our algorithm, we calculated the elapsed time of each algorithm based on the metrics in Table 1. Fig. 4 clearly shows that our algorithm reduces the elapsed time of foreground application up to 53% compared with other algorithms. This is because our algorithm maintains the pages, which belong to foreground applications, on DRAM as long as possible by giving higher priority to foreground applications than background application. As a result, the hit ratio of foreground applications increases. On the other hand, the hit ratio of other algorithms decreases because they give same priority to all applications for considering the temporal locality. Especially, our algorithm shows the best performance in all cases.

Since energy consumption is one of the most important issues, we finally compared it with LRU and 2QLRU algorithm. As a result, we found that our algorithm significantly reduces the power consumption by 23% on average. This is because we can take the benefit of PCM (i.e., low power consumption) since our algorithm exploits the hybrid main memory architecture comprised of DRAM and PCM.

## 5 Conclusion

Generally, mobile devices employ DRAM as their main memory to mitigate performance gap between CPU and secondary storage. However, since DRAM continuously consumes the battery power of mobile device to keep data in DRAM, DRAM-based main memory architecture reveals a power consumption problem.

In this paper, we introduce a hybrid main memory architecture that is comprised of DRAM and PCM, and propose a novel buffer cache algorithm that saves battery power of device by exploiting non-volatility of PCM. In addition, our algorithm efficiently improves the performance of foreground applications because it gives higher priority to foreground applications than background applications. As a result, our algorithm shows high hit ratio by maintaining the pages, which belong to foreground application, as long as possible on DRAM. Our experimental results clearly show that our algorithm reduces the elapsed time of the foreground applications by 53% on average and the power consumption by 23% on average without any negative performance effects on background applications.

## Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-(H8501-15-1015)) supervised by the IITP(Institute for Information & communications Technology Promotion). Young Ik Eom is the corresponding author of this paper.

## References

1. Eilert, S., Leinwander, M., Crisenza, G.: Phase Change Memory: A New Memory Technology to Enable New Memory Usage Models. In: International Memory Workshop, pp. 1--2 (2009)
2. Qureshi, M.K., Srinivasan, V., Rivers, J.A.: Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In: International Symposium on Computer architecture, pp. 24--33. (2009)
3. Dhiman, G., Ayoub, R., Rosing, R.: PDRAM: A Hybrid PRAM and DRAM Main Memory System. In: Design Automation Conference, pp. 664--669 (2009)
4. Lee, S., Bahn, H., Noh, S.H.: CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures. In: IEEE Transactions on Computers, pp. 2187--2200 (2013)
5. Android Open Source Project, https://developer.android.com/guide/components/processes-and-threads.html
6. Johnson, T., Shasha, D.: 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In: 20th International Conference on Very Large Data Bases, pp. 439-450 (1994)
7. Carroll, A., Heiser, G.: An Analysis of Power Consumption in a Smartphone. In: USENIX Annual Technical Conference, pp. 1--14 (2010)
8. Datta, SK., Bonnet, C., Nikaein, N.: Android Power Management: Current and Future Trends. In: Enabling Technologies for Smartphone and Internet of Things, pp. 48--53 (2012)
9. Lim, G., Min, C., Kang, D.H., Eom, Y.I.: User-Aware Power Management for Mobile Devices. In: Global Conference on Consumer Electronics, pp. 151--152 (2013)
10. Han, S.J., Kang, D.H., Eom, Y.I.: Low Power Killer: Extending the Battery Lifespan by Reducing I/O on Mobile Devices. In: IEEE International Conference on Consumer Electronics, pp. 579--580 (2015)
11. Chu, S., Chen, S., Weng, S.F.: Design a Low-Power Scheduling Mechanism for a Multicore Android System. In: Parallel Architectures, Algorithms and Programming, pp. 25--30 (2012)
12. Gandhewar, N., Sheikh, R.: Google Android: An Emerging Software Platform for Mobile Devices. International Journal on Computer Science and Engineering, 1.1, pp. 12--17 (2010)
13. Android Open Source Project, https://source.android.com/devices/tech/power/index.html
14. Android Open Source Project, https://source.android.com/devices/#Binder IPC
15. Android Open Source Project, https://source.android.com/devices/tech/ram/low-ram.html