



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

M.S. THESIS

An Access-Pattern-Aware
Page Placement Algorithm for
the Hybrid Memory Architecture

하이브리드 메모리 구조에서
접근 패턴에 기반한 페이지 배치 알고리즘

BY

Youngdeok Seo

AUGUST 2017

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

M.S. THESIS

An Access-Pattern-Aware
Page Placement Algorithm for
the Hybrid Memory Architecture

하이브리드 메모리 구조에서
접근 패턴에 기반한 페이지 배치 알고리즘

BY

Youngdeok Seo

AUGUST 2017

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

An Access-Pattern-Aware
Page Placement Algorithm for
the Hybrid Memory Architecture

하이브리드 메모리 구조에서
접근 패턴에 기반한 페이지 배치 알고리즘

지도교수 엄 현 상

이 논문을 공학석사 학위논문으로 제출함

2017 년 6 월

서울대학교 대학원

컴퓨터 공학부

서 영 덕

서영덕의 공학석사 학위논문을 인준함

2017 년 7 월

위 원 장	_____	엄현영	(인)
부위원장	_____	엄현상	(인)
위 원	_____	장병탁	(인)

Abstract

Due to the high memory footprint pressure, the hybrid memory architecture consisting of small-sized DRAM and large-sized PCM has been regarded as a promising approach. This architecture aims at (1) alleviating the high power consumption caused by large-sized DRAM, and (2) harnessing the non-volatility and in-place update capability of PCM. Numerous studies have addressed the importance of page placement scheme between two different types of memory frames. Particularly, they have made every effort to provide good wear-leveling, hide the low write speed of PCM and reduce the power consumption. However, they lack one of the two points: (1) read-dominated workloads also decrease the system-wide energy efficiency and (2) excessive page migration should be avoided. In order to solve the abovementioned problems, we propose an access-pattern-aware page placement algorithm. Fundamentally, it uses page-leveling policy using multi-queue. To set the level of a page, it uses weighted access counting which puts a more emphasis on write accesses without ignoring read accesses. To minimize the number of migrations from DRAM to PCM, it performs state-transition-based recency checking for pages in DRAM. Our experimental results clearly demonstrate that it can reduce the average memory access time by up to 39% and the power consumption by up to 57%, respectively, compared to the previous approaches. Furthermore, they show that the PCM wear-out performance can be improved by 27%.

Keywords: NVRAM, Page placement algorithm, Hybrid memory architecture, PCM

Student Number: 2015-22901

Contents

Abstract	i
Chapter 1 Introduction	1
Chapter 2 Background and Motivation	6
2.1 Memory Architecture with DRAM and NVRAM	6
2.2 Motivating Examples	8
Chapter 3 Solution	12
3.1 Overview	12
3.2 Page Leveling	15
3.3 Getting a Free Page Frame from DRAM	18
3.4 Recency Checking	21
Chapter 4 Experimental Evaluation	26
4.1 Experimental Setup	26
4.2 Effect of Recency Checking	31
4.3 Effect of page placement algorithm depending on w -based access frequency	33
4.3.1 Effect on the number of page migrations	34
4.3.2 Effect on the average memory access time and the power consumption	35

4.3.3	Effect on the Lifetime of PCM	39
4.4	Performance analysis performed while varying the DRAM size . .	40
4.5	Techniques for further performance improvements	43
Chapter 5	Related Work	49
Chapter 6	Conclusion	52
	Bibliography	54
	초록	57

List of Tables

Table 2.1	Access speed and power consumption of DRAM and PCM	8
Table 4.1	Benchmark description	28
Table 4.2	System configurations for experiments	31
Table 4.3	Overall average normalized PCM write counts with dif- ferent values of w	40

List of Figures

Figure 2.1	Typical hybrid memory architecture with DRAM and PCM	7
Figure 2.2	Read/write access patterns for SPEC CPU2006 benchmarks: (a/d) mcf, (b/e) gcc and (c/f) namd	9
Figure 3.1	(a) Overall sequence of multi-level queue based page placement , (b) activity example of each LRU queue . .	14
Figure 3.2	Two state transition diagrams for a page in DRAM and they depict what the function <i>status_update(p)</i> called in Algorithm 3 does: (a) DRS (read status), (b) DWS (write status)	23
Figure 3.3	Example movement between PCM and DRAM depending on <i>DWS</i> and <i>DRS</i>	24
Figure 4.1	Overview of the experimental setup and the PIN-based trace-driven simulator	27
Figure 4.2	Evaluating the effectiveness of <i>Recency-Checking</i> : (a) number of page migration from PCM to DRAM, (b) number of page migration from DRAM to PCM, (c) average access time and (d) power consumption	32

Figure 4.3	Numbers of page migrations: (a) PCM to DRAM, (b) DRAM to PCM and (c) in both directions	36
Figure 4.4	Average memory access time: (a) read latency of PCM = $2\times$ read latency of DRAM, and (b) read latency of PCM = read latency of DRAM	38
Figure 4.5	Comparison of power consumption	38
Figure 4.6	Comparison in PCM write count	40
Figure 4.7	Average memory access time for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf	45
Figure 4.8	Power consumption for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf	46
Figure 4.9	PCM write count values for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf	47
Figure 4.10	Average memory access time with TA(Trace Analysis)	48
Figure 4.11	Comparison of power consumption with TA	48
Figure 4.12	Comparison in PCM write count with TA	48

Chapter 1

Introduction

Since the users of modern computing machines expect performance-demanding applications, embedded systems such as smartphones have started being armed with more powerful computing resources such as multi-cores and large-sized DRAM. Particularly, the emerging smartphone applications have been developed to require large memory footprint, *i.e.*, large high-density images are captured and highly accelerated GPU processing is performed. This leads to serious pressure on the capacity of memory. In addition, to improve the responsiveness of mobile devices, more data needs to be resident in memory rather than a storage device such as a flash memory. Therefore the manufactures of smartphones have adopted large-sized DRAM.

As DRAM requires periodic refreshment to retain the stored data, a significant portion of total energy is consumed in the memory subsystem. Thus it has been one of the critical issues in the design space of systems. This necessitates the development of a memory subsystem which takes into account both the memory capacity and energy consumption simultaneously. To this end, adopt-

ing a non-volatile memory has been investigated in designing the memory hierarchy of systems. Recently, phase change memory (PCM) has been the major breakthrough for non-volatile memories on account of its scalability and better energy performance [1, 2, 3, 12]. In addition, it is a byte-addressable memory, thus it can be accessed similar to DRAM and provides fast read and write to persistent storage data via memory-mapped access.

In spite of abovementioned advantages, applying PCM instead of DRAM in the memory hierarchy has two major limitations: (1) the write endurance is limited to $10^7 \sim 10^8$ [1, 3, 12] and (2) the access speed of write is much lower than that of DRAM. To overcome these drawbacks, small-sized DRAM is incorporated into the memory hierarchy along with PCM; DRAM can hide the slow write access speed and the frequent write access can be absorbed by DRAM, In addition, our algorithm uses page-leveling based on multi-queue. increasing the endurance of PCM.

The key issue in this architecture is the development of page placement scheme such that popular pages are located in DRAM while unpopular ones are in PCM, where a popular page means a frequently accessed page [3, 5, 7]. In an access-popularity-based page placement scheme, read and write count values are used to assess the degree of read- and write-popularity. If both values are not differentiated, *i.e.*, one read count is matched to one write count, read-intensive pages and write-intensive pages may have the same probability to be accessed in DRAM [5]. Under such a scheme, the total PCM write count value, memory access time and the system-wide power consumption are increased rather than a write-history-aware algorithm which keeps only write-intensive pages in DRAM [3].

The latency and power consumption for read access are lower than those of write access in PCM. However they are around twice larger than those of DRAM

[1, 3]. Thus, memory access time and power consumption can be increased when all read-intensive pages are kept in PCM rather than DRAM. Thus, for read-intensive workloads, [5] might have high probability to denote the smaller total memory access time and less power consumption than [3].

Due to the use of DRAM is much smaller than the PCM part in the hybrid memory architecture, the DRAM one is readily saturated with currently used pages. Thus, the pages which are unlikely to be accessed again in the DRAM part are moved to the PCM [3, 5]. In other words, to select victim pages in the DRAM part, the page placement scheme should take into account the access recency as well as access popularity of pages [3, 5].

Importantly, the write endurance for PCM is limited, thus the page placement scheme should be equipped with a policy which maintains PCM write count as small as possible. There are two cases which increase the PCM write count value: 1) a write operation that occur on a page in PCM and 2) moving a page from DRAM to PCM. As the running time of a system increases DRAM is occupied by popular pages and no empty memory frame is left. In such a case, to avoid writing a page on PCM, if the page placement scheme tries to occur every write operation on DRAM only, a write access on PCM accompanies a minor fault [3] which evicts an unpopular page from DRAM and migrates the page to PCM in order to host the write operation in DRAM [3]. Resultantly, every minor fault brings about the increment of the PCM write count value [3]. Thus, the page placement scheme should take care of not only the PCM write operation but also the number of migrations between two distinct types of memory frames.

Taking all the aforementioned issues into account, we propose a light-weight software-managed solution approach in the hybrid memory architecture consisting of DRAM and PCM, called an access-pattern-aware page placement algo-

rithm. It makes use of access recency as well as access popularity in estimating the future access of a page [13]. It takes the baseline notion of previous works in that pages are efficiently ranked according to their popularity (*i.e.*, access frequency) and top-ranked pages in PCM are moved to DRAM [5, 7].

In this paper, we demonstrate the following contributions:

- (i) The latency and power consumption brought by a read operation in PCM are larger than those in DRAM. Therefore, if every write operation is performed on DRAM only, read-intensive workloads can be negatively affected in terms of access speed and energy performance. Based on this observation, we newly devised a biased page access counting method which puts a more weight onto write accesses than read ones, and then the access popularity of a page is computed by adding read counts and weighted write counts. We applied it in grouping pages according to the access popularities.
- (ii) Different from the previous works [5, 7], the access recency of a page is determined by the state transition of the page. In order to quantitatively measure the access recency of a page in DRAM, we define two light-weight transition diagrams of the page: a write-state one and a read-state one. The state represented in each transition diagram is changed by periodically checking the history of write and read accesses. Note that, the target pages for the recency checking routine are confined to the ones in the lowest-ranked group in DRAM, thus the incurred overhead by the routine is quite small and the access frequency is also reflected by default in the routine since the pages in the group have the least access popularity.
- (iii) Our approach basically ranks pages according to their access frequency, and only top-ranked pages are migrated from PCM to DRAM. Thus, it

can lead to a smaller number of page migrations between two distinct types of memory frames than the preceding approaches [3, 6], thereby reducing the number of PCM write count and the memory access time.

- (iv) We propose a pure software solution without any additional hardware or architectural support.

Particularly, to demonstrate the effectiveness of our algorithm, we performed a series of experiments using SPEC CPU2006 benchmark programs. The experimental results show that our proposed approach can reduce the average memory access time by 39% and can increase power savings by up to 57% compared to the preceding ones. The remainder of this paper is structured as follows: Section 2 gives an explanation of the background and motivating examples. Section 3 presents the detailed solution approach we took along with the devised algorithms. Section 4 describes the experimental evaluation and provides a brief analysis of the results. The prior works on the memory architecture consisting of DRAM and PCM are discussed in Section 5, and finally we conclude this paper in Section 6.

Chapter 2

Background and Motivation

The proposed solution is designed on top of the hybrid memory architecture. To aid in understanding throughout the rest of this paper, we provide a brief introduction of the target memory architecture and then discuss the issues which are addressed in our research.

2.1 Memory Architecture with DRAM and NVRAM

As the candidates for the non-volatile memory, new memory technologies such as Magnetic RAM (MRAM), Ferroelectric RAM (FRAM) and Phase-change memory (PCM) have been introduced. Among these technologies, PCM has been regarded as the most favored memory technology in terms of performance per memory density [1, 2]. Since PCM is byte-addressable and provides in-place update as DRAM [3], the persistent data can be read or written in a memory-mapped manner.

The memory architecture composed of DRAM and NVRAM can be divided

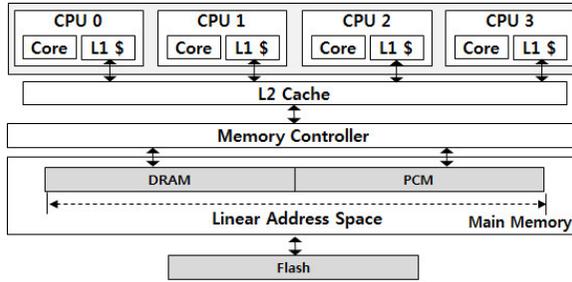


Figure 2.1 Typical hybrid memory architecture with DRAM and PCM

into two categories: the DRAM cache architecture [2, 8, 9] and the hybrid memory architecture [3, 5, 6]. In the DRAM cache architecture, DRAM is used as a cache memory for PCM, where only PCM is used as the main memory. Thus, the access to DRAM is hidden in the operating system (OS), and only hardware can assign pages to DRAM and maintains the data consistency between PCM and DRAM. This architecture cannot provide the full span of memory space which both PCM and DRAM can support.

In contrast, in the hybrid memory architecture, both PCM and DRAM have the same level in the memory hierarchy. Thus, the memory space in PCM and DRAM can be fully accessed by OS. In this paper, we focus only on the hybrid memory architecture, and Figure 2.1 shows a typical memory hierarchy in the architecture. In this architecture, both PCM and DRAM are in the same linear address space, and thus the OS can directly access data in both PCM and DRAM using a conventional memory controller [2].

Table 2.1 shows the result of comparing the characteristics of DRAM and PCM. Unlike DRAM, PCM has an asymmetric latency between a write access and a read access. The active power consumed during the read or write accesses in PCM is larger than that of DRAM. In contrast, the static (or idle) power consumption in PCM is smaller than that of DRAM [12]. To represent logic

zero and one, a PCM cell is configured to crystalline and amorphous phases. Through a heating process, two distinct phases can be changed to each other [1]. As shown in the table, due to the heating process, PCM has two major weaknesses: (1) the values of latency in PCM for read and write accesses are twice and seven times larger than those of DRAM, respectively, and (2) the maximum number of write operations is limited [1, 2, 5].

The purpose of taking our proposed approach is to alleviate abovementioned limitations while retaining the strength such as byte-addressability and linear address region throughout the entire space of DRAM and PCM. Thus, we focus on hiding the higher access latency in PCM and reducing the larger active power consumption in PCM than that of DRAM, by using an efficient page placement policy in the hybrid memory architecture.

Throughout the rest of this paper, we differentiate between page frames and pages. A page frame means a 4KB-sized physical memory frame and a page is a virtual memory space which is mapped to the page frame.

2.2 Motivating Examples

The energy management policy is the first issue addressed in our research in the hybrid memory architecture. Figure 2.2 depicts the read and write access

Table 2.1 Access speed and power consumption of DRAM and PCM

	DRAM	PCM
Non-volatility	No	Yes
Read/Write Latency	50/50 (ns)	50 or 100/350 (ns)
Read/Write Power	0.1/0.1 (nJ/bit)	0.2/1.0 (nJ/bit)
Static Power	1(W/GB)	0.1(W/GB)
Endurance	NA	10^7

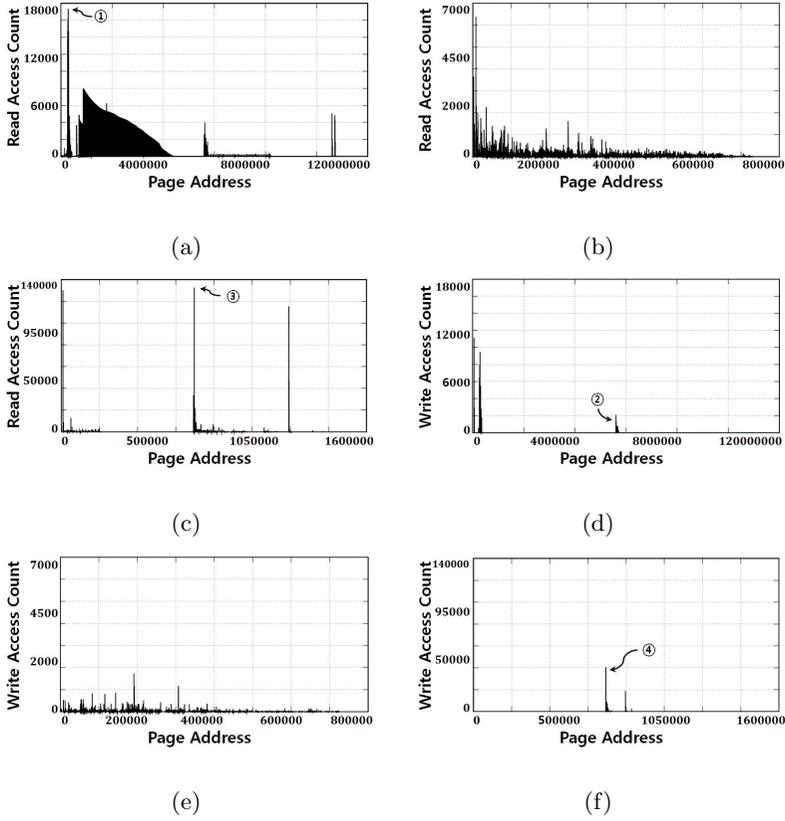


Figure 2.2 Read/write access patterns for SPEC CPU2006 benchmarks: (a/d) *mcf*, (b/e) *gcc* and (c/f) *namd*

patterns for benchmark programs of SPEC CPU2006 [15, 16]. Figure 2.2(a) and Figure 2.2(d) are for *mcf*, Figure 2.2(b) and Figure 2.2(e) are the result of *gcc*, and Figure 2.2(c) and Figure 2.2(f) show the patterns of *namd*. The *x*-axis and *y*-axis indicate the page address and accumulated access count, respectively. Suppose all the pages are in PCM.

For example, a page p_2 with page address 7500000 has 2000 write access count as pointed by ② in Figure 2.2(d) and another page p_1 with page address 200000 has read count value 17000 pointed by ① in Figure 2.2(a). If we compare

the power consumption of these two pages by using the power unit shown in Table 2.1, p_2 consumes 2000 nJ/bit, and p_1 , 3400 nJ/bit. The read-intensive page p_1 consumes 1.7 times more energy than p_2 . Through this example, we can know that migrating frequently written pages to DRAM while leaving read-dominated pages in PCM can deteriorate the energy performance.

Second, the page access frequency and recency should be taken into account simultaneously in page placement. Figure 2.2(b) and Figure 2.2(e) show the read and write access pattern of *gcc*. This benchmark program uniformly accesses the entire page address space while *namd* utilizes specific pages for read and write accesses as shown in Figure 2.2(c) and Figure 2.2(f), respectively. Consider a page p of *namd* whose page address is 775000. The read and write access frequencies (③ in Figure 2.2(c) and ④ Figure 2.2(f)) of the page p are very high and the page should be kept in DRAM to avoid being referenced in the slower PCM. At the same time, we can expect the reduced power consumption. Suppose that p experiences a large number of accesses in a short period of time, and then referenced again quite a long time later. If there are lots of pages similar to p in a system and only the access frequency is considered in the page placement policy, the pages whose access frequencies are lower than that of p while being evenly referenced throughout their lifetimes, cannot be accessed in DRAM.

We consider that page migration between two different types of memory frames is expensive. For ease of explanation, let us define p as a virtual page address, and pf^p and pf^d are physical page frames of PCM and DRAM, respectively. If p is mapped to pf^d currently and going to be migrated to PCM, p should be unmapped from pf^d first and it accompanies page table entry (PTE) modification for p . Then the content of pf^d should be copied to pf^p and p is remapped to pf^p by modifying the PTE of the process which owns p . This

procedure unavoidably brings about system latency and it directly affects the user experience holding the target embedded system.

In this paper, to overcome the problems explained in abovementioned motivating examples, we propose a page placement algorithm in the hybrid memory system.

Chapter 3

Solution

In this section, we first overview our target system model. We then technically explain the details of the proposed page placement algorithm for the hybrid memory architecture. The algorithm consists of (1) page leveling, (2) getting a free page frame from DRAM and (3) recency checking, and we explain them in order.

3.1 Overview

The key idea behind our solution approach is (1) leveling pages according to their access frequencies, (2) migrating popular pages in PCM to DRAM and moving pages not recently accessed in DRAM to PCM, and (3) picking a page out of DRAM via a recency evaluation mechanism when DRAM has no free page to accommodate a popular page from PCM.

To quantitatively represent the access frequency while taking into account both the read and write accesses, we first introduce the notion of weighted

access count of a page p given as follows.

$$N_{acc}(p) = \frac{read_count(p)}{w} + write_count(p), \quad (3.1)$$

where w is the weight value defined as a system-wide constant,

and $read_count(p)$ and $write_count(p)$ are read and write count values for page p , respectively. Since for PCM, the latency of a write access is much higher than that of a read access and a write access consumes much more energy than that for DRAM as described in the Table 2.1, we put a more emphasis on write accesses than read accesses. Note that we define the weight value w as the number of read accesses which is regarded as the single write access. In this paper, $N_{acc}(p)$ denotes the access frequency of a page p and the access recency of the page is assessed by monitoring whether the page has experienced write or read accesses for a designated period of time or not. When the target system starts, $\forall p \in P_{all}$, $N_{acc}(p) = 0$, where P_{all} is the set consists of all the pages used in the target system.

Figure 3.1(a) illustrates the overall sequence of the proposed solution. Our solution utilizes multi-level queues [5, 7]. Each queue is managed in an LRU (Least Recently Used) manner [5, 7], and it is called an LRU queue in this paper. All the data structures of all LRU queues in the system are stored in DRAM for implementation simplicity and faster handling. Each LRU queue has a linked list, and each list has the descriptors for its pages. A page descriptor consists of read and write access counts to compute $N_{acc}(p)$ of the page p and its physical page frame number (PFN).

The page descriptor for each page assigned in PCM is linked in one of the LRU queues, $Q_0^p, Q_1^p, Q_2^p, \dots, Q_{n-1}^p$, where n is the number of LRU queues for PCM. Similarly, DRAM has its own LRU queues, $Q_n^d, Q_{n+1}^d, Q_{n+2}^d, \dots, Q_{m-1}^d$, according to their $N_{acc}(p)$ values, where m is the number of total LRU queues

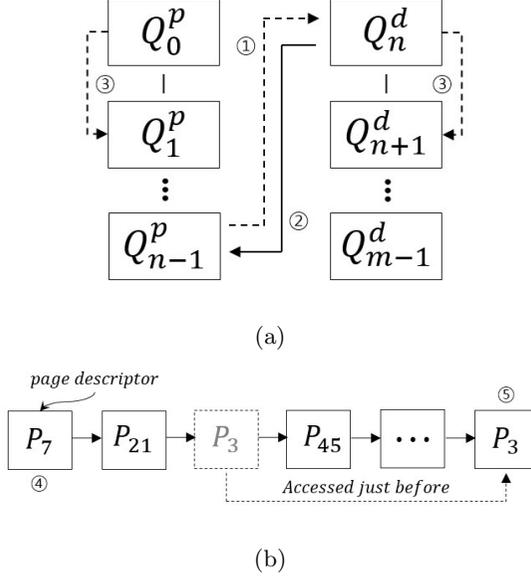


Figure 3.1 (a) Overall sequence of multi-level queue based page placement , (b) activity example of each LRU queue

in the system. Thus, a page in the system can be ranked in one of the queue levels from 0 to $m - 1$.

As for page migrations between DRAM and PCM, there are two cases. In the first case, a page linked in Q_{n-1}^p is moved to Q_n^d when its $N_{acc}(p)$ -based queue level reaches a pre-defined threshold value n (① in Figure 3.1(a)). In the second one, a page in Q_n^d is selected as a victim page and then the page is moved to Q_{n-1}^p when the page has not been accessed recently (② in Figure 3.1(a)). To check the recency of a page, we use state transition diagrams for write and read references and their states are updated periodically according to the read and write accesses. Note that if Q_i^d is empty, Q_{i+1}^d is used for selecting a victim page to be transmitted to PCM, where $i = n, \dots, m - 2$.

Figure 3.1(b) shows the internal activities performed in each LRU queue, and the activities are valid only for the pages which have the same queue level.

The descriptor of a page p is located in the MRU (Most Recently Used) position depicted as ⑤ whenever its $N_{acc}(p)$ increases. The LRU position marked by ④ is allocated for a page p whose $N_{acc}(p)$ value has not been updated recently. The following subsections explain the detailed algorithms of our proposed solution.

3.2 Page Leveling

In most operating systems supporting a virtual memory environment, a virtual page address is mapped to a physical page frame by establishing the page table. Each page table entry (PTE) of a page has one accessed bit and one dirty bit. The accessed bit is automatically set by the CPU each time the page experiences either write or read access. The OS checks regularly the bit to know how frequently the page is used. The dirty bit indicates whether the contents of the page have been modified (*i.e.*, a write operation). Using these two bits, the OS can figure out whether the current access type for the page is either a read operation or a write one at once.

Algorithm 1 shows the pseudo code for the *Page_Leveling* algorithm and this algorithm is performed on every access to a page. At the beginning of the boot process, m and n values are delivered as the system-wide constant parameters. Then, $N_{acc}(p)$, $read_count(p)$, $write_count(p)$ and l values for every page p in the system are set to the initial value zero. Thus, when a page p is accessed for the first time, this page is mapped to one of the physical page frames in PCM and its page descriptor is enqueued to Q_0^p . Whenever a page p is accessed, the operation type op is delivered to Algorithm 1 as an input by the OS. Depending on op and by Equation 3.1, the $N_{acc}(p)$ value of the page p is increased as described in Lines 5~14.

As the value $N_{acc}(p)$ increases, it is necessary to move the page descriptor

Algorithm 1: Page_Leveling

Input: p (referenced page), op (read or write)

Data: m and n (the number of total LRU queues in the system and that of PCM), $read_count(p)$ and $write_count(p)$ (read and write count values for page p) l (current queue level of page p)

```
1 if ( $p \in \text{DRAM}$  or  $p \in \text{PCM}$ ):
2   if ( $N_{acc}(p) \geq 2^m - 1$ ):
3     return
4   else:
5     if ( $op == \text{read}$ ):
6       if ( $read\_count(p) < w$ ):
7          $read\_count(p) = read\_count(p) + 1$ 
8       else:
9          $read\_count(p) = 0$ 
10         $N_{acc}(p) = N_{acc}(p) + 1$ 
11      else:
12         $N_{acc}(p) = N_{acc}(p) + 1$ 
13       $l = \lfloor \log_2 N_{acc}(p) \rfloor$ 
14       $dequeue(p)$ 
15      if ( $p \in \text{PCM}$ ):
16        if ( $l \geq n$ ):
17           $pf^d = \text{Get\_Page\_From\_Dram}()$ 
18          mapping  $p$  to  $pf^d$  and  $enqueue(p, Q_n^d)$ 
19        else:
20           $enqueue(p, Q_l^p)$ 
21      else:
22         $enqueue(p, Q_l^d)$ 
```

of the page p to a higher level LRU queue. In this case, we should determine a suitable destination LRU queue Q_l^p of p according to its $N_{acc}(p)$ value, where $0 \leq l \leq n - 1$. To do this, in Line 16, we define the queue level l of a page p as below [5, 7].

$$l = \lfloor \log_2 N_{acc}(p) \rfloor \quad (3.2)$$

This function is identically used in determining queue level of a page in DRAM. The range of $N_{acc}(p)$ for a page p in Q_l^p is $2^l \leq N_{acc}(p) < 2^{l+1}$. When $N_{acc}(p)$ reaches 2^{l+1} , the page descriptor is moved to Q_{l+1}^p and we call it as *promotion* as shown ③ in Figure 3.1(a). If the obtained value l for a page p in PCM (Line 16) is changed from the previous value, the page is linked to a new LRU queue as in Line 23 (if the page is in DRAM, Line 26 is used instead). Since m is the total number of LRU queues and n is the number of LRU queues in PCM, $m - n$ LRU queues are in DRAM. Both Q_{m-1}^d and Q_{n-1}^p hold page descriptors which are most frequently accessed in DRAM and PCM, respectively. When the queue level of a page p in Q_{n-1}^p reaches a value greater than $n - 1$, the page is moved to DRAM (Line 21). For this movement, through the function *Get_Page_From_Dram()*, a free page frame pf^d is acquired. In this way, a free space in DRAM for the newly incoming page from PCM is provided.

If a page p is in DRAM, its queue level is simply updated (Lines 26). In this case, the maximum value $N_{acc}(p)$ can reach is confined to $2^m - 1$ as in Lines 2~3. Thus, the largest queue level l remains $m - 1$.

In the algorithm, through a function *enqueue*(p, Q_l^p), the algorithm links the descriptor of a page p to the most recently used (MRU) position of Q_l^p . Conversely, a function *dequeue*(p) unlinks a page p from its LRU queue.

3.3 Getting a Free Page Frame from DRAM

As stated in Line 21 of Algorithm 1, a page p in PCM is migrated to DRAM when its queue level is no less than the threshold value n . At this point in time, *Get_Page_From_Dram()* is executed to return a free page of DRAM, and Algorithm 2 describes the detailed activities of it. When the algorithm is executed and DRAM has a free page to return, the proposed scheme can obtain a free page immediately (Lines 1~2).

However, in the hybrid memory architecture, the DRAM part is much smaller than the PCM one, and thus the DRAM one is apt to be fully occupied by the pages of running processes. Therefore, the algorithm attempts to select a victim page which currently occupies one of the page frames in the DRAM part as stated in Lines 4~11.

In selecting a victim page, the algorithm first find the lowest level queue among the non-empty queues in DRAM to reflect the access frequency. Then, among the pages in the discovered queue, the algorithm selects a page which has the lowest access recency. Our proposed approach utilizes *victim_flag* to identify the degree of recency of a page in DRAM. As shown in Line 6 of the algorithm, a page p is selected as the victim page if its *victim_flag* is set to *TRUE*. Note that there can exist multiple pages whose *victim_flags* are *TRUE* but the algorithm selects only the first found page. The *victim_flag* is updated by using the *Recency_Checking* algorithm and its detailed explanation is made in the following subsection.

In Line 13, we can see the function *Get_Page_From_PCM()* and this function is called only in two cases. In the first case, a page is created and accessed for the first time by a process. The second one is shown in Line 13 of the algorithm. When a page p is selected as a victim, the virtual address of p is unmapped from

Algorithm 2: *Get_Page_From_Dram*

Output: pf^d (a page frame in DRAM)

```
1 if (a free  $pf^d$  is in DRAM):
2   |   return  $pf^d$ 
3 else:
4   |    $l$  = the lowest level of the queue which is not empty in DRAM
5   |   for each  $p \in Q_l^d$ :
6   |       |   if (victim_flag of  $p$  == TRUE):
7   |           |   dequeue( $p$ )
8   |           |   break
9   |   unmapping  $p$  from  $pf^d$ 
10   $pf^p$  = Get_Page_From_PCM()
11  mapping  $p$  to  $pf^p$ 
12  enqueue( $p$ ,  $Q_{n-1}^p$ )
13  return  $pf^d$ 
```

its physical page frame pf^d in DRAM (Line 12). After a free page frame pf^p is obtained through *Get_Page_From_PCM*(), p is mapped to pf^p . In the middle of this procedure, the data originally stored in pf^d is copied to pf^p . We do not show the pseudo code of *Get_Page_From_PCM*() here because its procedure is basically similar to *Get_Page_From_Dram*() excluding the migration part (Lines 13~15). When it searches for a free page frame, it first looks up for a free page frame which is not mapped to the virtual address. If there is no free page frame, it performs the same procedure as described in Line 4 of Algorithm 2. However, *Get_Page_From_PCM*() does not use such a complicated algorithm to assess the recency of a page in selecting a victim page as that of *Get_Page_From_Dram*(). Instead, using the *dequeue*() function, the head of the lowest level queue in

Algorithm 3: *Recency_Checking*

Data: *DRS* (read status of a page in DRAM), *DWS* (write status of a page in DRAM) *dr_access_bit* (DRAM read access bit), *dw_access_bit* (DRAM write access bit)

```
1  $l =$  the lowest level of the queue which is not empty in DRAM
2 for each  $p \in Q_l^d$ :
3    $status\_update(p)$ 
4   if ( $DRS$  of  $p + DWS$  of  $p \leq 1$ ):
5      $victim\_flag$  of  $p = \text{TRUE}$ 
6   if ( $DRS$  of  $p == 2$  or  $DWS$  of  $p > 0$ ):
7      $victim\_flag$  of  $p = \text{FALSE}$ 
8    $dr\_access\_bit$  of  $p = 0$ 
9    $dw\_access\_bit$  of  $p = 0$ 
```

PCM is returned. The reason for this is that a recently accessed page is located in the tail of its LRU queue while the page not accessed recently is in the head.

In Line 21 of Algorithm 1, the page migration from PCM to DRAM is described. During this migration, the $N_{acc}(p)$ value of a page p is not initialized to zero. Instead, this value is maintained even after the page is migrated to DRAM. Thus, the lowest level LRU queue in DRAM is Q_n^d . The procedure from Line 12 to Line 15 of Algorithm 2 represents the page migration from DRAM to PCM. In this case, the migrated page p is enqueued to Q_{n-1}^p and its $N_{acc}(p)$ value is adjusted to 2^{n-1} . Since the page p migrated from DRAM denotes that it has been accessed frequently in PCM before being moved to DRAM, there is a high probability that this page is accessed frequently again. Thus, we enqueue the page p to Q_{n-1}^p .

3.4 Recency Checking

In the hybrid memory architecture, the DRAM size is smaller than that of PCM, and thus the DRAM is quickly filled with the pages of currently running processes and no free page frames are left. Therefore, in this architecture, we need to establish a rule for managing two different types of memory frames to determine which pages should be retained in the DRAM or which pages should be kept in PCM. This necessitates developing an efficient page migration policy between two distinct types of memory frames.

Basically, the proposed solution holds the pages with small $N_{acc}(p)$ values in PCM while the pages with large ones are located in DRAM. Thus, our approach performs page migration of page p from PCM to DRAM based on the $N_{acc}(p)$ value. Different from this, in the case of migrations from DRAM to PCM, priority is given to the recency rather than the frequency of a page in DRAM, in selecting a candidate page to be migrated to PCM. As a metric of the recency of a page in DRAM, we use *victim_flag* of a page as stated in Algorithm 2. This metric is measured per page and its value is updated periodically.

If a page p , with a large $N_{acc}(p)$ value and low recency, resides in DRAM and even for a long time it is not accessed, a page q , with high recency and a $N_{acc}(q)$ value smaller than the $N_{acc}(p)$ one can lose a chance to be accessed in DRAM. Consequently, the system-wide latency can be increased. To avoid this, we propose *Recency_Checking* algorithm and its pseudo code is shown in Algorithm 3. Fundamentally, the algorithm aims at marking the least recently accessed pages among the least popular pages in DRAM by updating their *victim_flags*.

Algorithm 3 starts with finding the LRU queue at the lowest queue level in DRAM (Line 1) and it enables that finding the victim candidate pages are

confined to the least popular pages in DRAM. Then the algorithm updates the states of the pages in the LRU queue (Line 3). In order to represent the state of a page, we define two state transition diagrams. One represents the state transitions of the write status (DWS), and the other, those for the read status (DRS) of a page. Figure 3.2 shows the state transitions according to the write and read accesses of a page, and depicts the activities of the function *status_update()* in Line 3 of Algorithm 3.

During a period of time, if the write accesses are recorded more than once for a page, the *dw_access_bit* of the page descriptor is set to one, and if not, it remains as zero. The same update policy of *dw_access_bit* is applied to *dr_access_bit*. The write-status has three states, $DWS \in \{0, 3, 4\}$ and the read-status has also three states, $DRS \in \{0, 1, 2\}$. If the *dw_access_bit* is one, the write-status can be increased to state 3 or state 4 while being decreased to state 3 or state 0, if *dw_access_bit* is zero in a certain period. If the *dr_access_bit* is one, the read status can be increased to state 1 or state 2 while being decreased to state 1 or state 0, if *dr_access_bit* is zero in a period.

After updating DRS and DWS of a page, the algorithm decides the value of *victim_flag* of the target page. To decide this value, the algorithm first checks the DRS and DWS values (Lines 4~9), and each of them represents one of the states of read and write status, respectively. Because the DWS value cannot be smaller than 3 if at least one write access has happened recently, and the DRS one cannot be smaller than 2 if the *dr_access_bit* value is set to one for the recent two consecutive periods, if the if-condition in Line 4 is true, this means that there has been no write access and no read-intensive access for the target page during the last two periods. In this case, the algorithm defines the page as the one with low recency and its *victim_flag* is set to *TRUE* (Lines 4~6). By contrast, if the page experiences two consecutive read accesses or at least

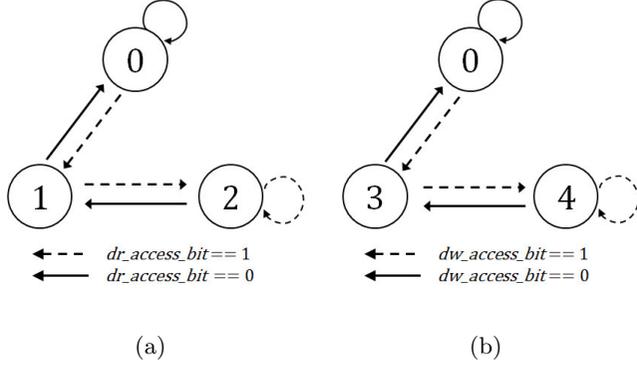


Figure 3.2 Two state transition diagrams for a page in DRAM and they depict what the function $status_update(p)$ called in Algorithm 3 does: (a) DRS (read status), (b) DWS (write status)

one write access during the recent two periods, the algorithm determines that the page has a high access recency by marking its $victim_flag$ as $FALSE$ (Lines 7~9).

Both dr_access_bit and dw_access_bit just record only one period data for read and write accesses of a page, so they are cleared to zero at every end point of the period to obtain new dr_access_bit and dw_access_bit values during the next period (Lines 10~11).

Figure 3.3 illustrates how the state transitions and page migrations might occur with time. For example, there are five pages p_1, p_2, p_3, p_4 and p_5 in the target system. We represent the state of each page as DWS/DRS and Q level means the level of the LRU queue for a page. Starting at t_0 , all the DWS/DRS values of the pages are 0/0 and their initial locations are all at PCM. Before t_0 (and t_1), only p_3, p_4 and p_5 have been moved to DRAM since their Q level has been no less than the threshold value for the migration. Even though they are in DRAM until t_1 , their DWS/DRS values still remain 0/0 since $Recency_Checking$ has not started yet.

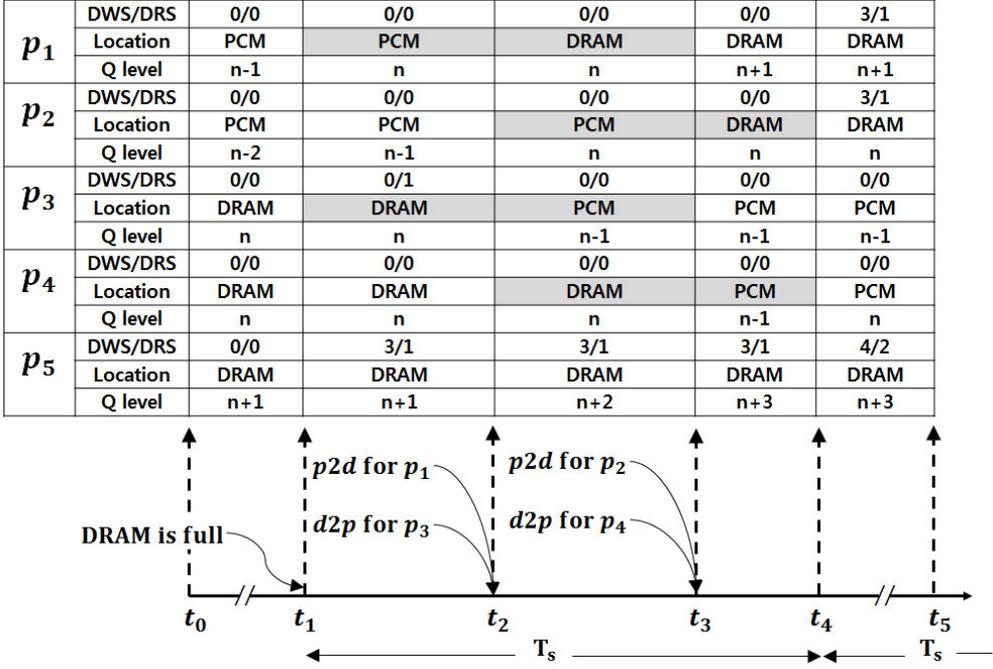


Figure 3.3 Example movement between PCM and DRAM depending on *DWS* and *DRS*

At t_1 , DRAM is saturated, so the *Recency-Checking* algorithm performs its operation shortly to find out the victim page to move to PCM and its period is denoted by T_s in the figure. At t_2 , the *Q level* of p_1 reaches the threshold n where n is the number of LRU queues in PCM. The *Recency-Checking* algorithm has found the first victim page p_3 at t_1 based on the condition in Line 4 of Algorithm 3. Thus at t_2 , p_1 and p_3 exchange their locations. This is because the required amount of time has not elapsed for *Recency-Checking*. As p_1 , the *Q level* of p_2 has reached the threshold value, the page is migrated to DRAM at t_3 . Accordingly, the victim page p_4 is moved to PCM. Despite p_1 and p_5 are allocated in DRAM between t_2 and t_4 , their states are not changed.

At t_4 , DRAM still has no free page, and thus the *Recency_Checking* algorithm starts again. According to the result of the algorithm, the state of each page is changed. At this point of time, the *Q level* of p_4 is adjusted to $n - 1$, and until t_5 its *Q level* has increased to n on account of the increased $N_{acc}(p_4)$. Thus, at t_5 , p_4 is supposed to return to DRAM. Since the *Q level* of p_5 has increased monotonously, no page migration is needed for the page.

Chapter 4

Experimental Evaluation

To evaluate the effectiveness of the proposed page placement algorithm, this section reports on the experiments we conducted. We shortly describe the experimental setup including the configurations for the target system. We then show the results of experiments and also give brief analyses of them.

4.1 Experimental Setup

Figure 4.1 shows the overall experimental environment. For further and accurate observation of the access patterns in which benchmark programs read and write memory, we used Intel’s PIN tool. It is a dynamic binary instrumentation framework and performs instrumentation on the compiled binary files at runtime [2, 14].

To realistically demonstrate the usefulness of our proposed approach, we used several benchmark test suites from SPEC CPU2006 [15, 16], and Table 4.1 gives a brief description of the suites. We also measured the total write and read

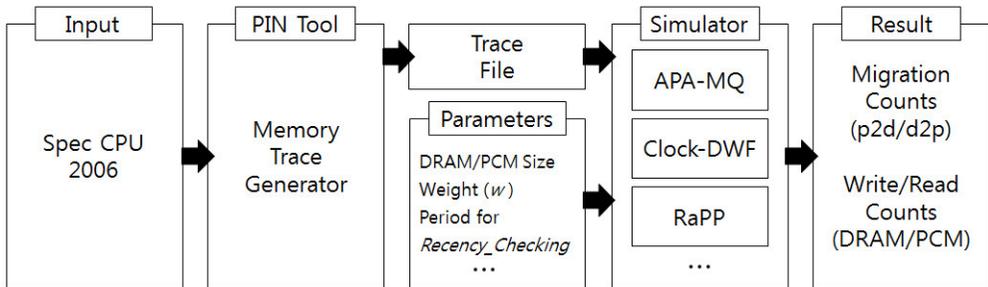


Figure 4.1 Overview of the experimental setup and the PIN-based trace-driven simulator

access counts of each program offline, and figured out the ratio between them as shown in the table.

By compiling each benchmark program with the PIN tool, we could obtain a trace file of each workload. The trace file is used as the input of *Page Placement Simulator*. The actual implementation of page placement algorithm is embedded in the *Page Placement Simulator*.

The simulator also receives some configuration parameters required for adjusting the PCM and DRAM sizes, determining the time interval T_s between recency checking procedures, m the number of total LRU queues in the system, n the number of queues assigned to PCM and the value of weight w . When the simulator finishes its simulation, it generates the per-page results which are the number of page migrations from PCM to DRAM and that in the reverse direction, and the aggregated write/read access counts in PCM and DRAM, respectively.

In order to demonstrate that our design can lead to effective guarantees on performance improvements compared to the preceding mechanisms, we compared our approach to the most closely related systems structured in the hybrid memory architecture, *i.e.*, *CLOCK-DWF* [3] and *RaPP* [5]. Since *RaPP* relies

Table 4.1 Benchmark description

Benchmark	Ratio W:R	Description
gcc	1 : 3.24	Based on gcc version 3.2, generating code for Opteron
mcf	1 : 22.41	Vehicle scheduling, using a network simplex algorithm to schedule public transport
gromacs	1 : 4.82	Simulating Newtonian equations of motion for hundreds to millions of ppapers
cactusADM	1 : 2.07	Solving the Einstein evolution equations using a staggered-leapfrog numerical method
namd	1 : 8.25	Simulating large biomolecular systems
hmmer	1 : 0.003	Protein sequence analysis using profile hidden Markov models
astar	1 : 1.25	Pathfinding library for 2D maps, including the well known A* algorithm
wrf	1 : 3.34	Weather modeling from scales of meters to thousands of kilometers

on a special hardware block implemented inside of the memory controller and our design is a pure software-based solution, we could not directly compare our approach to *RaPP*. Thus, we elaborately redesigned *RaPP* in the *Page Placement Simulator* by implementing the functions including the access recency checking routine and access popularity-based ranked LRU queues in PCM and DRAM. We also implemented *CLOCK-DWF* in the *Page Placement Simulator*.

As the metrics for performance evaluation, we used the average memory access time and the total power consumption. First, in our experiment, the average memory access time is defined as the per-workload data which is computed by averaging the all latency values measured both on DRAM and PCM for the all page accesses of the workload. It is defined as follows:

$$\begin{aligned}
T_{avg} &= T_D \times \alpha + T_P \times (1 - \alpha) \\
T_P &= \frac{N_{pr} \times L_{pr} + N_{pw} \times L_{pw}}{N_{pr} + N_{pw}},
\end{aligned} \tag{4.1}$$

where T_{avg} means the average memory access time. T_D and T_P represent the average access times in DRAM and PCM, respectively, where α means the probability that a page is located in DRAM. The write and read latencies of PCM are denoted by L_{pw} and L_{pr} , respectively, and the write and read counts in PCM are represented by N_{pw} and N_{pr} , respectively.

Second, when a page mapped to a memory frame is read or written, it leads to the energy dissipation which is directly proportional to the active power consumption. Another factor which affects the energy dissipation is the static power consumption and it is commensurate to the size of memory frames. The total power consumption is the sum of both active and static power consumptions and it is described as follows [3]:

$$\begin{aligned}
P_{total} &= P_{static} + P_{active}, \\
&\text{where} \\
P_{static} &= UP_{static} \times Size \\
P_{active} &= \frac{N_r \times E_r + N_w \times E_w}{N_r \times L_r + N_w \times L_w}
\end{aligned} \tag{4.2}$$

UP_{static} is the statically consumed power per capacity, and includes both refresh power and leakage power [3]. *Size* refers to the size of the memory frames (*i.e.*, those in DRAM or PCM). Both N_r and N_w are the numbers of read and write counts, L_r and L_w are the average latencies of read and write

accesses, and P_r and P_w are the required amounts of power consumed for a read operation and a write one, respectively. All the detailed parameters used in this experiment are listed in Table 2.1.

The total elapsed time taken for a page access is made up of the L1/L2 cache memory access, the page frame access in main memory (*i.e.*, PCM and DRAM), and the page fault handling, and most of the time is spent on the page fault handling [3]. Therefore, as clearly explained in [3], even different page replacement algorithms are used, it is hard to compare the average memory access time among the algorithms under the environment with randomly generated page faults. For this reason, we excluded the page fault effect and only measured the hit access time in main memory (*i.e.*, PCM and DRAM). To do so, in every benchmark experiment, we reconfigured PCM size enough to host the total memory footprint and set the DRAM size smaller than PCM size. To realistically represent the hybrid memory architecture with small-sized DRAM and large-sized PCM, the ratio between the size of DRAM and that of PCM is configured to 3:7, 2:8 and 1:9, and then we compared them each other.

As shown in Figure 2.2, most of $N_{acc}(p)$ values of pages used in the workloads are smaller than 2^{10} so we set m as 10 in our experiment. Different from m , we should carefully determine the threshold value n and the period of *Recency-Checking* T_s which are explained in Section 3, as it affects strongly the efficacy of the proposed solution approach. First, if n is too large, the moment of page migration for a page p is delayed, thereby increasing the $N_{acc}(p)$ value in PCM. Whereas if it is too small, too many page migrations from PCM to DRAM occur, and in turn, if DRAM is full, the number of migrations from DRAM to PCM also increases. Second, if T_s is too large, it is difficult to distinguish a page with high recency from the one with low recency value. When T_s is too small, either *DRS* or *DWS* of a page may experience unnecessary change

too frequently. By conducting extensive experiments, we have found that $n=2$ and $T_s=1$ second are the most suitable for our target system. The detailed baseline system configurations for our experiments are shown in Table 4.2.

In this paper, we focus on the w -based biased page access counting method and the recency-based page migration policy from DRAM to PCM, under relatively small-sized DRAM compared to PCM. Thus, unlike aforementioned n , m and T_s , we evaluated the performance of the proposed solution with varying the w value and DRAM size.

4.2 Effect of Recency Checking

In selecting a victim page in DRAM to accommodate a popular page migrated from PCM, our proposed solution periodically checks the access recency of pages in the lowest level LRU queue in DRAM. To see if this procedure is effective or not, we measured the performance with and without our *Recency_Checking* algorithm executed. To clearly identify the pure impact of periodically performed *Recency_Checking* procedure, we set the weight value w defined in Equation 3.1, to one. It means that write and read accesses are regarded as the same ones, putting no more emphasis on the write access than the read one. We also set the ratio between the size of DRAM and that of PCM as 2:8.

The page migration between two different types of memory frames unavoid-

Table 4.2 System configurations for experiments

Processor Core	Intel <i>Core</i> TM i5-4460 CPU×4, 3.2GHz
L1 Cache	32KB, 32B lines, 32-way set associative
L2 Cache	1MB, 64B lines, direct mapped
Main Memory	4GB

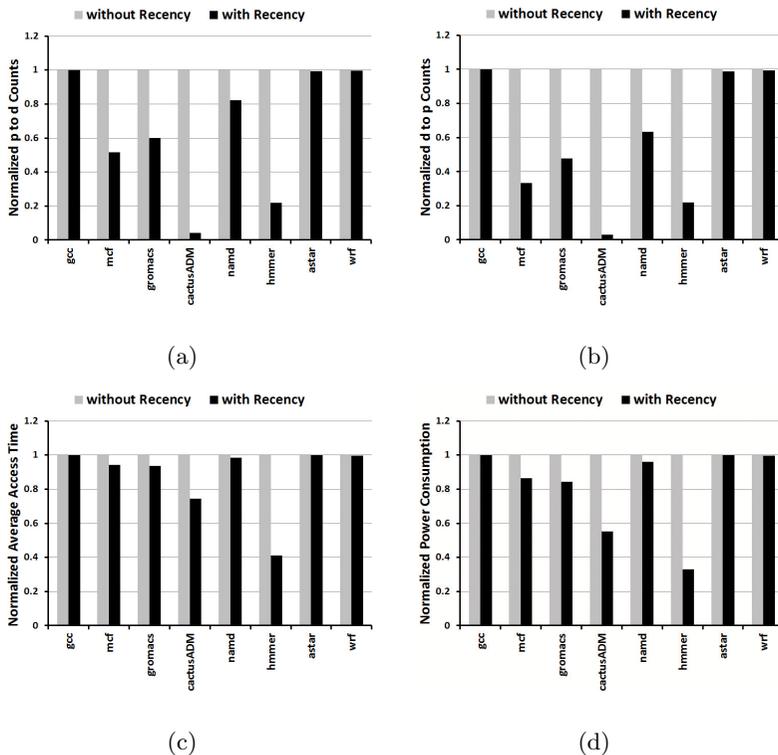


Figure 4.2 Evaluating the effectiveness of *Recency Checking*: (a) number of page migration from PCM to DRAM, (b) number of page migration from DRAM to PCM, (c) average access time and (d) power consumption

ably accompanies PTE updating and data copy, and thus it is an expensive operation. If the number of page migrations increases, it directly affects the memory access time. Thus, we measured the number of migrations from PCM to DRAM and that in reverse direction. Hereinafter, $p2d$ and $d2p$ represent the migrations in the directions, respectively.

Figure 4.2 displays the results relative to the scheme without our approach. Here, the scheme without our approach means that the victim page is only found at the head of the lowest level LRU queue in DRAM.

As shown in the figure, the reduced numbers of $p2d$ and $d2p$ are less than 2% in the cases of *gcc*, *astar* and *wrf*. However for the other five workloads, the number of $p2d$ is decreased by 17.7%~95.7% and that of $d2p$ is decreased by 36.8%~97%. According to both $p2d$ and $d2p$ results, their average access times are reduced by 2%~59%, and the power consumptions are reduced by 4%~67%, as shown in Figure 4.2(c) and Figure 4.2(d), respectively. In particular, $d2p$ causes PCM writes, and thus it is more directly related to the memory access time and the power consumption at the same time than $p2d$. Since the percents of reduced numbers of $p2d$ and $d2p$ for *gcc*, *astar* and *wrf* are smaller than 2%, the reductions in the average access time and the power consumption are also about less than 2%.

The reduced percentages of $p2d$ and $d2p$ for *cactusADM* are 95.7% and 97%, respectively, while they are 88% and 87.9% for *hmmer*. Counterintuitively, *hmmer* showed more reduced amounts of average access time and power consumption. The reason for this is that the number of hits in PCM writes of *cactusADM* is larger than that of *hmmer*.

4.3 Effect of page placement algorithm depending on w -based access frequency

As shown in Equation 3.1, the larger w value is, the more emphasis is put on the write access. We performed experiments while varying the w value from two to four, and in every experiment, we applied the *Recency-Checking* algorithm together and configured DRAM size as 20% of PCM size. We also compared the experimental results with those of *Clock-DWF* and *RaPP*. In every figure in this subsection, all the experimental results are normalized to those of *Clock-DWF*.

We do not show the experimental results obtained when $w=5$ or $w=6$ in this

paper because those with $w=4$ are basically similar to or sometimes slightly better than those with $w=5$ or $w=6$, on average; it is sufficient to show the results obtained when $w=2, 3$ and 4 because they demonstrate that it is beneficial to differentiate read accesses from write ones as in our approach, in the hybrid architecture.

4.3.1 Effect on the number of page migrations

We measured the numbers of $p2d$ and $d2p$, which are shown in Figure 4.3. As the w value increases, $N_{acc}(p)$ of page p decreases, and thus the number of *promotions* defined in Figure 3.1(a) is reduced. It is directly related to the reduced number of $p2d$, and it is shown in the result of every benchmark program in Figure 4.3(a).

This trend is also found in the reverse direction, *i.e.* $d2p$ depending on w . All the pages that have been migrated from DRAM are those which have turned out to be popular in PCM. Thus, if the number of $p2d$ decreases, the frequency of victim page selection in DRAM becomes low. As a result, $d2p$ rarely happens, and Figure 4.3(b) shows this result.

While *Clock-DWF* hosts only the written pages in DRAM, *RaPP* and our solution accommodate read pages in DRAM. In doing this, *RaPP* does not discriminate write accesses from read ones and it is similar to the case of using our proposed scheme when $w=1$. If $w=1$ or $w=2$, the increase speed of $N_{acc}(p)$ for a page p is faster than that in the cases, where $w=3$ or $w=4$, as shown in Equation 3.1. Thus, in the cases of *gcc*, *mcf*, *gromacs*, *astar* and *wrf*, our proposed solution with $w=2$ and *RaPP* show larger numbers of $p2d$ than those of *Clock-DWF*.

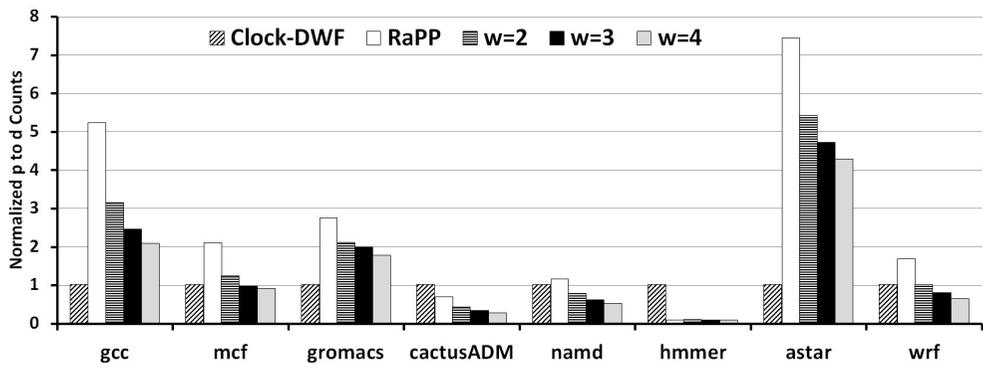
Unlike the case of $p2d$, in that of $d2p$, regardless of the value of w , our approach led to the smaller numbers of $d2p$ in all workloads than *Clock-DWF*

and *RaPP*. This result shows that our approach only keeps the pages with large $N_{acc}(p)$ values and high access recency in DRAM. Figure 4.3(c) shows the total counts of all the migrations during the execution of each benchmark program. In all the cases other than that of *gcc*, our approach led to the smaller numbers of migrations. Since *hmmem* is a highly write-dominated workload, all the write-intensive pages were kept in DRAM and the numbers of *p2d* and *d2p* were very small compared to the other workloads. In addition, both of the migration counts were not affected by varying w .

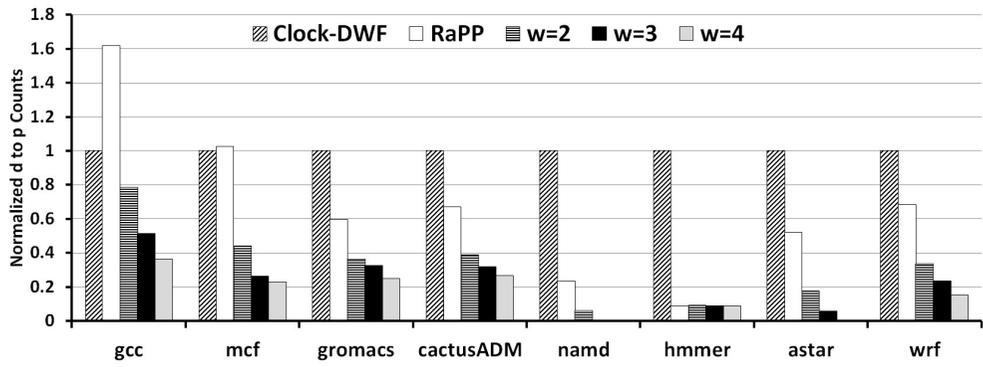
4.3.2 Effect on the average memory access time and the power consumption

Through the results of migrations between two different types of memory frames shown in Figure 4.3, our proposed scheme is expected to reduce the average memory access time and power consumption. As in the case shown in Figure 4.3, during the experiments, w was increased to two, three and four; each time, we measured the normalized average memory access time and power consumption of each workload, which are shown in Figure 4.4 and Figure 4.5, respectively.

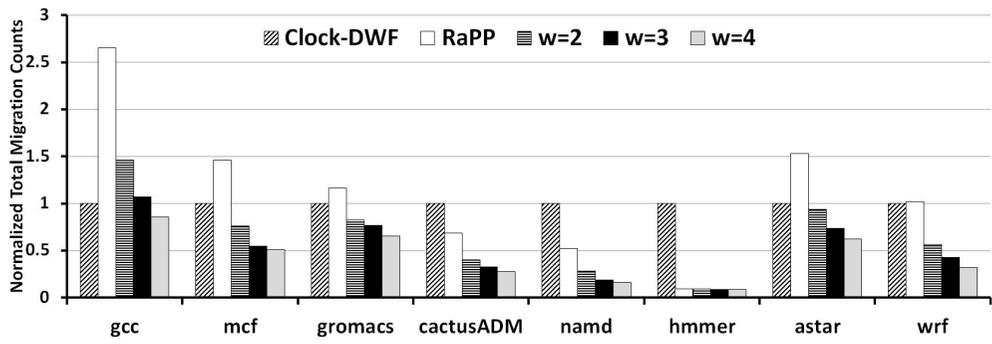
As shown in Table 2.1, there can be two cases of latency values for read access in PCM; The first case is when the read access time of PCM is twice that of DRAM, and the second one is when the read access latency of PCM is identical to that of DRAM. Since read access latency only affects memory access time, we report on the two cases only in the results of average memory access time rather than the results of power consumption. Figure 4.4(a) shows the first case. When $w=2$, the average memory access times were 61%~96% of those of *Clock-DWF*. When $w=3$, they were 62%~94.5% of those of *Clock-DWF*. Similarly, when $w=4$, they were reduced by 6.2%~36.5%. In all benchmark cases, our proposed scheme reduced the average memory access time.



(a)



(b)



(c)

Figure 4.3 Numbers of page migrations: (a) PCM to DRAM, (b) DRAM to PCM and (c) in both directions

Figure 4.4(b) displays the second case. *Clock-DWF* does not take into account read access counts in *p2d* while our proposed solution does. If PCM and DRAM have the same latency in the read access operation, there is no benefit in read access on DRAM. In this case, we cannot expect the enhanced average memory access time via our solution. However, as shown in Figure 4.4(b), except the results of *gcc* and *astar*, our solution denoted better performance in overall memory access speed than *Clock-DWF*. The reason for this is that the overall number of page migration was reduced under our scheme compared to *Clock-DWF*.

As expected, our proposed scheme could be demonstrated to be effective in terms of power savings. In Figure 4.5, when $w=2$, except for *gcc*, the power consumptions for all workloads were reduced by 3.4%~57%. And the reduced percentages of power reduction were 1.1%~56.8% and 4%~56.7% when $w=3$ and $w=4$, respectively.

As shown in Figure 4.3(b), the number of *d2p* decreases if the w value increases. However, the use of larger w values does not always lead to performance improvements in average memory access time and power consumption. In Figure 4.4(a), in the cases of *mcf*, *gromacs* and *namd*, the average access times with $w=2$ are smaller than the cases where $w=4$, and also as shown in Figure 4.5, their power consumptions with $w=2$ are smaller than those when $w=4$. There are two cases for writing data into one of page frames in PCM: PCM write hits and *d2p*. Even though in the case where $w=4$, a smaller number of *d2p* occurred, if there were a larger number of PCM write hits in the case, the average access time and power consumption could be larger than those in the case where $w=2$.

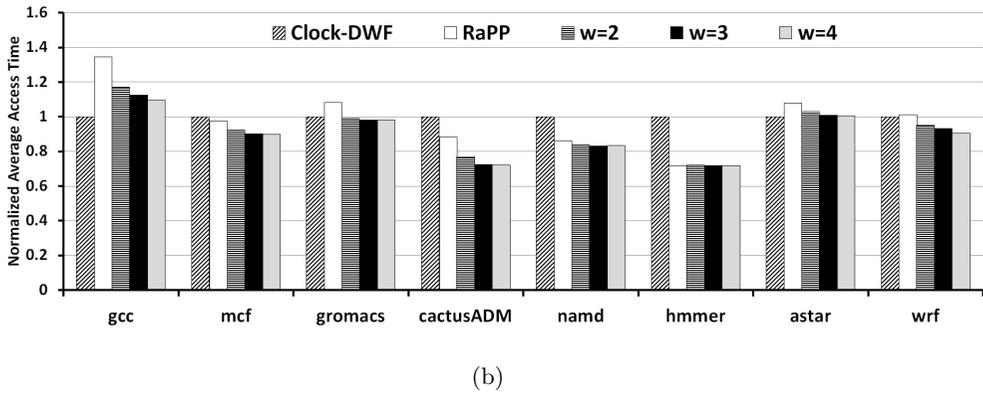
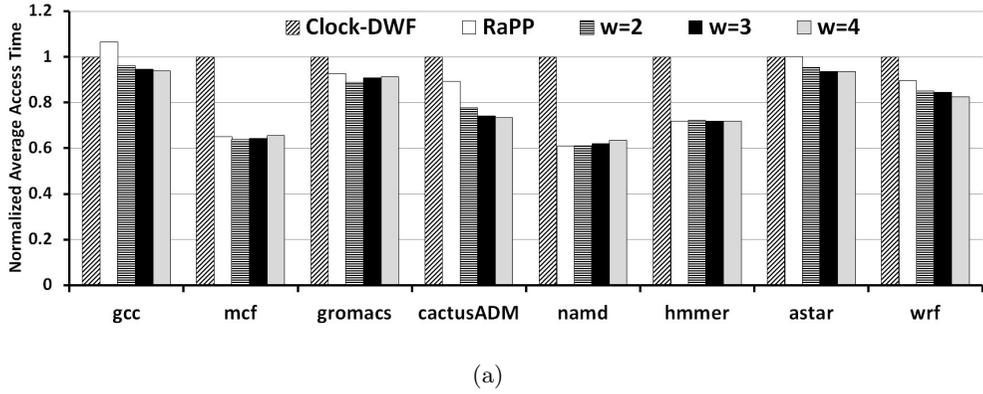


Figure 4.4 Average memory access time: (a) read latency of PCM = $2 \times$ read latency of DRAM, and (b) read latency of PCM = read latency of DRAM

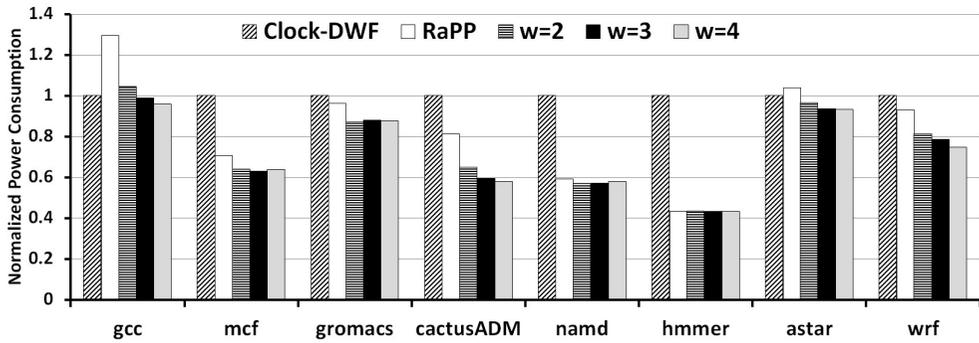


Figure 4.5 Comparison of power consumption

4.3.3 Effect on the Lifetime of PCM

As many research works addressed [1, 2, 3, 5], the hybrid memory architecture with PCM inevitably necessitates the development of page placement scheme which can reduce the PCM write count and can even out the write count in PCM to solve the endurance problem. We evaluated the lifetime of PCM through the number of write accesses on PCM. The purpose of taking our proposed approach was not to develop a good wear-leveling policy for physical frames in PCM. Moreover, many studies have proposed state-of-the-art wear-leveling algorithms for PCM [1, 3, 10, 11]. Therefore, in our experiments, we did not deal with the management scheme for the evenly distributed writing throughout the entire physical frames of PCM. We assumed that write counts for page frames are evenly distributed in PCM in our experiments. We also show the results, in this subsection, relative to those for *Clock-DWF*.

Figure 4.6 shows the result of the PCM write count according to w . When $w=2$, in the cases of *gcc* and *astar*, the use of our proposed scheme increased the write count by 80% and 170%, respectively, while with the other six workloads, that of ours could reduce the write count values by 7.5%~77.7%. When $w=3$, with the same workloads *gcc* and *astar* when $w=2$, the observed write count values were increased by 59% and 40%, however in the cases of the other six workloads, we could reduce the write count values by 12%~88%. Similarly, when $w=4$, the use of our scheme increased the write count values by 25%~43% with the same two abovementioned workloads while we could see that the other six workloads benefited from our approach by 13%~88%.

To see intuitively how our proposed approach affects the number of PCM write count, we averaged percentage values of all benchmark programs as shown in Figure 4.6 in each scheme. Table 4.3 shows this result. When $w=2$ in the case

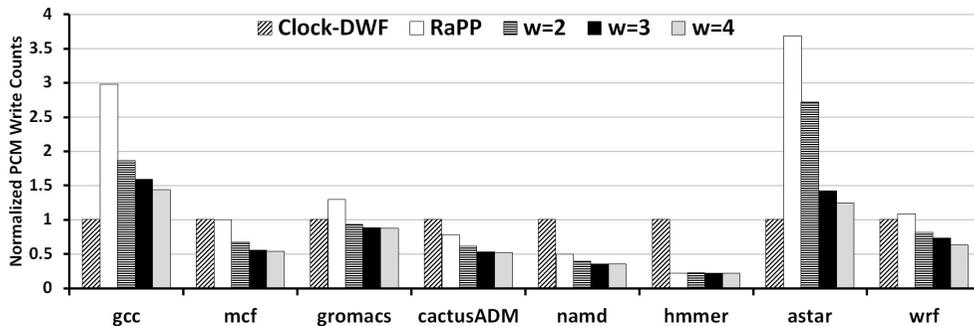


Figure 4.6 Comparison in PCM write count

Table 4.3 Overall average normalized PCM write counts with different values of w

	Clock-DWF	RaPP	1:2	1:3	1:4
PCM W/C	1	1.7	1.02	0.79	0.73

of using our proposed scheme, *Clock-DWF* and ours showed almost the same values, while the PCM write counts were reduced by 21%~27% when $w=3$ and $w=4$.

4.4 Performance analysis performed while varying the DRAM size

This subsection shows how our proposed algorithm performs with different sizes of DRAM in the hybrid memory architectures. To do this, we varied the DRAM size from 0% to 100% of PCM size. 0% is not meaningful because it cannot take advantage of using DRAM. Therefore, the experiment was performed with the size fixed at 1%. In addition, 100% is not possible in a hybrid structure, but it is meaningful to compare it with a conventional DRAM memory system. As the performance metrics, we made comparisons in the average memory access time,

power consumption and PCM write count. As in the figures in the previous subsection, all the resulting values in every figure are normalized to those of *Clock-DWF*. The x -axis in every figure indicates the size ratio between DRAM and PCM (*e.g.*, 3 : 7 means that the ratio between the size of DRAM and that of PCM is 3 : 7). The memory access patterns (*i.e.*, read- or write-intensive) for benchmark programs of SPEC CPU2006 are described in Table 4.1.

Figure 4.7 and Figure 4.8 show the results of the average memory access time and power consumption, respectively. As shown in both the figures, for read-intensive workloads such as *mcf* and *namd*, the effect of our proposed algorithm was particularly noticeable. The use of our algorithm reduced the access time for those two workloads by almost 40% when the size ratio and w were set to 3:7 and three. In the same configuration, the power consumption of the workloads were reduced by 32%~42%. These results come from the fact that read-intensive pages were more actively moved from PCM to DRAM than under *Clock-DWF*. Compared to other workloads in Table 4.1, read and write access counts of *astar* are balanced. In such a case, even not quite significant as in the cases of *mcf* and *namd*, the average memory access time and the power consumption were reduced by 5%~9% and 3%~8%, respectively.

For the case of a write-intensive workload such as *hmmmer*, as the DRAM size is smaller, the effect of our proposed solution was significant. The write-intensive benchmark program moves more pages from PCM to DRAM under *Clock-DWF* than under ours. Accordingly, if the DRAM size too small (*e.g.*, 1:9 in the x -axis), *Clock-DWF* performs *d2p* for victim pages more than our approach. As shown in Figure 4.7(f) and Figure 4.8(f), the performance gap between *Clock-DWF* and our solution is smaller as the DRAM size is larger in both memory access speed and power consumption. When the DRAM size is very small, the DRAM is saturated very quickly. In this case, the overhead due

to additional migration is severe because high level pages are often misidentified as victim. On the other hand, when using only DRAM, average memory access time is shortened due to the fast access speed of DRAM, but the static power is consumed so much that the advantages of PCM cannot be utilized. Figure 4.7 and Figure 4.8 show that when the DRAM size is between 10% and 30%, a relatively meaningful performance result is obtained.

Figure 4.9 shows the results of PCM write counts for the workloads in Table 4.1. When the DRAM size is configured to 10% and 20% of PCM size, 6 out of 8 benchmark programs represented reduced amount of PCM write count values under our solution regardless of w values. Whereas in case of 30%, a half of benchmark programs had the smaller number of PCM write counts with our algorithm and all the results with three different w values were pretty similar. When we averaged percentage values of all benchmark programs as did in Table 4.3, the differences among the results from the different three w and that of *Clock-DWF* were smaller than 6%.

These results infer that if the size of DRAM is smaller than or equal to 20% of PCM size, our proposed algorithm can be effective in terms of all the three metrics (*i.e.*, average memory access time, power consumption and PCM write counts). When the DRAM size is configured to relatively larger (in this experiment, greater than or equal to 30% of PCM size), the proposed algorithm can be comparable to *Clock-DWF* in terms of PCM write counts while making up for the weak points in the average memory access time and the power consumption.

As shown in Figure 4.7 and Figure 4.8, in every workload, the maximum difference among the proposed schemes with the three different w values was smaller than 8%, and in most cases, it was smaller than 5%. Unlike the results of the average memory access time and the power consumption, the number of

PCM write count was affected by the w values. As depicted in Figure 4.9(g), for example, the result when $w=2$ is larger than that of when $w=4$ by more than twice.

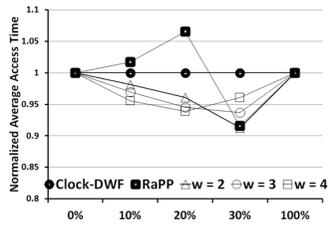
4.5 Techniques for further performance improvements

To further reduce the memory access time and power consumption in the environments of using various workloads, it is necessary to analyze the workload characteristics and patterns and configure the environments to suit them. One way is to identify the workload characteristics through the offline analysis of the workload. Depending on the type and strength of the access pattern, some parameters such as the weight of the internal algorithm can be automatically adjusted.

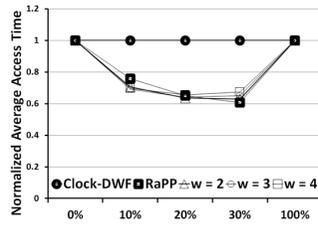
Another approach is to use program analysis techniques. Our algorithm currently performs page leveling based on the frequency of access to each page. In this case, we can observe how frequent references occur to each virtual address space through a trace analysis technique. It was expected that the early migration of high-level pages to DRAM would significantly reduce the additional migration overhead. To this end, we investigated at how much performance improvement could be made based on the analysis of trace. Some implementation details are as follows: first, the number of accesses to each page is counted in the trace generation step in Figure 4.1. Thereafter, the pages are sorted in the order where the reference occurs most frequently, and a mark is made, which is the number of pages with respect to the size of DRAM. As soon as the marked page is accessed in the simulation stage, it is adjusted to the set threshold level so that the page can be moved from PCM to DRAM. We compared the version with the TA (Trace Analysis) feature and the previous experimental results.

As shown in Figure 4.10, Figure 4.11 and Figure 4.12, average memory access time could be reduced by 20% on average and power consumption by 25% on average. When we added the trace analysis part in the previous experiment, the average memory access time was reduced by an average of 24% and the power consumption by 29% on average compared to the previous approach. The wear-out performance is also as good as the conventional algorithm.

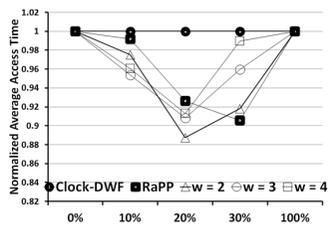
The above experimental result confirms that the migration overhead can be minimized by performing initial placement on the page based on the access frequency. This experiment is a preliminary one of memory algorithm research using static analysis to be carried out in the future. It is expected that the program execution flow can be utilized more effectively in page placement by using the static analysis technique. However, there are some challenges to improving the performance using static analysis. First, some delay due to static analysis can occur. If it is a time-critical workload, the overall program speed may be degraded by an internal delay for optimization. Second, it is not easy to map the dynamically allocated memory space to the virtual address space range. The meaning of the program execution should be analyzed in detail, and the diagnostic results obtained should be recognized either as the actual experimental result or false information. In particular, in real world workloads, it is often difficult to analyze the program at the actual code level, such as online profiling. Therefore, our future research will focus on analyzing the context of the program using static analysis techniques and aim to minimize the analysis delay and increase the accuracy.



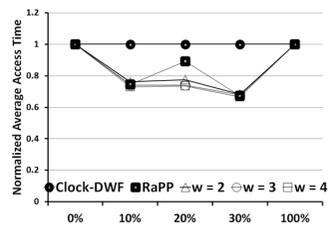
(a)



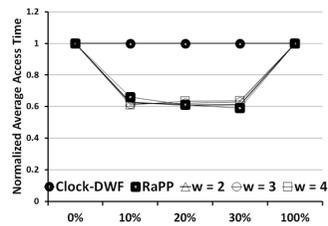
(b)



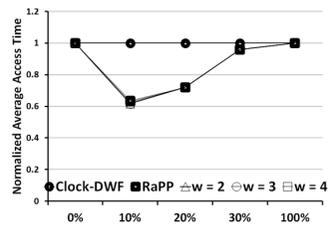
(c)



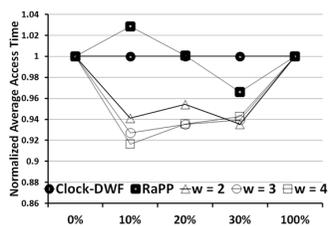
(d)



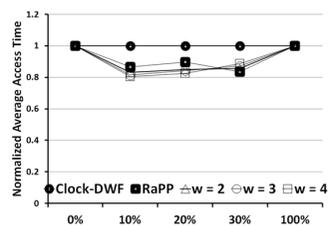
(e)



(f)

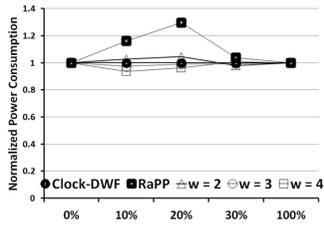


(g)

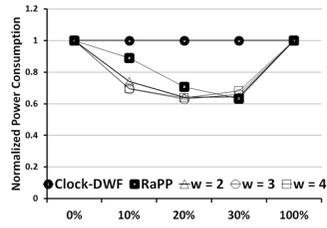


(h)

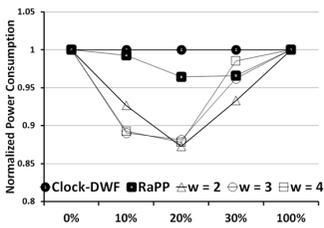
Figure 4.7 Average memory access time for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf



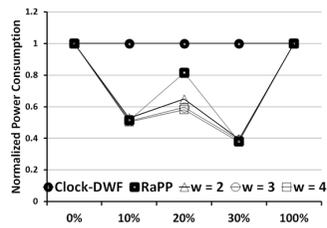
(a)



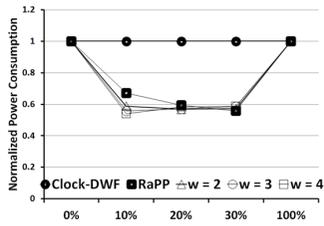
(b)



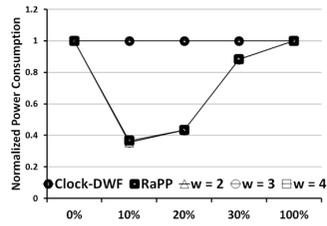
(c)



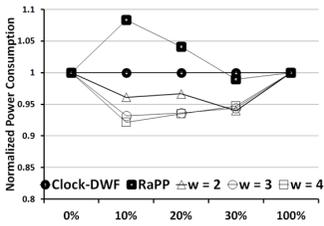
(d)



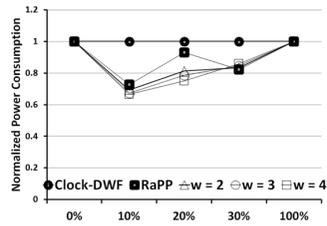
(e)



(f)

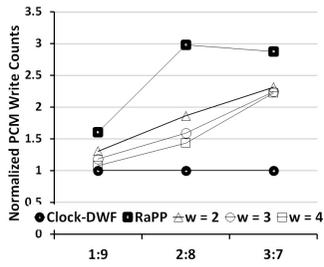


(g)

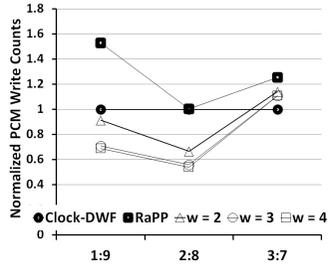


(h)

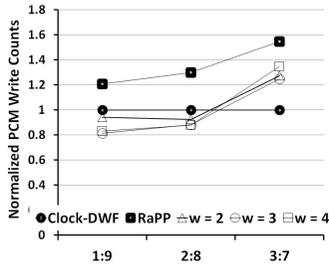
Figure 4.8 Power consumption for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf



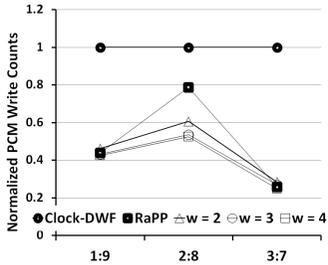
(a)



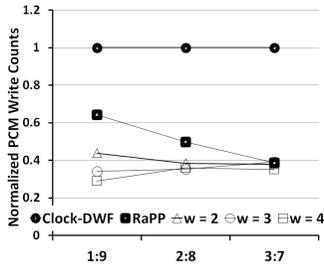
(b)



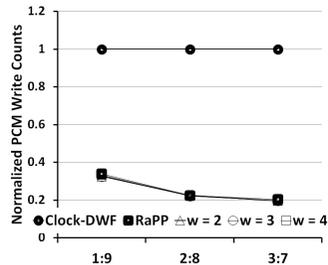
(c)



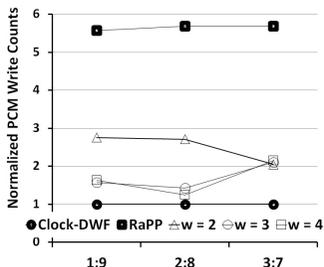
(d)



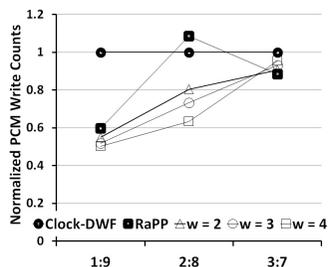
(e)



(f)



(g)



(h)

Figure 4.9 PCM write count values for SPEC CPU2006 benchmarks: (a) gcc, (b) mcf, (c) gromacs, (d) cactusADM, (e) namd, (f) hmmer, (g) astar and (h) wrf

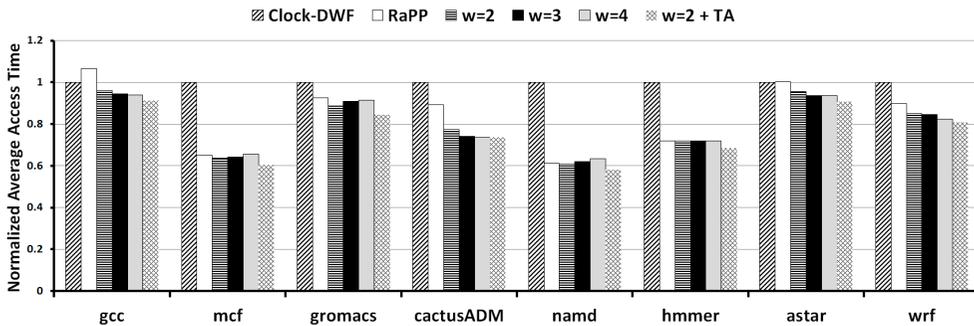


Figure 4.10 Average memory access time with TA(Trace Analysis)

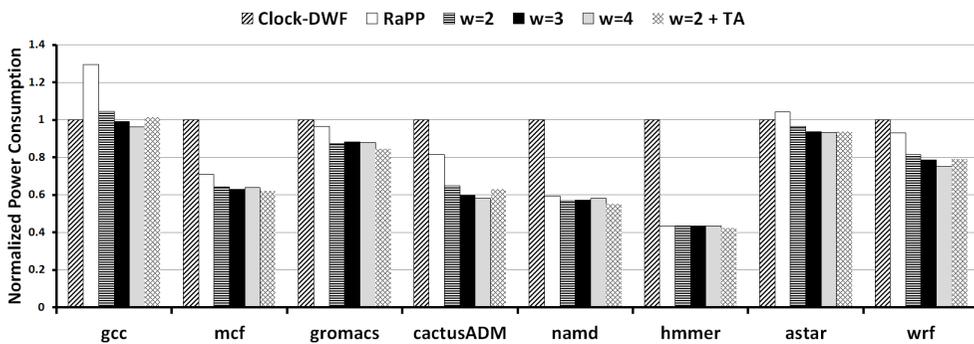


Figure 4.11 Comparison of power consumption with TA

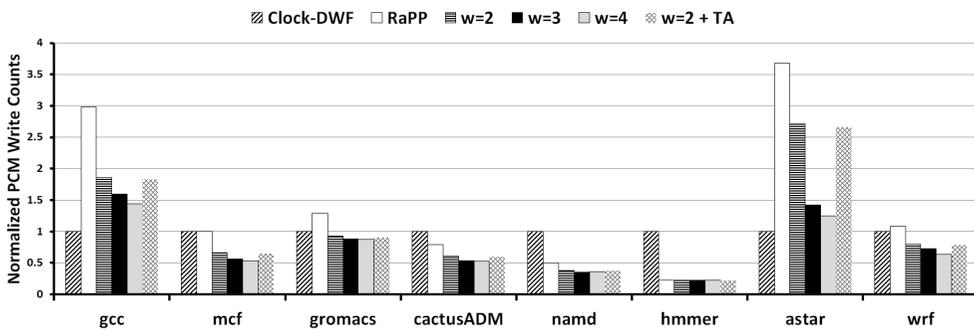


Figure 4.12 Comparison in PCM write count with TA

Chapter 5

Related Work

The memory architecture consisting of DRAM and PCM has been widely investigated in computing devices where issues regarding memory power conservation, memory access speed and PCM endurance are of particular importance. According to the role of DRAM, this architecture is classified into either the DRAM cache architecture or the hybrid memory one.

In the DRAM cache architecture [2, 8, 9], DRAM was used as a buffer cache for the main memory PCM. In a study [2], the authors presented an adaptive wear-leveling algorithm for a PCM main memory system with a DRAM buffer. The PCM-aware DRAM buffering mechanism reduced the PCM write count, and prevented skewed writing by taking into account the PCM write count and LRU-based clean data. With the adaptive multiple swapping and shifting scheme, they could even out the PCM write count values. In addition, unnecessary write operations were reduced by the buffer-aware swapping and shifting scheme.

In another study [9], the authors addressed the cost and power consumption

of computing systems if the systems adhered to DRAM-based main memory. Using an architecture model of PCM, they explored the trade-offs in the memory architecture that consists of small DRAM and large PCM storage.

Lin et al. proposed a hierarchical buffer cache architecture [8]. As the first-level buffer cache, they used DRAM to guarantee the required degree of access speed. By using PCM as the last-level buffer cache, the impact of frequent synchronous writes can be reduced. In order to improve the QoS in smartphone, a least-recently-activated first replacement policy (LRA) was implemented. Using this policy, the performance of the foreground application can be improved.

Some researchers have addressed various issues in the hybrid memory architecture, and our work is based on this memory architecture. Lee et al. proposed *CLOCK-DWF* [3]. It is based on the *CLOCK* algorithm [4] which checks the recency of a page by a reference bit and the write access by a dirty bit. *CLOCK-DWF* extended the original algorithm such that it applies write reference characterization.

Since the size of DRAM is relatively small in the hybrid memory system, it ranks pages in DRAM to find victim pages which will be moved to PCM. In doing this, it checks the dirty bit and the number of write count of a page to check the access frequency and that of rotation count to check that the page has been accessed recently. Thus, the recency and frequency of pages can be checked at the same time. As a result, *CLOCK-DWF* keeps frequently written pages in DRAM while read intensive pages are located in PCM, thereby reducing the total latency in accessing pages and system-wide power consumption. In *CLOCK-DWF*, when a page in PCM experiences a write access, it is migrated to DRAM. Thus, every single write access in PCM causes page migration from PCM to DRAM and a page in DRAM should be reclaimed if DRAM is fully occupied. This can lead to an excessive number of page migrations.

Based on the assumption that the lifespan of DRAM is infinite, *WP* was proposed in the study [6]. In this study, the OS managed DRAM as a write partition. It kept the infrequently written pages in PCM while only hot modified (high write frequency) pages were kept in DRAM. Therefore, it could avoid the long latency and high power consumption caused by write references in PCM.

When calculating the access frequency of a page, in both studies [3, 6] only the write reference was considered. Since the power consumption in PCM led by the read reference is twice larger than that of DRAM (as shown in Table 2.1), highly read-dominated pages located in PCM cannot be ignored. Unlike them [3, 6], Ramos et al. proposed *RaPP* [5] which takes into account read counts as well as write counts in assessing the access frequency of a page. It is a modified version of the multi-queue (MQ) algorithm [7]. It dynamically ranks pages according to their access frequency. *RaPP* is structured with LRU queues which are composed of page descriptors. The level of each queue is determined by the access frequency and more frequently accessed pages are ranked in higher level queues. *RaPP* uses a specific hardware (*i.e.*, implemented in the memory controller) and it is in charge of maintaining LRU queues and page migration between PCM and DRAM. Thus, the queue management and page migration are not visible to the OS. *RaPP* does not differentiate read access from write access in computing the access frequency of a page. Since both the latency and the energy consumption caused by write accesses are much larger than those for read accesses in PCM, we need to make a careful discrimination in counting the number of accesses based on the access types.

Chapter 6

Conclusion

In this paper, we pointed out two important factors in designing the hybrid memory architecture that consists of PCM and DRAM. First, maintaining only the written pages in DRAM can lower the overall memory access speed and increase power consumption. Second, the number of page migrations should be minimized since the migration from PCM to DRAM is directly linked to PCM writes, and thus it seriously affects the endurance problem and power consumption. To address these issues, we proposed an access-pattern-aware page placement algorithm. The algorithm predicts the future access pattern of a page by monitoring the access frequency and recency simultaneously. In evaluating the access frequency of a page, we used both write and read access counts while putting a more emphasis on write references. To do this, we defined the weight w . Using the w value, the access count value of a page is increased by one whenever just a single write reference or w (times) read ones are detected. To place a page with high access recency in DRAM, we devised a state-transition-based recency checking algorithm. In addition, we used a multi-level queue ranked by

the access frequency of pages. By exploiting the recency checking algorithm and leveled queue scheme, we could reduce the number of page migrations between two different types of memory frames. The experimental results show that our proposed algorithm can reduce the average memory access time by up to 39% and the power consumption by up to 57%, respectively, compared to the previous approaches. Additionally, they show that the average number of PCM write count can be reduced by 27%.

Bibliography

- [1] Chang, Hung-Sheng and Chang, Yuan-Hao and Kuo, Tei-Wei and Li, Hsiang-Pang. “A Light-Weighted Software-Controlled Cache for PCM-based Main Memory Systems,” *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15)*, 2015.
- [2] Park, Sung Kyu and Maeng, Min Kyu and Park, Ki-Woong and Park, Kyu Ho. “Adaptive Wear-leveling Algorithm for PRAM Main Memory with a DRAM Buffer,” *ACM Trans. Embed. Comput. Syst.*, Vol. 13. No. 4. ACM, 2014.
- [3] Soyeon Lee, and Hyokyung Bahn, and Sam H. Noh. “CLOCK-DWF: A Write-History-Aware Page Replacement Algorithm for Hybrid PCM and DRAM Memory Architectures,” *IEEE Transactions on Computers*, Vol. 63. No. 9. IEEE, 2014.
- [4] F. J. Corbato. “A paging experiment with the multics system,” *IEEE Transactions on Computers*, in Honor of P.M. Morse, MIT Press, 1969.
- [5] Ramos, Luiz E. and Gorbatov, Eugene and Bianchini, Ricardo. “Page Placement in Hybrid Memory Systems,” *Proceedings of the International Conference on Supercomputing (ICS '11)*, 2011.

- [6] Wangyuan Zhang and Tao Li. “Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures,” *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT '09)*, 2009.
- [7] Yuanyuan Zhou and James Philbin and Kai Li. “The Multi-Queue Replacement Algorithm for Second Level Buffer Caches,” *Proceedings of the General Track: 2001 USENIX Annual Technical Conference (ATC '01)*, 2001.
- [8] Lin, Ye-Jyun and Yang, Chia-Lin and Li, Hsiang-Pang and Wang, Cheng-Yuan Michael. “A Hybrid DRAM/PCM Buffer Cache Architecture for Smartphones with QoS Consideration,” *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 22. No. 2. ACM, 2017.
- [9] Moinuddin K. Qureshi and Vijayalakshmi Srinivasan and Jude A. Rivers. “Scalable high performance main memory system using phase-change memory technology,” *36th International Symposium on Computer Architecture (ISCA '09)*, 2009.
- [10] Chang, Hung-Sheng and Chang, Yuan-Hao and Hsiu, Pi-Cheng and Kuo, Tei-Wei and Li, Hsiang-Pang. “Marching-Based Wear-Leveling for PCM-Based Storage Systems,” *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 20. No. 2. ACM, 2015.
- [11] Asadinia, Marjan and Arjomand, Mohammad and Azad, Hamid Sarbazi. “Prolonging Lifetime of PCM-Based Main Memories Through On-Demand Page Pairing,” *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 20. No. 2. ACM, 2015.

- [12] Ping Zhou and Bo Zhao and Jun Yang and Youtao Zhang. “A durable and energy efficient main memory using phase change memory technology,” *36th International Symposium on Computer Architecture (ISCA '09)*, 2009.
- [13] John T. Robinson and Murthy V. Devarakonda. “Data Cache Management Using Frequency-Based Replacement,” *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '90)*, 1990.
- [14] Luk, Chi-Keung and Cohn, Robert and Muth, Robert and Patil, Harish and Klauser, Artur and Lowney, Geoff and Wallace, Steven and Reddi, Vijay Janapa and Hazelwood, Kim. “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation,” *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*, 2005.
- [15] <https://www.spec.org/cpu2006>
- [16] Binu P. John and Pradeep Nair and Eugene John and Fred W. Hudson. “Performance Measurement of Single, Dual and Quad Core Machines Using SPEC CPU2006,” *Proceedings of the 2009 International Conference on Computer Design (CDES '09)*, 2009.

초록

최근 이미지 프로세싱, 데이터 분석, 딥러닝 등 많은 분야에서 인메모리 컴퓨팅에 대한 요구가 늘고 있고 그에 따라 전체적인 메모리 사용량도 높아지고 있다. 상대적으로 작은 사이즈의 DRAM과 PCM을 함께 사용하는 하이브리드 메모리 아키텍처는 DRAM의 빠른 접근 속도와 PCM의 낮은 소비 전력이라는 장점을 극대화할 수 있는 구조로 현재까지 내부에 적용할 수 있는 알고리즘에 대한 다양한 연구가 진행되었다. 기존의 연구는 PCM의 느린 접근 속도와 셀 당 쓰기 횟수 제한이 있는 문제를 해결하는데 주로 초점을 맞추었으나 다음과 같은 문제가 있다. 첫째, 읽기 접근이 대부분인 워크로드에서는 DRAM을 잘 활용하지 못한다. 둘째, DRAM과 PCM간 페이지 이동이 상대적으로 많이 발생한다. 이러한 문제점을 해결하기 위하여 본 논문에서는 접근 패턴에 기반한 페이지 배치 알고리즘을 제안한다. 새로이 제안된 알고리즘은 쓰기 접근과 읽기 접근을 함께 고려하며 상대적으로 쓰기 접근에 가중치를 부여하여 멀티 큐를 기반으로 페이지 레벨링을 한다. 또한 메모리 간 페이지 이동 수를 최소화하기 위해 주기적으로 페이지의 최신성을 반영하여 DRAM에서 쫓겨날 페이지를 선정한다. 제안된 알고리즘을 반영하여 실험한 결과 기존의 결과에 비해 평균 메모리 접근 시간은 최대 39%, 전력 소모는 최대 57% 줄었으며, 추가적으로 PCM 마모 성능도 최대 27% 개선하였다.

주요어: 비휘발성 메모리, 페이지 배치 알고리즘, 하이브리드 메모리 구조, 상변화 메모리

학번: 2015-22901