UIMigrate: Adaptive Data Migration for Hybrid Non-Volatile Memory Systems

Yujuan Tan*, Baiping Wang*, Zhichao Yan†, Qiuwei Deng*, Xianzhang Chen* and Duo Liu*
*College of Computer Science, Chongqing University, China
†Department of Computer Science and Engineering, University of Texas at Arlington, USA

***Abstract*—Byte-addressable, non-volatile memory (NVRAM) combines the benefits of DRAM and flash memory. Its slower speed compared to DRAM, however, makes it hard to entirely replace DRAM with NVRAM. Hybrid NVRAM systems that equip both DRAM and NVRAM on the memory bus become a better solution: frequently accessed, *hot* pages can be stored in DRAM while other *cold* pages can reside in NVRAM. This way, the system gets the benefits of both high performance (from DRAM) and lower power consumption and cost/performance (from NVRAM). Realizing an efficient hybrid NVRAM system requires careful page migration and accurate data temperature measurement. Existing solutions, however, often cause invalid migrations due to inaccurate data temperature accounting, because hot and cold pages are separately identified in DRAM and NVRAM regions.**

Based on this observation, we propose UIMigrate, an adaptive data migration approach for hybrid NVRAM systems. The key idea is to consider data temperature across the whole DRAM-NVRAM space when determining whether a page should be migrated between DRAM and NVRAM. In addition, UIMigrate adapts workload changes by dynamically adjusting migration decisions as workload changes. Our experiments using SPEC 2006 show that UIMigrate can reduce the number of migrations and improves performance by up to 90.4% compared to existing state-of-the-art approaches.

I. Introduction

Traditional DRAM is facing limitations in terms of scaling and energy consumption [1], [2], [3], [4]. Emerging byte-addressable, non-volatile RAM (NVRAM) [5], [6], [7], [8], [9] promises higher density and lower energy consumptions on the memory bus, thus is becoming an attractive solution for replacing or complementing DRAM in future computer systems. However, most NVRAM devices also exhibit lower performance compared to DRAM, e.g., phase change memory (PCM) is expected to be 3–5× slower than DRAM. Blindly replacing DRAM with PCM will not meet the requirements for high memory performance.

An effective way to extend the memory size without sacrificing memory performance is to devise a hybrid NVRAM system [10], [11], [12], [13], [14], [15] that combines both DRAM and NVM: data is placed in and migrates between DRAM and/or NVM depending on its access patterns. In particular, hot (write-intensive) memory pages can be placed in DRAM to improve performance, while cold pages can be placed in NVRAM. This way, hybrid NVRAM systems can potentially achieve the benefits of both NVRAM and DRAM: large capacity from NVM and high performance from DRAM.

*Corresponding author: Yujuan Tan, E-mail: tanyujuan@gmail.com, Duo Liu, E-mail: liuduo@cqu.edu.cn, College of Computer Science, Chongqing University, Chongqing, China.

There have been many related proposals for data migration between DRAM and NVRAM, such as CLOCK-DWF [16], Double LRU [17], and Refinery Swap [18]. However, they often generate a large number of *invalid* page migrations, where the cost to migrate a page exceeds than the performance benefit. Existing studies have shown that page migration could bring huge overhead, such as page allocation and page table updates. It could take 90× the time used for programming a PCM page [18]. Invalid page migrations thus will significantly degrade memory performance in hybrid NVRAM systems.

We observe that the most common reason that leads to invalid page migration is inaccurate hot/cold pages identification. In particular, existing approaches use sperate mechanisms to identify hot and cold pages in DRAM and NVRAM, respectively. If a page is deemed hot, it will be migrated to DRAM; meanwhile, a victim "cold" page would be evicted from DRAM to NVRAM to make room for the incoming hot page from NVRAM. However, due to the use of different hot/cold pages identification mechanisms, the hot page identified in NVRAM may not be actually hotter than the cold page identified in DRAM, and the hot pages to be migrated to DRAM may be in fact colder than the victim page to be evicted. Page migrations in such cases are invalid since they violate the design principle that the DRAM should store hot pages while NVRAM should store cold pages.

In addition, existing approaches also use the same migration method for all workloads. However, different workloads have different page access patterns and workloads change overtime. Using a universal migration method cannot adapt to the different page access patterns and would cause many unnecessary page migrations to further affect memory performance.

Based on these observations, we propose *UIMigrate*, an adaptive page migration method which uses a unified hotness identification mechanism. UIMigrate consists of three parts, hot page identification, page migration, and self-adaptive adjustment. Our hot page identification method provides a way to uniformly measure the page hotness across DRAM and NVRAM. The page migration method decides which page is hot or cold based on the unified hot page identification method. After identified hot pages, UIMigrate migrates them to DRAM and evicts victim pages from DRAM. UIMigrate also self-adapts to changing workloads by adjusting migration parameters to promote or suppress page migrations, based on migration revenue and page access patterns.

UIMigrate can remove many invalid migrations to improve memory performance. Our experiments using ten benchmarks from the SPEC CPU 2006 [19] show that UIMigrate can

978-3-9819263-2-3/DATE19/©2019 EDAA

reduce the number of migrations and improve memory performance by 90.4% and 54% respectively compared to CLOCK-DWF, 91% and 48% respectively compared to Double LRU, 85.3% and 40% respectively compared to Refinery Swap.

The rest of this paper is organized as follows. Section 2 describes related work and our observations to motivate UIMigrate. The design and implementation of UIMigrate are detailed in Section 3. Section 4 evaluates UIMigrate and Section 5 concludes the paper.

II. Background

In hybrid memory systems, due to NVRAM's much lower write performance than DRAM, it is necessary to migrate hot pages in DRAM to improve memory performance. But limited by DRAM capacity, hot pages need to be identified and selected strictly and accurately. In this section, we first review related work on page migration with a focus on identifying hot pages, and then analyze their common problems to motivate our work.

A. Page Migration in Hybrid Memory Systems

To improve memory performance, existing approaches migrate two kinds of pages to DRAM: new pages and hot pages. If DRAM is full on migration, a victim page will be chosen and evicted to NVRAM. Next, we describe state-of-the-art approaches in this area.

CLOCK-DWF [16]. CLOCK-DWF first proposes to load the write-request pages into DRAM. For new pages, if it is for a write request, it will be swapped into DRAM. Otherwise, it will be placed in NVRAM. For pages stored in NVRAM, if one is hit by a write request, it will be migrated from NVRAM to DRAM. At this time when DRAM is full, CLOCK-DWF will select a victim page that has the lowest number of writes or that has not been accessed for the longest period of time in DRAM to be evicted.

Double LRU [17]. Double LRU recognizes the high migration cost between NVRAM and DRAM, and tries to restrict the number of page migrations by setting some threshold. It uses two separate LRU linked lists to manage pages in DRAM and NVRAM. For each page in NVRAM, it maintains a read/write request count. When a page is accessed, Double LRU checks its read/write request count, and if the count reaches a certain threshold, it will be migrated from NVRAM to DRAM; otherwise it will remain in NVRAM. In DRAM, the page at the end of the LRU list is always selected as the victim. For new pages, Double LRU stores them directly in DRAM, assuming new pages will be accessed frequently in the near future.

Refinery Swap [18]. Refinery Swap is similar to Double LRU: it also restricts page migration by setting a request count threshold. But Different from Double LRU, Refinery Swap only counts write requests. In DRAM, Refinery Swap uses three LRU lists to manage pages, including a write activity list, read activity list and cold data list. The write activity list and read activity list are used to store the last written and read pages, respectively. Refinery Swap periodically checks the access status of each page. If a page is not accessed in two consecutive periods, Refinery Swap moves it to the cold data list. Whenever DRAM is full, the page at the end of the cold data list is selected as the victim. In addition, for new pages, Refinery Swap stores them directly in the DRAM, like what Double LRU does.

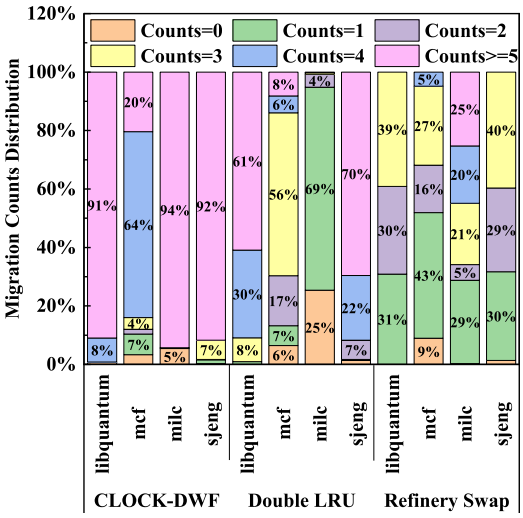


Fig. 1: The page migration count distribution of CLOCK-DWF, Double LRU and Refinery Swap.

B. The Page Migration Problem

In order to effectively utilize DRAM's high performance and avoid the impact of NVRAM's lower write performance, it is beneficial to migrate hot pages to DRAM. However, limited by DRAM's capacity, migrating a hot page to DRAM requires moving a cold page out of DRAM. This leads to a total of two migrations. Some prior work has showed that migrating or swapping a page can bring huge overhead. In some cases, migrating a page from NVRAM to DRAM could be 90× slower than writing a page in NVRAM. Thus, it is necessary to tradeoff the page migration cost and performance benefits by migrating a page to DRAM for the page access in DRAM instead of NVRAM. There is no need to migrate a page where the migration cost is larger than its performance benefit.

Moreover, when migrating a hot page to DRAM, a cold page must be selected to be moved out of DRAM immediately. In other words, the hot page migrated to DRAM is going to replace the victim cold page to enjoy the high performance of DRAM. Therefore, it is desirable to make sure that the migrated hot page will have more accesses than the victim cold page in the future. However, existing approaches fail to achieve this goal by using separate mechanisms to identify the hot pages in DRAM and the cold pages in NVRAM. For example, Double LRU and Refinery Swap use the write and read requests counts to measure whether a page is hot or cold in NVRAM, and simply use the last access time of each page to identify the victim cold page in DRAM. There is no means to compare the popularity of the hot page identified in NVRAM and the cold page identified in DRAM. Thus, it is highly probable that the migrated hot page is actually less popular than the replaced victim cold page. In this case, the migration will have no benefit and will degrade overall memory performance. Moreover, if the victim page to be

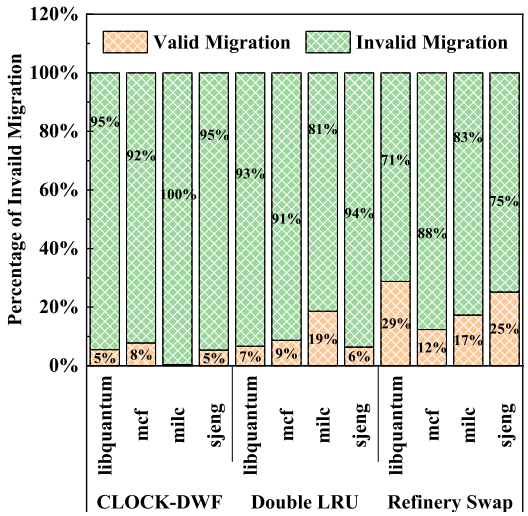


Fig. 2: The percentage of invalid migrations of CLOCK-DWF, Double LRU and Refinery Swap

replaced is potentially popular, it would return back to DRAM after a few page accesses. Similarly, if the hot page to be migrated is not so popular, it will be selected as a victim page to be moved out of DRAM in a short time. Thus, one invalid migration will result in another two or more invalid migrations, which would significantly degrade overall performance.

C. Statistical Evidence

We evaluated state-of-the-art page migrations approaches using four benchmarks from SPEC 2006 [19] by running 200 million instructions. The characteristics of the workloads are described in Table I in Section 4.1. We set DRAM size to 30% of the workload size and NVRAM is enough to store the entire workload. Figure 2 shows the page migration count distribution of CLOCK-DWF, Double LRU and Refinery Swap with the four workloads. As shown in Figure 1, CLOCK-DWF incurs the highest number of migrations due to the lack of migration restrictions. For CLOCK-DWF, more than 84% of the pages in all four workloads require at least four migrations. Double LRU and Refinery Swap showed reduced page migrations. Under Double LRU, more than 70% of the pages in `sjeng`, `mcf` and `libquantum` require at least three migrations, and under Refinery Swap, more than 90% of pages in all workloads require at least one migration. Although Double LRU and Refinery Swap have removed large amounts of migrations through thresholding, they still have a lot of invalid migrations where the migration cost is larger than the migration benefit due to the separated identification mechanisms in DRAM and NVRAM.

Figure 2 shows the percentage of invalid page migrations. A page migration is invalid if the migration cost is larger than the performance benefit we could get from it. As shown by the figure 2, CLOCK-DWF has more than 90% of invalid migrations for all workloads. Double LRU and Refinery Swap showed a reduced number of invalid migrations. For example, Double LRU reduced the invalid migration ratio from 100% to 81% for `milc`, and Refinery Swap reduced invalid migration ratios of all workloads to under 90%. However, due to use

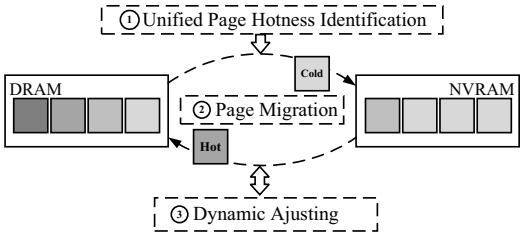


Fig. 3: UIMigrate Architecture

of separate page identification mechanisms in DRAM and NVRAM, they still incur a very high number of invalid migrations, degrading overall memory performance.

III. UIMigrate

Motivated by the observations that existing approaches use separate hot/cold page identification in DRAM and NVRAM that causes large amounts of invalid page migrations, we propose an adaptive page migration method based on a unified page hotness identification mechanism, called UIMigrate. UIMigrate consists of three parts: unified hot page identification, page migration, and self-adaptive adjustment. Figure 3 shows the architecture. The unified hot page identification and page migration mechanisms determine which page and how the page should be migrated. Self-adaptive adjustment adjusts the migration decision based on migration revenue to accommodate different page access patterns and workloads.

A. Unified Hot Page Identification

Unified hot page identification quantifies the hotness or popularity of each page in DRAM and NVRAM. Basically, page hotness can be directly counted by page accesses. UIMigrate adds one to the hotness value upon each access. However, based on access time locality, the page accessed more recently should be more popular. Thus, while updating page hotness upon each access, UIMigrate also uses an attenuation coefficient to lower the page popularity of old accesses. As shown in Equation 1, where $hot_{current}$ and hot_{old}represent the current hotness and the old hotness of each page respectively, d represents the attenuation coefficient:

$$hot_{current} = hot_{old} * d + 1 \tag{1}$$

Equation 2 shows how UIMigarte calculates the attenuation coefficient d. In the Equation, acc_count_{global} represents the total access counts of all the pages, acc_count_{page} represents the acc_count_{global} value recorded when the current page was last accessed, $DRAM_{size}$ represents the number of pages in DRAM. If all DRAM pages should be accessed in one cycle, $\frac{acc\_count_{global} - acc\_count_{page}}{DRAM_{size}}$ denotes the number of cycles the page that has not been accessed. Finally, d denotes the decay ratio of the old access popularity.

$$d = \frac{hot_{old}}{hot_{old} + \frac{acc\_count_{global} - acc\_count_{page}}{DRAM_{size}}} \tag{2}$$

B. Page Migration

The page migration component migrates pages using the unified hot page identification for pages in NVRAM and DRAM. It covers two kinds of pages: new pages from secondary storage and hot pages identified in NVRAM.

1) New Pages Migration: New pages are read from external storage devices (e.g., disks or flash) when page faults occur during program execution. Existing approaches, such as Double LRU and Refinery Swap, store them directly in DRAM, assuming they will be accessed frequently in the future. However, when DRAM is full, one victim page needs to be evicted. In this case, if the new page would have fewer accesses than the selected victim in the future, it is not beneficial to let the new page replace the victim page in DRAM. Based on this motivation, UIMigrate permits some new pages to be stored in NVRAM. It decides whether a new page should go to DRAM or NVRAM based on the quantified hotness of the victim page. UIMigrate sets a threshold, called $new\_page\_threshold$, to measure the hotness of each victim page. When the quantified hotness of a selected victim is larger than the preset threshold, it means that this victim page is too hot to be evicted, and so UIMigrate will store the new page in NVRAM. Otherwise, it will be migrated to DRAM.

2) Hot Page Migration: When a hot page is identified in NVRAM, it should be migrated to DRAM to improve memory performance. If DRAM is full, a victim page needs to be selected and evicted to make room for the incoming NVRAM page. Existing approaches focus on how to identify hot pages in NVRAM and how to select the victim page in DRAM, but fail to compare the access hotness between the hot page and victim page and measure the migration benefit, thus resulting in many costly invalid migrations. To solve this problem, UIMigrate first uses the unified hot page identification component to quantify the access hotness of all the pages in either DRAM or NVRAM. With the quantified page hotness under the same measurement, it then checks the hotness of the pages in NVRAM and the victim page in DRAM. When the hotness of a page in NVRAM is found to be higher than the hotness of the victim page in DRAM, it will be migrated to DRAM. UIMigrate uses an LRU linked list to organize the pages in DRAM; the page at the end of the list is selected as the victim.

To further reduce invalid page migrations and tradeoff the migration cost and benefit, UIMigrate sets two other thresholds: $hot\_page\_threshold$ and $cold\_page\_threshold$. These thresholds are used to prevent harmful migrations. $Hot\_page\_threshold$ prevents migrating pages that are not hot enough, while $cold\_page\_threshold$ prevents the victim page that is not cold enough to be evicted from DRAM. In other words, a page that can be migrated from NVRAM to DRAM must meet the following three conditions. First, the quantified hotness of this page must be larger than $hot\_page\_threshold$. Second, the quantified hotness of the victim page must be smaller than $cold\_page\_threshold$. Third, the quantified hotness of this page in NVRAM must be higher than that of the victim page in DRAM.

C. Self-Adaptive Adjustment

Workloads evolve over time and different workloads have different page access patterns. In order to effectively adapt to the change of access patterns, UIMigrate adjusts migration thresholds ($new\_page\_threshold$, $hot\_page\_threshold$ and $cold\_page\_threshold$) automatically to promote or suppress the page migrations according to real-time migration revenue.

Migration revenue is defined as the time saved by page accesses in DRAM rather than in NVRAM minus the time required for migrating the page. Equations 3 and 4 show the revenue calculations for new pages swapping into DRAM and hot page migrated from NVRAM to DRAM, respectively. In both Equations 3 and 4, W_{count} and R_{count} represent the number of page writes and reads after migration to DRAM, T_{PW} and T_{DW} represent the time for writing a page in NVRAM and DRAM, respectively. T_{PR} and T_{DR} represent the time for reading a page in NVRAM and DRAM, respectively. So $W_{count} * (T_{PW} - T_{DW}) + R_{count} * (T_{PR} - T_{DR})$ represents the time saved by the page writing and reading in DRAM rather than NVRAM. When considering migration time, Equation 4 calculates the time used for both the hot page migrated from NVRAM to DRAM (represented by $T_{P\_T\_D}$ and the victim page from DRAM to NVRAM (represented by $T_{D\_T\_P}$). But for new pages, Equation 3 only calculates the time used for the victim page migrated from DRAM to NVRAM (represented by $T_{D\_T\_P}$), without the time used for the new page entering DRAM. This is because if the new page enters NVRAM instead of DRAM, it also needs time to enter DRAM, and thus we exclude the time used for entering DRAM in Equation 3.

$$R_{new} = W_{count} * (T_{PW} - T_{DW}) + R_{count} * (T_{PR} - T_{DR}) - T_{D\_T\_P} \quad (3)$$

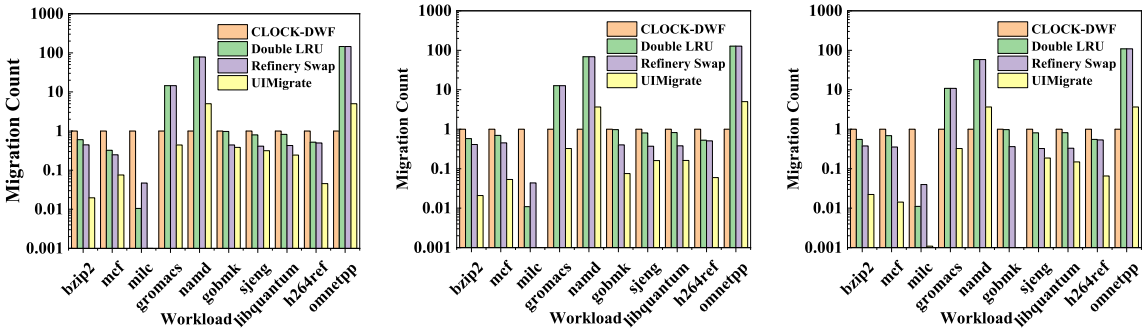
$$R_{N\_T\_D} = W_{count} * (T_{PW} - T_{DW}) + R_{count} * (T_{PR} - T_{DR}) - (T_{P\_T\_D} + T_{D\_T\_P}) \quad (4)$$

For cases where new pages are being swapped into DRAM or hot pages being migrated from NVRAM to DRAM, these pages will finally become cold and be selected as victim pages to be evicted. When they are evicted from DRAM, UIMigrate calculates the migration revenue based on Equations 3 and 4. If the migration revenue is below zero, it means that the migration cost is greater than the benefit. In this case, UIMigrate will increase $hot\_page\_threshold$ to prevent certain pages from getting hot in NVRAM and decrease $cold\_page\_threshold$ to prevent some pages from becoming cold in DRAM, thus retaining more pages in NVRAM. For new pages, UIMigrate will also reduce $new\_page\_threshold$, so that more new pages will go to NVRAM instead of DRAM. When the calculated migration benefit is larger than the migration cost, UIMigare will reduce $hot\_page\_threshold$ and increase $cold\_page\_threshold$ to make migrate more pages to DRAM, and increase $new\_page\_threshold$ to keep more new pages in DRAM.

IV. Evaluation

A. Experimental Setup

We developed a memory simulator to characterize the hybrid DRAM-NVRAM architecture and implemented UIMigrate approache. We also implemented CLOCK-DWF [16],



(a) DRAM Size: 20% (b) DRAM Size: 30% (c) DRAM Size: 40%

Fig. 4: The comparison of UIMigrate, CLOCK-DWF, Double LRU and Refinery Swap on migration counts.

TABLE I: The characteristics of the workloads.

Benchmarks	Footprints (KB)	Memory access count			Ratio of operations (reads : writes)
		Total	Reads	Writes	
bzip2	200664	1656858	1403233	253625	5.5:1
mcf	8724	914129	598516	315613	7.80:1
milc	8140	2471731	1920804	550927	3.86:1
gromacs	2464	66560	31611	34949	0.9:1
namd	1156	26143	13184	12959	1.01:1
gobmk	24876	19381982	9723166	9658816	1.01:1
sjeng	178904	16625603	8330468	8295135	1.01:1
libquantum	33072	3093250	1563233	1530017	1.01:1
h264ref	9760	92950	62630	30320	2.1:1
omnetpp	2168	20737	20420	317	64.4 : 1

Double LRU [17] and Refinery Swap [18], and use them as the baseline approaches. We take memory traces from ten benchmarks from SPEC 2006 [19] by running 200 million instructions. The characteristics of these workloads are shown in Table I. We compare UIMigrate and the three baseline approaches on migration counts, memory access latency and energy consumption. Memory access latency and energy consumption are calculated by the same parameters used in Refinery Swap [18]. We set DRAM size to between 10% to 50% of the workload size, and ensure that NVRAM is able to store the entire workload. Due to space limitation, here we only show the experimental results when DRAM size is 20% to 40% of the workload size.

B. Migration Counts

Figure 4 compares the normalized number of migrations required for UIMigrate and the other three baseline approaches. On average, UIMigrate's migration count is about 91% lower than Double LRU and 85.3% lower than Refinery Swap for all workloads. Compared to CLOCK-DWF, UIMigrate also reduced 90.4% of migrations for eight workloads. Such a small amount of migration benefits from removing large amounts of invalid migrations based on the unified hot page identification.

However, for `namd` and `omnetpp`, UIMigrate requires more migrations than CLOCK-DWF. This is because in these two workloads, there is only a few page writes, and in all of these page writes, most of them are concentrated on a small number of new pages. Thus, CLOCK-DWF needs the least migrations as it only migrates the write request of new pages to DRAM. While for Double LRU and Refinery Swap, they need much more migrations because they migrate all the new pages to DRAM, regardless of the request type. UIMigrate, which uses the adaptive $new\_page\_threshold$ to suppress new page migrations, needs many fewer migrations than Double LRU and Refinery Swap but more than CLOCK-DWF.

C. Latency

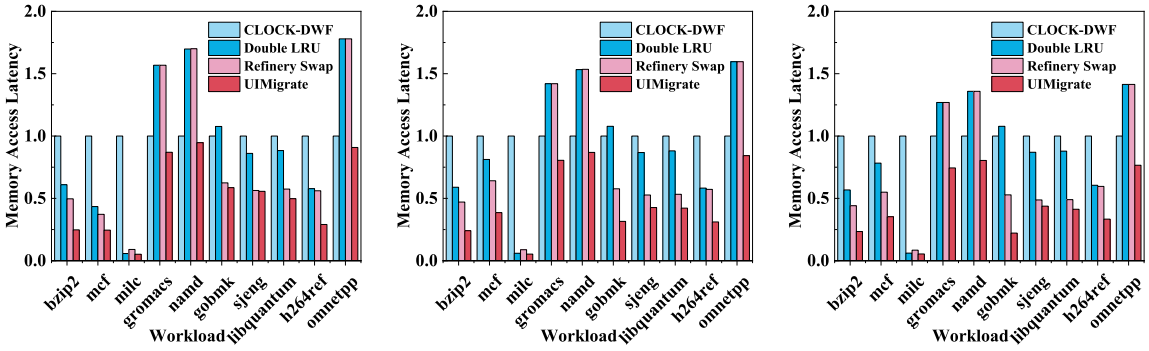
Figure 5 compares the memory access latency among UIMigrate and the three baseline approaches. All the results are normalized to CLOCK-DWF. As results show, UIMigrate has the lowest latency for all workloads. On average, UIMigrate's latency is about 48% lower than Double LRU, 40% lower than Refinery Swap, and 54% lower than CLOCK-DWF. It is worth noting that for `namd` and `omnetpp`, although UIMigrate requires more migrations, it still exhibits lower latency. This is because in UIMigrate, it uses $new\_page\_threshold$ that can be automatically adjusted based on migration revenue (See Section 3.3) to control page migration. For additional migrations required by UIMigarte, the migration advantage is greater than the migration cost, leading to lower memory access latency.

D. Energy Consumption

Figure 6 compares the energy consumption between UIMigrate and the three baseline approaches. All the results are also normalized to that of CLOCK-DWF. As shown in Figure 6, UIMigrate consumes the least amount of energy. On average, UIMigrate's energy consumption is 21% lower than Double LRU, 11% lower than Refinery Swap, and 42% lower than CLOCK-DWF. However, UIMigrate improves less on the energy consumption aspect than on the latency aspect. There are two reasons. First, UIMigrate calculates the migration revenue based on migration time and page access time to control page migration, without considering energy consumption requirements. Second, UIMigrate focuses on removing invalid page migrations, and each removal of the migration can save more access time than energy consumption.

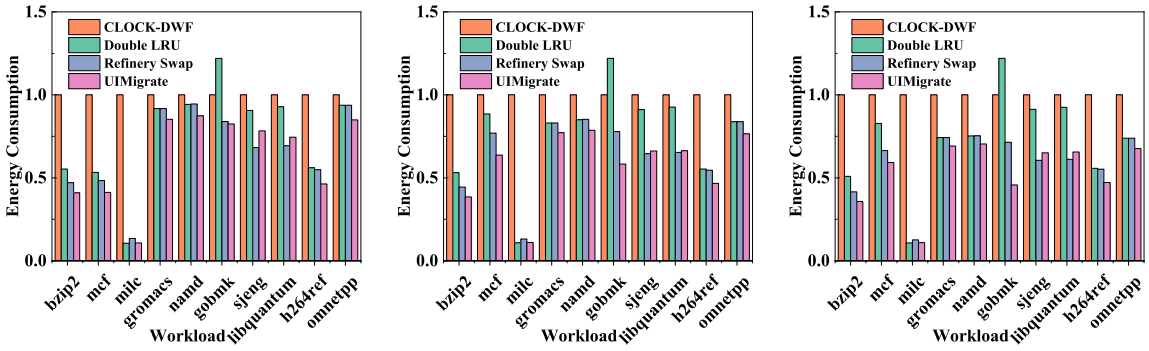
V. Conclusion

Existing page management approaches fail to identify hot/cold pages accurately and thus result in a large amount of invalid page migrations and suboptimal performance in hybrid NVRAM systems. This paper propose UIMigrate, an adaptive page migration method based on a unified hot page identification mechanism. Our experimental results show that UIMigrate can remove many invalid migrations and improve memory performance compared to state-of-the-art approaches



(a) DRAM Size: 20% (b) DRAM Size: 30% (c) DRAM Size: 40%

Fig. 5: The comparison of UIMigrate, CLOCK-DWF, Double LRU and Refinery Swap on memory access latency.



(a) DRAM Size: 20% (b) DRAM Size: 30% (c) DRAM Size: 40%

Fig. 6: The comparison of UIMigrate, CLOCK-DWF, Double LRU and Refinery Swap on energy consumption.

such as CLOCK-DWF (by 54%), Double LRU (by 48%) and Refinery Swap (by 40%). In the future, we plan to utilize machine learning techinques to better identify hot pages and make more accurate page migration decisions for better memory performance.

VI. Acknowledgement

This work is partially supported by grants from Fundamental Research Funds for the Central Universities (2018CDXYJSJ0026), National Natural Science Foundation of China (61402061, 61672116 and 61802038), Chongqing High-Tech Research Program (cstc2016jcyjA0274 and cstc2016jcyjA0332), China Postdoctoral Science Foundation (2017M620412), and Chongqing Postdoctoral Special Science Foundation (XmT2018003).

References

[1] S. Mittal, "A survey of architectural techniques for dram power management," *Int. J. High Perform. Syst. Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[2] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *SUPERFRI*, vol. 1, no. 3, pp. 19–55, 2015.

[3] J. S. Vetter and S. Mittal, "Opportunities for nonvolatile memory systems in extreme-scale high-performance computing," *Comput. Sci. Eng.*, vol. 17, no. 2, pp. 73–82, 2015.

[4] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1537–1550, 2016.

[5] M. Krause, "The solid state storage (r-) evolution," in *SDC*, 2012.

[6] S. Raoux, F. Xiong, M. Wuttig, and E. Pop, "Phase change materials and phase change memory," *MRS Bull.*, vol. 39, no. 8, pp. 703–710, 2014.

[7] G. W. Burr, M. J. Brightsky, A. Sebastian, H.-Y. Cheng, and Wu, "Recent progress in phase-change memory technology," *IEEE J. Emerg. Select. Top. Circ. Syst.*, vol. 6, no. 2, pp. 146–162, 2016.

[8] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," *Solid-State Electron.*, vol. 125, pp. 25–38, 2016.

[9] D. Liu and K. Zhong, "Durable address translation in pcm-based flash storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 475–490, 2017.

[10] L. E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," in *ICS*. ACM, 2011, pp. 85–95.

[11] P. Wu, D. Li, Z. Chen, J. S. Vetter, and S. Mittal, "Algorithm-directed data placement in explicitly managed non-volatile memory," in *HPDC*. ACM, 2016, pp. 141–152.

[12] W. Wei, D. Jiang, S. A. McKee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *PACT*. IEEE, 2015, pp. 163–173.

[13] J. Meza, J. Chang, H. Yoon, and Mutlu, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," *IEEE Comput. Archit. Lett.*, vol. 11, no. 2, pp. 61–64, 2012.

[14] B. Wang, B. Wu, and Li, "Exploring hybrid memory for gpu energy efficiency through software-hardware co-design," in *PACT*. IEEE Press, 2013, pp. 93–102.

[15] H. Yoon, J. Meza, and Ausavarungnirun, "Row buffer locality aware caching policies for hybrid memories," in *ICCD*. IEEE, 2012, pp. 337–344.

[16] S. Lee, H. Bahn, and S. H. Noh, "Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2187–2200, 2014.

[17] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid dram-nvm memory architecture," in *DATE*. EDA Consortium, 2016, pp. 936–941.

[18] X. Chen, E. H.-M. Sha, W. Jiang, Q. Zhuge, J. Chen, J. Qin, and Y. Zeng, "The design of an efficient swap mechanism for hybrid dram-nvm systems," in *EMSOFT*. ACM, 2016, p. 22.

[19] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, 2006.