AdaptHM: A Fully Adaptive Data Migration Strategy for Hybrid Memory Systems

Zhouxuan Peng, *Dan Feng, Senior Member, IEEE, Jianxi Chen, Member, IEEE,* Jing Hu, Yachun Liu, Jinlei Hu, Jintong Zhang, Tianyu Wan, and Zuoning Chen

***Abstract*—Data migration strategies (DMS) improve the overall performance of hybrid memory systems by migrating frequently accessed (hot) data to faster memory. However, designing an efficient DMS is challenging since the key metrics of DMS - *hot data selection*, *migration granularity*, and *migration frequency* - are sensitive to access patterns of workloads. Most existing strategies focus on only one of these metrics and often overlook the crucial impact of access patterns, resulting in sub-optimal performance and unnecessary migration traffic. In this paper, we propose AdaptHM, a fully access-pattern-aware Adaptive data migration strategy for Hybrid Memory systems. AdaptHM achieves adaptability on all three metrics through its unique multi-level data framework. First, AdaptHM adopts a *group-level* competition policy to select hot *blocks*, which responds faster to access patterns than threshold-based policies. Second, AdaptHM enables *segment-level* dynamic migration granularity by decoupling migration from remapping, which shows better access pattern resilience than existing schemes with fixed-size global migration granularity. Third, AdaptHM adjusts the migration frequency at *set-level* by periodically assessing the migration benefit, avoiding unnecessary migrations. Experimental results demonstrate that AdaptHM improves the performance by an average of 12.78% and reduces energy consumption by up to 37.24% compared to the state-of-the-art scheme.**

***Index Terms*—Heterogeneous Memory Systems, Non-Volatile Memory, Data Migration Strategy.**

I. Introduction

HYBRID memory systems, consisting of multiple memory technologies, have been widely studied in the last decade [1]–[30]. By exploiting advantages of different memory techniques, hybrid memory systems have shown strong potential to meet the growing demands for performance, capacity, cost, and energy consumption in modern applications. In general, a hybrid memory system comprises a fast, small Near Memory (NM), and a slow, large Far Memory (FM). The NM/FM abstraction is derived from previous works [5], [9], [10] and frees the following discussion from characteristic details of specific memory technology. There are two main approaches to building a hybrid memory system: the first uses NM as an additional cache layer between the Last Level Cache (LLC) and the FM [1], [16]–[23], the second approach uses both NM and FM as main memory to build a flat address space [2], [3], [5]–[8], [10], [14]. Recent research prefers the latter architecture since it provides higher aggregate bandwidth and larger available capacity. However, using NM and FM equally results in degraded overall performance due to FM's large capacity share and poor performance.

To enhance the overall performance of hybrid memory systems, data migration strategies (DMS) that dynamically identify and migrate hot data into NM are essential. A DMS-enabled hybrid memory system performs like the NM since most memory accesses are served by NM. However, it is non-trivial to design an efficient DMS for hybrid memory systems due to following challenges:

1) How to identify hot data? Some DMS migrate every accessed FM page into NM without distinguishing hotness [2], [5], [10]. Although these cache-like policies can easily achieve high NM utilization, they introduce a large amount of unnecessary migration traffic. As reported by [14], migration traffic accounts for more than 60%, wasting energy and bandwidth. Alternatively, other DMS only migrate hot pages whose access counts or prospective benefits exceed a specific threshold by monitoring access history information [3], [26]. However, previous work shows that migration thresholds required for optimal performance vary with workloads due to access pattern changes [14]. Although some works propose to adjust the threshold periodically in runtime [3], [26], they respond slowly to the access pattern changes due to long adjustment intervals, resulting in inaccurate hot data identification and suboptimal performance.

2) How much to migrate? The Migration granularity that refers to the amount of data migrated in a single migration, has a significant impact on performance [27]. A large migration granularity benefits from the prefetching effect on workloads with streaming access behaviors. However, it suffers from unnecessary migration traffic and performance degradation on workloads with random access behaviors. Experiments in [15] show that an unsuitable migration granularity can lead to performance degradation by up to 4.69x. Therefore, an appropriate migration granularity is crucial for an efficient DMS. Previous works usually set a fixed global migration granularity that achieves the best geometric mean performance for all workloads [3], [10]. However, a real-world workload may exhibit different access behaviors at different running stages [12], which leads to varying optimal migration granularity. Moreover, memory regions may exhibit different access

Zhouxuan Peng, Dan Feng, Jianxi Chen, Jing Hu, Yachun Liu, Jinlei Hu, Jintong Zhang, and Tianyu Wan are with the Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System, Engineering Research Center of Data Storage Systems and Technology, MoE of China, Huazhong University of Science and Technology, Wuhan 430074, China. (e-mail: hustpzx@hust.edu.cn; dfeng@hust.edu.cn; chenjx@hust.edu.cn; hujingreal@hust.edu.cn; lycspring@hust.edu.cn; hujinlei2020@hust.edu.cn; zhangjintong@hust.edu.cn; wanty@hust.edu.cn).

Zuoning Chen is with the Chinese Academy of Engineering, Beijing 100088, China. (e-mail: chenzuoning@vip.163.com);

Dan Feng and Jianxi Chen are the corresponding authors. E-mail: (dfeng@hust.edu.cn; chenjx@hust.edu.cn).

patterns due to concurrent access from various applications. Therefore, a fixed global migration granularity is almost impossible to achieve optimal performance for workloads.

3) When to migrate? The frequency and timing of migration are often tied to hot data identification. For example, cache-like policies trigger migration when an FM block is accessed, while threshold-based policies trigger migration when a new hot block occurs. However, neither cache-like nor threshold-based policies can guarantee future access to those migrated blocks. These policies make decisions based on short-term historical access behaviors, which carries the risk that the migration overhead may outweigh the benefits. Migration is valuable only if the migrated block receives enough accesses after migration to offset the migration overhead. Additionally, prior works observed that migrating the hot blocks with accesses accrued over a long period does not lead to better performance [6], [28]. As a result, adjusting the migration frequency over long periods to ensure overall performance benefits is also a critical factor for DMS efficiency.

Existing studies do not comprehensively address these challenges for hybrid memory systems. RHPM [14] uses the relative hotness to guide the page migration without thresholds, but it requires extra metadata for each memory page. GRAPM [27] infers the preferred migration granularity for given applications based on memory access characteristics. However, the optional granularity is limited to three choices without considering the memory-region difference of access patterns. DRAPM [12] adjusts migration thresholds based on an observation window, which then controls the migration frequency. However, DRAPM involves many predetermined parameters and long monitoring intervals. The adjustment method is not intuitive. In summary, existing DMS only focus on one of these challenges, leaving others to be handled with simplification. However, all these challenges are critical for an efficient DMS.

To tackle the challenges mentioned above, we propose AdaptHM, a fully **Adapt**ive data migration strategy for **H**ybrid **M**emory systems. Unlike previous schemes, AdaptHM comprehensively takes into account three critical factors: *hot data identification, migration granularity, and migration frequency*. It enables adaptability to access patterns across all metrics with a novel multi-level data framework. In summary, this paper makes the following contributions:

- **Group-level Hot Block Competition Policy.** AdaptHM utilizes a competition policy to identify the hot block within a group of memory blocks. All blocks within a group compete fairly for the opportunity to be migrated to NM. The block with the most accesses is declared the winner and is migrated. After migration, the winner block is challenged by other blocks through the winner score. The score increases if the winner block is accessed and decreases if other blocks are accessed. If the score decreases to zero, the winner block is replaced by the last block which decreased the winner score. The competition policy ensures that the winner block always has more accesses than other blocks and adapts to changing access patterns. As a result, AdaptHM avoids blindly migrating blocks like cache-like policies and does not rely on thresholds. Furthermore, AdaptHM only keeps the score for the winner block, eliminating the need for per-block access history metadata and significantly reducing the metadata overhead.
- **Segment-level Dynamic Migration Granularity Policy.** AdaptHM divides the memory address space into segments, each consisting of multiple continuous blocks. Then it tracks access patterns at the segment level and sets an appropriate migration granularity for each segment. AdaptHM enables three different migration modes based on access pattern classification: i) fixed-length adaptive mode, ii) spatial footprint prediction mode, and iii) transparent mode. The segment-level dynamic migration granularity decouples migration from the remapping structure, allowing AdaptHM to support flexible migration modes and granularity.
- **Set-level Benefit-based Migration Frequency Policy.** Although AdaptHM ensures that hot block selection and migration granularity adapt to access patterns, these heuristic designs can not guarantee that migrations will yield benefits. It is possible for a hot block to be generated by a long-term accumulation and incur migration overhead. To address this, AdaptHM periodically assesses migration benefits at the set level, which is a memory area cross covered by both a group and a segment. AdaptHM then controls the migration frequency by adjusting the winner score step of the hot block competition. If the assessment is positive in the last period, the migration frequency is adjusted in the same direction as the last period. Otherwise, the migration frequency is adjusted in the opposite direction. As a result, the migration frequency also adapts to access patterns.
- **Implementation and Evaluation.** We have implemented AdaptHM and evaluate its performance. Extensive experimental results demonstrate that AdaptHM outperforms the state-of-the-art data migration strategy by an average of 12.78% and reduces energy consumption by up to 37.24%.

II. Background and Motivation

A. Hybrid Memory

Modern applications are eager for memory with larger capacity, higher bandwidth, lower latency and cost efficiency. However, neither traditional DDR DRAM nor emerging memory technologies such as Non-Volatile Memory (NVM) [31], [32], 3D-Stacked DRAM [33], [34] can meet these diverse demands alone. Therefore, building hybrid memory systems with different memory techniques is a promising way to cope with the dilemma. A hybrid memory system can be configured in two ways: cache architecture or Part-of-Memory (PoM) architecture. In cache architecture, NM is used as a cache for FM, which can be easily deployed and transparent to the OS. However, the whole NM is invisible for applications, causing significant performance degradation for high-memory-footprint workloads [7]. Moreover, it faces the challenge of managing and querying a large number of tags [16], [19]. Alternatively, PoM architecture treats both NM and FM as main memory and enables DMS to migrate data between them, providing a larger available capacity and higher aggregate bandwidth than cache architecture.

The DMS can be managed by software or hardware. A software-managed DMS usually relies on the OS page table to

TABLE I
CATEGORIZATION OF EXISTING DMS WITH RESPECT TO VARIOUS DESIGN OPTIONS.

Design	Flexibility	Hot Data Identification	Remapping/Migration Granularity	Migration Frequency	Manager
POM [3]	CGroup+full-assoc	Competing counters	2KB/2KB segment	Threshold	HW
CAMEO [2]	CGroup+full-assoc	Cache mode	64B/64B cacheline	On access	HW
SILC-FM [5]	CGroup+set-assoc	Aging counter(segment) Cache mode(subblock)	2KB Segment/64B subblock	Threshold	HW
GRAPM [27]	All to All	Access bits	4KB/limited optional	Threshold	SW
HYBRID2 [10]	All to All	Access counter(sector) Cache mode(cacheline)	2KB Sector/256B cacheline	On access	HW
DRAPM [12]	All to All	Access bits	4KB/4KB page	Periodical&Global	HW+SW
RHPM [14]	CGroup+fix-remap	Relative hotness	256B/256B block	Relative hotness	HW
AdaptHM	CGroup+fix-remap	Adaptive competing	256B/Local Adaptive	Periodical&Local	HW

support address remapping, which limits the remapping granularity at the coarse memory page level (e.g., 4KB). Moreover, the migration procedure incurs high software-related overhead since it needs to interrupt the processor and trap into the kernel [4], [24], [25]. Instead, hardware-managed DMS have flexibility in remapping granularity with much lower migration overhead. However, it comes at the price of extra metadata structure required for address remapping.

B. Related Works on Data Migration Strategy

Plenty of prior work on data migration strategy for hybrid memory systems has identified many important design options. We analyze the impact of each design option and point out our design decisions and novelty with respect to related works. The most representative works are categorized in Table I.

Flexibility. The flexibility of remapping structure between NM and FM directly impacts the metadata space and operation overhead. Although all-to-all remapping offers the best flexibility [10], [12], [27], it requires the largest metadata space if the same remapping granularity is adopted. For example, HYBRID2 [10] has to maintain a remap table and an inverted remap table to support its all-to-all remapping, which occupies considerable available memory capacity and incurs metadata querying overhead. To shrink the metadata space, other schemes adopt Congruence Groups (CGroup) to manage remappings [2], [3], [5], [14]. Remappings and migrations are limited within a specific group of pages in these works. The tradeoffs of flexibility choice within a CGroup are similar to cache organization. In a nutshell, higher flexibility requires large metadata space. Considering that AdaptHM requires extra metadata to support its adaptability on migration granularity and frequency, we adopt the same flexibility option as RHPM [14] since fixed-remapping leads to the least metadata overhead.

Hot Data Identification. The efficiency of a DMS is mainly determined by the accuracy of hot data identification. Therefore, the hot data identification policy is a core component of every scheme. For software-managed DMS, they rely on access bits of the OS page table to decide whether a memory page is worth migrating [12], [27]. However, the identification accuracy of access bits is poor since it cannot distinguish the hotness between pages. Instead, hardware-managed DMS track memory access by counters, leading to more precise, fine-grained hot data identification [2], [3], [5], [10]. However, setting an appropriate threshold is challenging for counter-based DMS since the optimal threshold varies with access patterns [14]. RHPM [14] proposes relative hotness to guide data migration, which avoids thresholds by hotness competing between pages. However, the per-page counters and corresponding operations incur considerable overhead. In this work, we propose a group-level hot block competition policy to identify hot blocks. It does not require per-block metadata or rely on thresholds. In addition, this method offers an opportunity for AdaptHM to adjust migration frequency based on observed benefits.

Remapping/Migration Granularity. The remapping granularity and migration granularity were tightly coupled in early DMS [2], [3]. However, they actually lead to different tradeoffs. In general, remapping granularity directly impacts the remapping table size since it determines the number of remapping entries under a certain NM capacity. Finer remapping granularity yields larger metadata space. Instead, migration granularity impacts performance and migration traffic. A large migration granularity benefits from the prefetching effect on workloads with streaming access behaviors, but results in performance degradation and unnecessary migration traffic on workloads with random and fined-grained access behaviors. Recent DMS (SILC [5]) uses a large remapping granularity (2KB) to shrink remapping table and a finer migration granularity (64B) to avoid unnecessary migrations [5], [10]. However, the fixed fine-grained migration granularity waives the prefetching benefit and incurs more software overhead. Although some software-managed DMS propose dynamic migration granularity based on application demands [13], [27]. However, they only offer limited granularity choices with a coarse-grained 4KB basic granularity since they rely on the OS page table. Unlike prior works, our design provides flexible migration granularity choice, which could be any multiple of a basic block within a segment. Moreover, we enable the adaptability of local migration granularity for each segment based on observed access patterns. The migration granularity can even be discontinuous once a recurring access pattern is captured.

Migration Frequency. The migration frequency is often overlooked and simply tied to hot data identification by prior works. In other words, the migration is triggered once a new hot block occurs. However, heuristic hot data identification methods that are based on history access information cannot guarantee future access to those migrated blocks. Migrations do not always yield performance benefits but can also incur

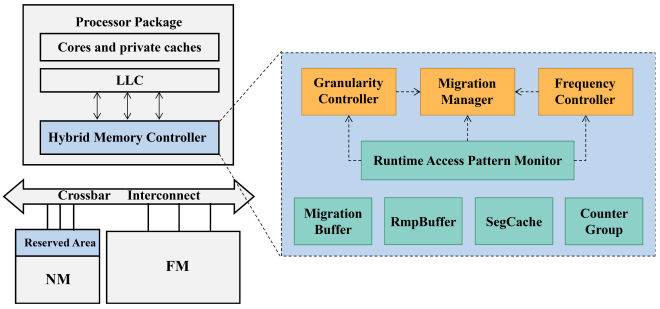


Fig. 1. The system overview of AdaptHM.

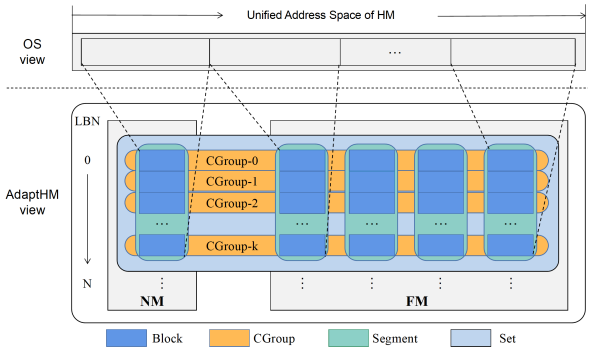


Fig. 2. The multi-level memory layout of AdaptHM.

data movement overhead and unnecessary migration traffic on workloads with variable access patterns. DRAPM [12] proposes to adjust the global migration frequency periodically by controlling the hotness thresholds based on observed migration behaviors. However, it does not consider the regional difference of access patterns under concurrent environments. In this work, we present a benefit-based migration frequency control approach, which adjusts migration frequency for each set based on periodical assessment. Moreover, each set independently adjusts its frequency, avoiding synchronous updates to all sets at the end of the assessment period.

In summary, existing data migration strategies are likely to result in sub-optimal performance and unnecessary migration traffic since they only focus on one of these critical aspects. Moreover, we observe that access pattern variation has a direct impact on hot data identification, migration granularity and migration frequency. The optimal decisions on all metrics vary with access pattern. The decisions become more difficult in a common case where multiple applications with different access patterns run concurrently. In this case, memory regions may exhibit different access patterns, requiring different migration thresholds, granularity, and frequency. However, existing DMS are not aware of the regional differences in access patterns.

The above analysis motivates us to explore a novel access-pattern-aware adaptive data migration strategy for hybrid memory systems. The main novelty of our design is the following: Firstly, AdaptHM comprehensively considers all the critical design options, unlike prior works that focus on only one of them. Secondly, AdaptHM enables access pattern adaptability on migration granularity and frequency for memory regions. Finally, the proposed three techniques cooperatively work together and are implemented on a unique multi-level data organization framework.

III. Design of AdaptHM

A. Overview

Fig. 1 shows the overview of AdaptHM, which is mainly implemented in the hardware-based hybrid memory controller (HMC) between LLC and main memory. The hardware structures required by AdaptHM are marked in green, and the logical components are marked in yellow. The underlying hardware structures are mainly used to accelerate the upper components. Additionally, AdaptHM reserves a part of the NM area for storing necessary metadata.

We briefly introduce the workflow of AdaptHM and explain the role of each component:

1) The HMC intercepts every memory request from LLC. The *Runtime Access Pattern Monitor (RAPM)* (III-D1) firstly extracts useful information from the coming request. The processed data is then fed to three core logical components. Overall, the RAPM is responsible for tracking access history information and identifying useful access patterns.
2) The request is then delivered to the *Migration Manger*, which executes our hot block competition policy (III-C). The Migration Manager stores a few frequently accessed remapping entries in the *RmpBuffer* (III-F1) to accelerate metadata querying. Besides, the *Migration Buffer* (III-F2) assists migration procedure. The Migration Manager determines whether a new hot block occurs. If no hot block occurs, it forwards the request to the actual address.
3) Once a new hot block occurs, the Migration Manager obtains the optimal migration granularity and mode from the *Granularity Controller* (III-D). The Granularity Controller decides suitable granularity and mode for those currently active segments. Granularity information is then stored in the *SegCache* (III-G2).
4) When the assessment period of a set is over, the *Frequency Controller* (III-E) calculates migration benefits based on information from the *Counter Groups* (III-F3). Then, it adjusts migration frequency based on the assessment result.

Overall, the HMC receives LLC misses as input and outputs memory requests to NM/FM. These HMC outputs fall into two categories: i) HMC forwards LLC misses to the correct memory addresses based on remapping metadata. ii) HMC issues memory requests to migrate data blocks or fetch/update metadata. During the workflow, the HMC remains transparent to the upper cache hierarchy and does not affect any of the system's original logical processes.

B. Memory Layout and Organization

From an OS perspective, hybrid memory is a unified linear logical address space composed of NM and FM. In order to support efficient data migration and adaptive capability, AdaptHM builds a multi-level logical data organization framework on top of the physical address space, as shown in Fig. 2. The address grows form top to bottom then from left to right, which means that the blocks in a segment are continuous in the vertical direction, while the blocks in a CGroup are scattered in the horizontal direction (LBN is an abbreviation

for Logical Block Number). A set is the area covered by both segments and CGroups. This layout provides several advantages in AdaptHM's design. Below, we will introduce each level and related design considerations in detail.

Block. A block is the basic memory management unit of AdaptHM, which means it servers as both the remapping granularity and the minimum migration granularity. Therefore, the block size involves trade-offs on two aspects: remapping table size and migration efficiency. In general, a large block size helps to shrink the remapping table size. However, it results in coarse-grained migration. To balance the requirement on both sides, we set a middle block size of 256B, following the experience of previous works [10], [14].

CGroup. Theoretically, an FM block can be remapped into any location in NM. However, such all-to-all remapping requires a huge metadata space. Therefore, AdaptHM adopts the congruence group (CGroup) concept to manage remapping. A CGroup consists of blocks that have the same offset with respect to the NM size (as shown in Fig. 2). To simplify the address translation and metadata structure, the CGroup follows a fixed-remapping manner with only one NM block in each CGroup. Thus, the number of blocks within a CGroup depends on the capacity ratio between NM and FM. AdaptHM executes the hot block competition policy on each CGroup to find the hot block that is worth migrating into NM. It should be noticed that blocks within a CGroup are scattered and positioned far apart from each other within the physical address space. This placement helps prevent the concentration of hot blocks within a CGroup due to spatial locality, reducing the likelihood of frequent competitive accesses.

Segment. To track access patterns of different memory regions, AdaptHM divides the memory logical address space into segments. As shown in Fig. 2, a segment consists of a fixed number of logically continuous blocks. The segments and CGroups are orthogonal in memory logical space. Therefore, AdaptHM can set appropriate migration granularity for each segment without interfering with the remapping structure. The segment size is set to 16KB for two reasons: 1) A 16KB segment is enough to cover optimal granularity for most of the evaluated workloads based on a sensitivity analysis of POM [3]. 2) A 16KB segment contains 64 blocks, which supports fetching all remapping entries of a segment with just one cacheline fetching (Each remapping entry is 1B, detailed in Section III-G).

Set. A set is a memory area that is covered by segments and CGroups as shown in Fig. 2. The set size is determined by segment size and capacity ratio between NM and FM. AdaptHM assesses the migration benefits at the set level since we can monitor migrations incurred by both hot block competition policy in CGroups and dynamic granularity policy in segments. Then, AdaptHM sets a suitable migration frequency for each set to avoid migration overhead.

Below, we describe all the critical components based on this multi-level data framework.

C. Group-level Hot Block Competition Policy

The hot data identification policy is a core component of any data migration strategy. However, the two existing mainstream policies have flaws: i) Cache-like policies incur a large amount of unnecessary traffic due to blind migration. ii) Threshold-based policies face the challenge that the optimal threshold varies with access patterns. Meanwhile, dynamic threshold methods react slowly to access pattern changes due to the long adjustment period. Moreover, most of existing hot block identification policies require per-block metadata to compare hotness between them, resulting in overhead on operation and metadata space.

Algorithm 1 Hot Block Competition Policy

Input: The address $addr$ of coming memory request
  Calculate LBN n of the accessed block b
  Obtain the $WinnerBlk$ and $WinnerScore$ of CGroup-n
  if b is the $WinnerBlk$ **then**
    $WinnerScore$ += INC_Step
  else
    $WinnerScore$ -= DEC_Step
    if $WinnerScore = 0$ **then**
      Update $WinnerBlk$ and trigger migration
    end if
  end if
  Forward request to actual address.

To efficiently identify hot blocks with low cost, AdaptHM proposes a group-level hot block competition policy. In a CGroup, all blocks fairly compete for the sole NM block. The block with the most accesses will be the winner, which will be migrated into NM later. Meanwhile, the winner block is challenged by other blocks in the same CGroup. Algorithm 1 shows the process of our hot block competition policy. AdaptHM maintains a WinnerScore for each CGroup. If the winner block is accessed, the *WinnerScore* is increased by an *INC_Step*. Otherwise, the *WinnerScore* is decreased by a *DEC_Step*. Both *INC_Step* and *DEC_Step* are set to 1 by default. Once the *WinnerScore* drops to zero, the current winner block is replaced by the last block which decreased the *WinnerScore*. Then a migration is triggered to move the new winner block into NM.

The competition rules ensure that the winner block has more access than other blocks. Overall, the proposed hot block competition policy has three advantages: i) It avoids blindly migrating blocks that may be rarely accessed, compared to cache-like policies. ii) It does not rely on thresholds to determine a hot block. Instead, the winner block is elected through competition between blocks. iii) Per-block metadata is not required. AdaptHM only maintains a *WinnerScore* for each CGroup but not each block. Corresponding metadata operations are also simplified.

D. Segment-level Dynamic Migration Granularity Policy

As analyzed in Section II-B, existing DMS tend to choose an optimal global migration granularity for all evaluated workloads. However, memory regions may exhibit different access patterns due to concurrent accesses from various applications. Each memory region may have different optimal migration granularity. Hence, a global migration granularity is almost impossible to achieve optimal performance for all workloads.

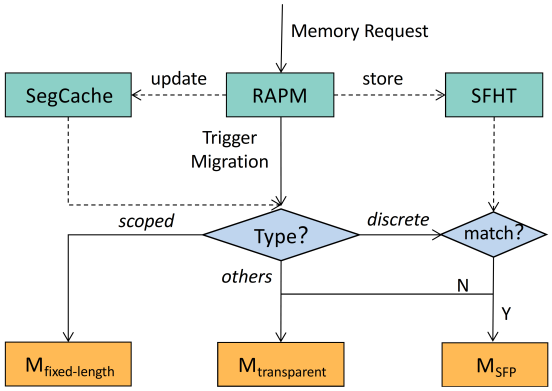


Fig. 3. The workflow of dynamic migration granularity policy.

Therefore, AdaptHM proposes a dynamic local migration granularity policy, whose core idea is to dynamically set migration granularity for memory regions based on their access patterns. To support variable and memory-region-customized migration granularity, AdaptHM decouples migration from the remapping structure as illustrated in Section III-B. AdaptHM tracks access patterns and adjusts migration granularity at segment-level memory regions. The dynamic local migration granularity relies on two core components: i) Runtime Access Pattern Monitor, ii) Granularity Controller. Below, we will introduce each component in detail.

1) Runtime Access Pattern Monitor: The goal of our runtime access pattern monitor (RAPM) is to identify useful access patterns that help to guide granularity adjustment. Therefore, RAPM simply classifies the access behaviors into two types: *scoped* and *discrete*, based on two intuitive rules: i) Large migration granularity benefits from streaming and/or scoped access patterns due to the prefetching effect, ii) Small granularity is suitable for random and/or discrete access patterns since it avoids unnecessary traffic and long tail latency. RAPM uses a bitmap to record accessed blocks for each opened (being accessed) segment. Then RAPM assesses the aggregation degree of accessed blocks according to the ratio of the number of accessed blocks (Access_N) and the access distance (Access_D, the maximum absolute difference of logical block number of accessed blocks). If the aggregation degree exceeds a threshold, the segment is under scoped access pattern. Otherwise, the segment is under discrete access pattern. The threshold is empirically set as 0.5. Considering concurrent access behaviors in hybrid memory, RAPM maintains a bitmap of the opened segments for each memory channel. The current opened segment is closed and the bitmap will be reset when another segment in the same channel is accessed.

2) Migration Granularity Controller: Migration granularity controller (MGC) determines the appropriate migration mode and granularity for segments. Corresponding to the access pattern classification of RAPM, MGC enables three migration modes: i) *Fixed-length Adaptive Mode*, ii) *Spatial Footprint Prediction Mode*, and iii) *Transparent Mode*.

- *Fixed-length Adaptive Mode*: A large migration granularity that covers all potentially accessed blocks maximizes prefetching benefits for scoped access patterns. Therefore, AdaptHM sets a fixed-length and continuous-space migration granularity for segments with scoped access patterns. Specifically, the maximum of (Access_N, Access_D) is set as the migration granularity to cover all possible accessed blocks.
- *Spatial Footprint Prediction Mode*: A small granularity is often better to avoid unnecessary migration traffic for discrete access patterns. However, small granularity leads to more software overhead and loss of the prefetching benefit. Therefore, AdaptHM enables Spatial Footprint Prediction (SFP) mode for potentially recurring discrete access patterns. The mechanism of SFP is that Program Counter (PC) and request address have a high correlation with the access patterns [35]. If a pair of PC and request address leads to an access pattern, then it is likely that a similar access pattern will repeat when the same PC and request address occurs. Therefore, AdaptHM stores the spatial footprints (i.e., the bitmaps of opened segments recorded by RAPM) into a Spatial Footprint History Table (SFHT, in NM). The XOR of the first accessed block address and the instruction PC is used as the index of SFHT. In general, the PC and address are encapsulated in the memory request and can be obtained directly by the HMC. Then, AdaptHM migrates blocks accordingly if a matching entry is found in SFHT for a segment with discrete access pattern. In this case, the migration granularity may be discontinuous. The collection period of the spatial footprint varies for each segment and occurs asynchronously based on their access patterns. It starts from the first access to the segment and ends when leaving the segment (i.e., access another segment in the same memory channel).
- *Transparent Mode*: For unclassified access patterns or discrete access patterns without matching entry in SFHT, AdaptHM switches to *transparent* mode and migrates at block granularity like in previous works. Besides, the default migration mode of any segment is *transparent*.

The workflow of segment-level dynamic migration granularity policy is summarized in Fig. 3. Except for classifying access patterns, RAPM is responsible for storing migration mode and granularity of a few recently accessed segments into the on-chip SegCache, which accelerates the judgement of the segment type. Overall, the MGC maximizes migration benefits and avoids unnecessary migrations by setting suitable migration mode and granularity for segments.

E. Benefit-based Migration Frequency Controller

Migration frequency is an overlooked design option in existing DMS and often tied to hot data identification policy. However, migration is valuable only if the migrated block receives sufficient accesses after migration to offset the migration overhead. Existing history-based heuristic hot block identification policies cannot provide such assurance, facing the risk that migration overhead outweighs the benefits. On the other hand, prior work that adjusts global migration frequency periodically is not aware of the regional difference of access patterns under concurrent environment [12].

To tackle the above challenges, AdaptHM proposes a benefit-based migration frequency control approach. As shown in Fig. 4, the workflow of migration frequency controller (MFC) consists of three main stages: *Statistics Collection*, *Benefit Assessment*, and *Frequency Adjustment*. Next, we introduce each stage in detail.

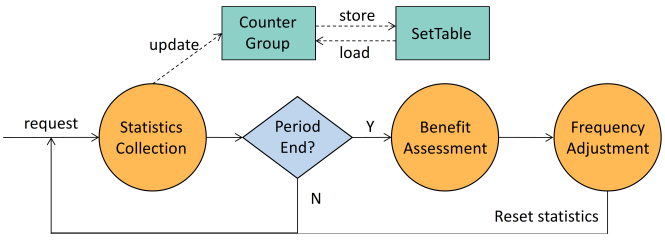


Fig. 4. The workflow of migration frequency controller.

1) Statistics Collection: To accurately assess the migration benefit during a period, MFC collects the following information for each set:

- N_{NM}: # of memory requests served by Near Memory during the period.
- N_{MG}: # of blocks migrated during the period.

All collected data is stored in the SetTable located in NM. To accelerate statistics updating, the MFC maintains counter groups on chip for sets that are currently being accessed. These counters are synchronized with the SetTable in the background based on LRU policy. Besides, the MFC allocates a period counter for each set to avoid synchronously updating all sets at the end of the period. The asynchronous period control method allows active sets to update their migration frequency quickly in response to variable access patterns. The period for each set is 10K memory accesses.

2) Benefit Assessment: At the end of the period, the MFC assesses the benefit using Equation 1 from [3].

$$B_{period} = N_{NM} \ / \ (\ K \times N_{MG} \ ) \tag{1}$$

K is the number of extra NM accesses required for a migrated block to compensate for the cost of a migration. K differs depending on the relative latency of NM and FM. In our configuration (see Table II), the cost of a single migration is about 784 cycles, [1] and the difference in access latency between NM and FM is about 65 cycles. [2] Thus, a migrated block needs to get 13 more NM accesses for migration to be valuable. K is computed in hardware at boot time. When B_{period} is greater than 1, AdaptHM gains migration benefit in the last period. Otherwise, the migration overhead outweighs benefits.

3) Frequency Adjustment: As introduced in Section III-C, the hot block within a CGroup is elected through *Winner-Score*, which is adjusted by *INC_Step* / *DEC_Step*. The MFC controls migration frequency by using differentiated incremental/decremental step. Specifically, a larger *INC_Step* means lower migration frequency since the current winner block is difficult to be replaced. Conversely, a larger *DEC_Step* means higher migration frequency. In each period, MFC can choose between two modes: encourage migration (*Mode*=1) or inhibit migration (*Mode*=0). After benefit assessment, MFC uses a simple hill-climbing algorithm [29] to adjust migration frequency. As shown in Algorithm 2, if the benefit is positive, the used mode in last period is beneficial. Thus, MFC adjusts step further in the direction that it was heading in. Otherwise, the used mode in last period leads to performance degradation. MFC changes the mode and resets steps. Overall, MFC adjusts migration frequency in the direction of increasing migration benefit and adapts to the variable access patterns. After the adjustment is complete, the MFC resets the statistics for the set.

Algorithm 2 Frequency Adjustment Policy

Input: The B_{period}, and adjustment $Mode$ of last period
if B_{period} is positive **then**
  if Encourage migration in last period ($Mode$=1) **then**
    Increase DEC_Step
  else
    #Inhibit migration in last period ($Mode$=0)
    Increase INC_Step
  end if
else
  if Inhibit migration in last period ($Mode$=0) **then**
    # Reset steps to encourage migration
    DEC_Step = 2
    INC_Step = 1
    $Mode$ = 1
  else
    # Encourage migration in last period ($Mode$=1)
    # Reset steps to inhibit migration
    DEC_Step = 1
    INC_Step = 2
    $Mode$ = 0
  end if
end if

F. Latency Optimization

As illustrated in Section III-B, AdaptHM uses a 256B block and a 16KB segment to manage remapping and migration, separately. Thus, both the remapping and migration metadata structures are likely to exceed megabyte level for GB-level memory capacity, which makes them unable to fit on chip. However, the remapping table is located in the critical path of memory access since it records the actual address. Putting remapping table in NM would require an extra memory access to fetch the remapping entry. Besides, dynamic migration granularity may add indeterminate tail latency to normal memory requests. Therefore, AdaptHM adopts the following techniques to optimize memory access latency.

1) Remapping Buffer: To avoid the extra metadata querying latency before each memory access, AdaptHM enables a small on-chip remapping buffer (RmpBuffer) to cache a few recently used remapping entries. Unlike the remapping cache used in prior works, the RmpBuffer fetches multiple continuous remapping entries to form a metadata cacheline (64B) at once and adopts the LRU replacement policy. The fetched metadata cacheline contains exactly all the remapping entries of a segment, which helps non-transparent-mode migration (i.e., fixed-length adaptive mode and SFP mode) (see Section III-G).

[1] An FM block access: (22 tRP + 22 tRCD + 22 tCAS + 4 * 8 bursts) * 2 (clock ratio of CPU to FM) = 196 cycles. A three-block migration process requires four of these (Section III-F2).

[2] Access latency difference: (22 tRP + 22 tRCD + 22 tCAS) * (3.2GHz / 1.6GHz) - (7 tRP + 7 tRCD + 7 tCAS) * (3.2GHz / 1GHz) = 64.8 cycles.

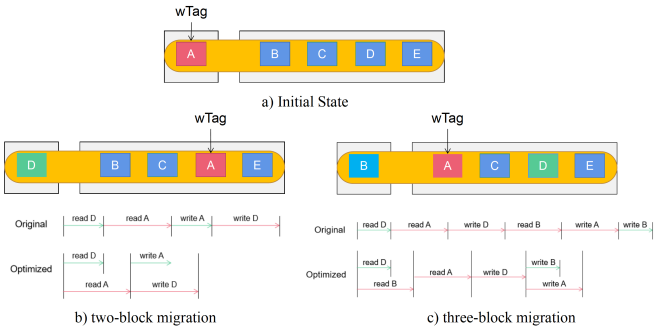


Fig. 5. The migration types and optimizations.

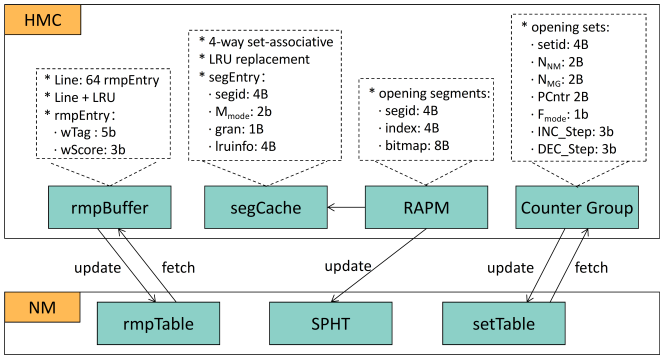


Fig. 6. The metadata structures and organization.

In case the RmpBuffer misses, AdaptHM adopts the parallel prefetching technique from RHPM [14] to overlap the metadata querying latency. The core idea is to fetch metadata and data in parallel when the RmpBuffer misses. If the prefetched data is exactly what the request requires, metadata querying latency is hidden. Although the actual address is unknown before we get the remapping entry, the actual data has only two possible locations since AdaptHM adopts the CGroup organization with the fixed-remapping manner. Considering that NM always has greater access possibility, AdaptHM chooses to prefetch the data cacheline in the NM block within the accessed CGroup and the metadata cacheline in parallel.

2) Migration Buffer: In transparent migration mode, a single migration may involve two or three blocks. Fig. 5a shows an initial state in a CGroup where block A is the winner, while other blocks are unchanged. When block D becomes the new winner (Fig. 5b), the migration process only involves two different blocks (FM block D and NM block A). However, if block B becomes the new winner after block A and block D complete the migration (Fig. 5c), the migration process will involve three different blocks (block A, B, D). Since we only maintain one wTag that points to the original position of the current winner (refer to Section III-G1), the positions of other blocks must remain unchanged. If the migration process is performed serially, all memory access delays are exposed on the critical path, resulting in significant tail latency. To shrink the migration latency, AdaptHM enables a small *Migration Buffer* in the HMC (see Fig. 1) to exploit the bus level parallelism between FM and NM. When migration is triggered, the HMC simultaneously initiate parallel read requests to corresponding NM and FM blocks. Once the required blocks reach the Migration Buffer, write requests are synchronously dispatched. Consequently, the access latency of NM is overlapped, with only FM accesses visible on the critical path (as shown in Fig. 5). In our implementation, we simulated 2 FM accesses for two-block migration and 4 FM accesses for three-block migration. In addition, AdaptHM preferentially migrates the accessed block to deal with the indeterminate tail latency in non-transparent migration mode. With this approach, the memory request can be completed early once the accessed block is fetched into the migration buffer. Moreover, the rest blocks of the migration caused by dynamic migration granularity controller leaves the critical path of memory request.

3) Counter Group: The on-chip counter group keeps useful information for the core components of AdaptHM, such as *runtime access pattern monitor*, *migration granularity controller* , and *migration frequency controller*. Maintaining these counters incurs considerable operation overhead. Therefore, AdaptHM delays updating these counters until the request is finished. Overall, the updating of counter group is out of the critical path of memory access and is independent of the migration procedure.

G. Metadata Optimization

To achieve comprehensive access pattern adaptability on multiple aspects, AdaptHM has to maintain plenty of metadata to support its core components. Fig. 6 shows the metadata structures and organization required by AdaptHM. As explained in Section III-F, metadata structures are deployed in both on-chip HMC and NM to accelerate querying and updating. However, the two-level metadata organization leads to higher on-chip hardware overhead and lower available NM capacity. To shrink the metadata space, AdaptHM adopts the following optimizations.

1) Optimization for Remapping Structure: The basic unit of remapping structure is the remapping entry (rmpEntry). Due to the CGroup with fixed remapping, rmpEntry only requires two fields: winner tag (wTag: 5 bits) and winner score (wScore: 3 bits) . The wTag is used to mark the original position of the winner block. A 5-bit wTag supports FM capacity expansion up to 31x the NM capacity, which is enough to verify the capacity scalability of AdaptHM. The 3-bit wScore is designed as a saturation counter. In the default configuration (capacity ratio of NM:FM=1:8), the small wScore counter is enough to capture the changes of access patterns. Eventually, AdaptHM only requires 1B remapping metadata for each CGroup, significantly less than prior works.

2) Optimization for Granularity Controller: To support flexible migration mode and granularity, the Granularity Controller collects useful metadata. However, AdaptHM does not keep granularity information for all segments due to timeliness of access patterns and the considerable metadata overhead. Instead, AdaptHM stores the granularity information of a few recently accessed fixed-length-mode segments into the on-chip segment cache (SegCache). On the other hand, useful access patterns are stored in the Spatial Footprint History Table (SFHT) on NM. Each SFHT entry contains an index and a bitmap. To accelerate querying, SFHT is a hash table with 10K entries. When a collision occurs, SFHT replaces the old

entry with the new one. The random replacement policy is adopted when SFHT is full.

3) Optimization for Frequency Controller: AdaptHM periodically assesses and adjusts the migration frequency of sets. The corresponding statistics are stored in the SetTable on NM. However, the SetTable faces the same latency problem as the rmpTable. To accelerate metadata updating, AdaptHM maintains counter groups for opened sets (i.e., those are currently being accessed). In the default configuration, 64 sets are kept opening and an LRU policy is used to update these counter groups. When a new set is accessed, AdaptHM retrieves statistics from the SetTable and flushes the statistics of the evicted set back into the SetTable. Although a single set entry is larger than the rmpEntry, the number of sets is only 1/16 of CGroups.

H. Cost Analysis and Limitations

The hardware cost of AdaptHM contains two main parts: on-chip capacity and NM capacity. The former includes: 1) a 1KB RmpBuffer, 2) a 1KB migration buffer, 3) a 4KB SegCache, 4) a 1KB RAPM, and 5) a 1KB Counter Group. The capacities of RmpBuffer and SegCache are set based on the sensitive tests in Section V-G, while the others are calculated according to the metadata requirements. Overall, AdaptHM requires a total on-chip capacity of approximately 8KB. Compared to the POM (32KB on-chip SRC) [3] and HYRBDI2 (512KB on-chip XTA) [10], the hardware cost of AdaptHM is significantly reduced. On the other hand, metadata structures located in NM occupy a portion of available memory capacity for applications. However, in the default capacity ratio configuration (NM:FM=1:8), the total space required by rmpTable and SetTable is only about 0.49% of the NM capacity. Additionally, the 120KB SFHT is negligible relative to the GB-level memory capacity. Overall, the hardware cost of AdaptHM is acceptable when compared to previous works.

Although AdaptHM achieves relatively comprehensive access pattern adaptability, it still has several limitations in design. Since it adopts the NM/FM abstraction as the basic assumption for hybrid memory, AdaptHM does not consider specific memory technology characteristics such as read/write asymmetry and write durability issues of NVMs. Many works have been devoted to addressing these issues [30], [36] and AdaptHM is orthogonal to them. Furthermore, AdaptHM is unsuitable for situations where NM is much smaller than FM (e.g., the ratio < 1:16) due to CGroup competition. Additionally, the main design considerations and assumptions are based on general-purpose computing systems and may be unsuitable for special systems that do not need to consider trade-offs between performance, cost, and capacity (e.g., supercomputers). Distributed computing environment and disaggregated memory architecture are also beyond the scope of this paper and are likely to be the future work of AdaptHM.

IV. Experimental Setup

In this section, we provide the details of the experimental setup and the benchmarks used for our evaluation.

TABLE II
SIMULATION CONFIGURATION

Cores	4 @3.2GHz(each), X86 ISA, out-of-order
L1(I/D)	32KB, 4-way associative, 64B cacheline
L2	256KB(private), 8-way associative, 64B cacheline
LLC	8MB(shared), 16-way associative, MESI, 64B cacheline
Near Memory	HBM2 2GHz, 2GB, 8 128-bit channels, 8 banks tCAS-tRCD-tRP:7-7-7 RD/WR+I/O energy: 6.4pJ/bit ACT/PRE energy:15nJ
Far Memory	DDR4-3200, 16GB, 2 64-bit channels, 8 banks tCAS-tRCD-tRP: 22-22-22 RD/WR+I/O energy: 33pJ/bit ACT/PRE energy:15nJ

TABLE III
WORKLOAD CHARACTERISTICS.

Suite	Workload (×4)	MPKI (single)	Suite	Workload (×4)	MPKI (single)
CORAL	xsbench	2.10	Mantevo	pathfinder	91.45
MiBench	susan	0.02	SPEC	wrf	12.62
	gsm	3.22		milc	11.64
	basicmath	13.89		sjeng	13.90
Mantevo	minixyce	0.01		bzip2	15.25
	minighost	0.78		cactusADM	21.32
	minife	6.82		leslie3d	31.09
	hpccg	10.64		mcf	49.56
Mix-1	gsm leslie3d hpccg cactusADM	N/A	Mix-3	minighost susan libquantum mcf	N/A
Mix-2	minixyce milc basicmath xsbench	N/A	Mix-4	minife bzip2 wrf sjeng	N/A

Simulator: We simulate AdaptHM in the full-system simulator GEM5 [37]. The default capacity ratio of NM to FM is set to 1:8. Table II summarizes our simulated configuration which adopts the same memory parameters as RHPM [14]. The energy model is the same as [38]. We simulated 8 cycles for a single access latency of the on-chip metadata structures in HMC, the same as our L2 cache.

Workloads: We simulated 20 workloads from four representative benchmark suites: SPEC2006 [39], Mantevo [40], MiBench [41], and CORAL [42], whose characteristics are shown in Table III. Each workload contains four applications (single or mixed). Our workloads run in a multi-programmed simulation mode, where each application instance runs 200 million instructions on a single core with warmed-up caches.

Configurations: We compare AdaptHM to four state-of-the-art data migration strategies. These are:

- **POM** [3]. POM adopts a threshold-based migration policy with 2KB remapping/migration granularity. The global threshold is periodically adjusted by sampling different areas.
- **SILC** [5]. SILC adopts threshold policy to decide a 2KB remapping page and on-demand migration with 64B subblocks. The migration granularity is variable in a migrated page without prefetching effect.
- **HYBRID2** [10]. HYBRID2 takes part of NM as a cache

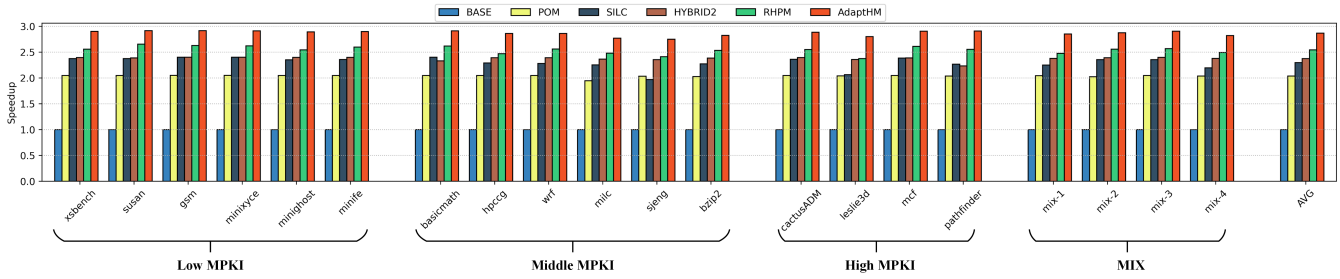
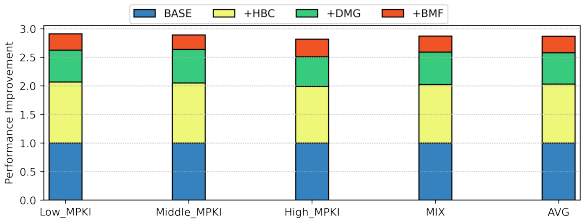


Fig. 7. Performance comparison with other schemes



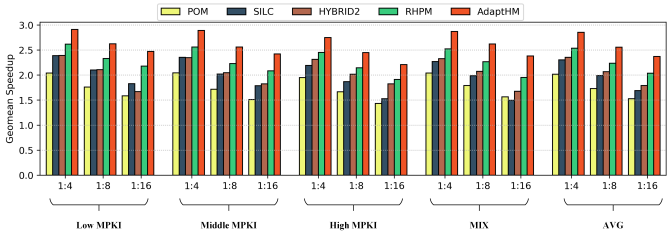


Fig. 8. Performance improvement breakdown

Fig. 9. Geometric speedup with various capacity ratios

for the rest of hybrid memory. The NM cache is remapped on demand and evicted by LRU. The migration granularity is set to 256B according to design space exploration.

- **RHPM** [14]. RHPM enables a relative hotness concept to find hot pages without relying on thresholds. It adopts 256B migration granularity based on sensitive tests.

The baseline configuration is a special type of hybrid memory where both memories are configured as FM. There is also an HMC responsible for forwarding LLC misses to the correct memory device without data migration strategy.

V. Evaluation

In this Section, we present the evaluation of AdaptHM under the default capacity ratio of 1:8. Unless otherwise specified, all results in the following subsections are normalized with the baseline configuration.

A. Overall Performance

Fig. 7 shows the geometric mean speedup for all workloads over the baseline configuration. The workloads are sorted by their average LLC Misses Per Kilo Instruction (MPKI, shown in Table III). Overall, AdaptHM achieves on average 40.73%, 24.8%, 20.71%, and 12.78% better speedup than POM [3], SILC [5], HYBRID2 [10], and RHPM [14], respectively.

POM obtains an average 2.04x speedup than the baseline by the threshold-based hot page identification policy. However, the long threshold adjustment period (10K LLC misses) makes POM respond slowly to access pattern changes. The 2KB fixed global migration granularity of POM lacks flexibility to cope with variable access patterns and results in sub-optimal performance. Moreover, POM does not consider the impact of migration frequency.

SILC adopts 2KB remapping pages with 64B sub-blocks, enabling on-demand migration in a remapped page. However, the migration granularity is still fixed and cannot offer prefetching benefits. Although the epoch-based threshold adjustment policy indirectly impacts on migration frequency, the 1-million-memory-access epoch is too long to follow access pattern changes. Moreover, the global epoch control causes long tail latency at the end of an epoch due to synchronous updating.

HYBRID2 takes part of NM as a cache of the flat address space, exploiting advantages of both PoM and cache architecture. The NM cache sectors are dynamically remapped and are migrated on demand. Besides, HYBRID2 determines to migrate a sector to NM based on its cost function, which also indirectly impacts migration frequency. However, it still adopts 256B fixed migration granularity and consumes more time on metadata operation than other schemes due to its all-to-all remapping supported by two remapping tables.

RHPM tracks the hot pages based on the proposed relative hotness concept without relying on thresholds. The hot page identification policy of RHPM shows better adaptability to access patterns than prior works. However, RHPM requires per-page metadata and also adopts fixed 256B migration granularity. Moreover, RHPM does not consider the impact of migration frequency either.

Compared to them, AdaptHM comprehensively considers the adaptability to access patterns among hot block identification, migration granularity, and migration frequency. The hot block competition policy of AdaptHM does not rely on thresholds and shows a quick response to access pattern changes like RHPM. However, unlike RHPM, AdaptHM does not require per-block metadata. The metadata overhead is effectively reduced on both space and operation aspects. The migration granularity of AdaptHM can be dynamically set at segment level based on the access patterns, offering better flexibility and adaptability. Finally, AdaptHM adjusts the migration frequency at set level based on the periodical benefit assessment. The asynchronous period control avoids long tail latency at the end of period and offers adaptability to different sets. Therefore, AdaptHM achieves the best performance on evaluated workloads.

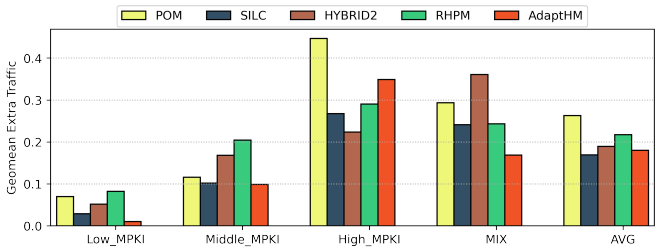


Fig. 10. Geometric mean of extra traffic

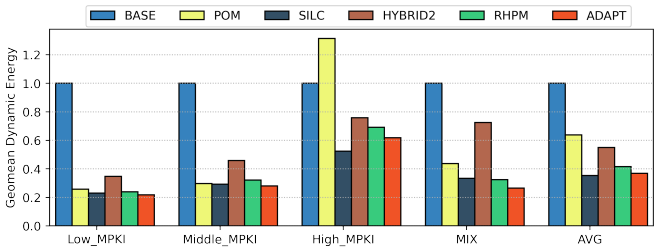


Fig. 11. Geometric mean of dynamic memory energy consumption

B. Performance breakdown

We show the breakdown of AdaptHM performance improvement in Fig. 8. The stacked bar begins with the baseline without migration. We show performance improvement achieved by each component of AdaptHM. HBC shows the performance improvement achieved with only the hot block competition (HBC) policy. HBC achieves on average 2.03x speedup on all workloads compared to the baseline. It shows a uniform distribution on different MPKI classes since the HBC identifies hot blocks dynamically with access pattern changes. DMG adds the dynamic segment-level migration granularity based on HBC, further improving the adaptability to access patterns on different memory regions. DMG achieves an average 1.27x speedup on all workloads than the HBC. The top stack bar is BMF, which adds the benefit-based migration frequency policy based on DMG. BMF obtains on average 1.11x speedup than the DMG. There are two reasons for the gradual decline in speedup: 1) upper components (BMF, DMG) rely on the underlying component (HBC), 2) HBC directly obtains the benefits from both locality and adaptability to access patterns.

C. Capacity Analysis

In this section, we validate the scalability of AdaptHM and analyze the performance improvement with different NM to FM capacity ratios. We vary the NM:FM capacity ratio from 1:4 to 1:16. The results are normalized with the baseline. Fig. 9 shows the performance improvement on all MPKI classes with various schemes and capacity ratios. For low MPKI benchmarks, SILC performs slightly better than HYBRID2 since the locking page feature of SILC benefits from locality. RHPM and AdaptHM show resilience on capacity ratios and perform better than others. Overall, a larger NM capacity share leads to better performance due to two reasons: 1) more data can be migrated into NM, 2) less competition within a CGroup.

D. Traffic Analysis

Fig. 10 shows the geometric mean of extra traffic percentage in total traffic for all benchmark groups. The extra traffic is generated by memory accesses that are not initially issued by applications. For low MPKI benchmarks, all schemes cause the minimum extra traffic since they benefit from the locality of workloads. With the locality decreasing, more extra traffic is introduced due to migration. POM results in 44.67% of extra traffic on high MPKI benchmarks due to its 2KB large migration granularity. When locality is unavailable, POM migrates many unnecessary pages that will not be accessed after migration. Thanks to the on-demand migration feature, SILC leads to the least average extra traffic. Overall, AdaptHM generates 18% extra traffic on average, which is 17.32% less than that of RHPM. The reduction comes from two aspects: 1) AdaptHM simplifies remapping structure and thus reduces metadata access, 2) the migration frequency controller helps to reduce unnecessary migration traffic.

E. Energy Consumption

Fig. 11 shows the geometric mean of the dynamic memory system energy consumption normalized to the baseline. The baseline configuration consumes the most energy since it leads to more FM access without migration. POM consumes more dynamic energy than the baseline on high MPKI workloads since it causes large extra traffic (see Fig. 10). Overall, SILC leads to the least average energy consumption due to its on-demand fine-grained migration and locking page feature. The average energy consumption of AdaptHM is only 36.88% of the baseline, which is slightly higher than SILC and lower than others. The distribution of energy consumption is roughly consistent with the extra traffic.

F. Migration Overhead

We analyze the migration overhead of various schemes from both traffic and time caused by migration. Fig. 12 shows the geometric mean of the migration traffic normalized to POM for each benchmark group. Similar to Fig. 10, POM causes the most average migration traffic on all benchmark groups. AdaptHM generates the least average migration traffic, which is 44.45% less than that of POM. The result indicates that metadata access of AdaptHM accounts for a significant part in the extra traffic. After all, AdaptHM requires more metadata structures than other schemes to support its comprehensive adaptability.

On the other hand, Fig. 13 shows the geometric mean of the migration time for various schemes. Although POM results in the most migration traffic, it has the lowest migration time. The results are not contradictory due to its 2KB migration granularity. Thanks to the large migration granularity, POM requires far fewer migration counts than other schemes to migrate the same total amount of data. For example, POM only requires 8 migrations to migrate 16KB data (2KB/migration), while SILC requires 256 migrations (64B/migration). Conversely, SILC causes the most migration time due to its 64B migration granularity. The software overhead caused by the large number

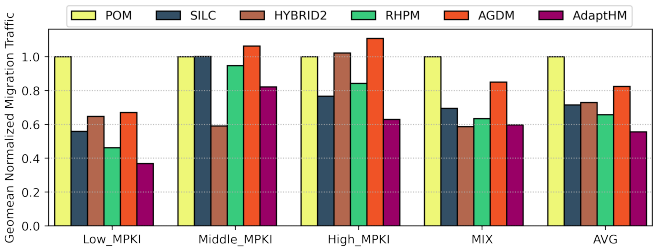


Fig. 12. Geometric mean of migration traffic

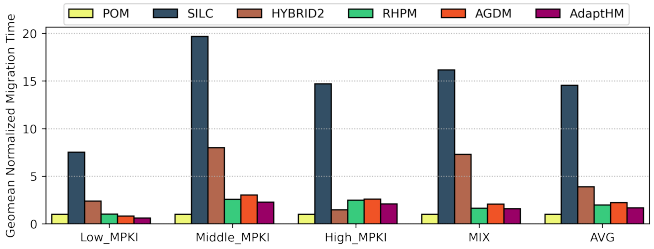


Fig. 13. Geometric mean of migration time

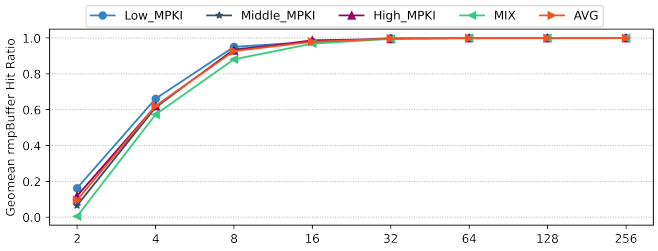


Fig. 14. Geometric RmpBuffer hit ratio

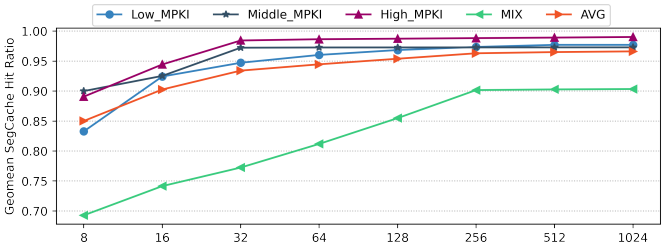


Fig. 15. Geometric SegCache hit ratio

of migrations contributes to its migration time. Overall, the migration time of AdaptHM is only slightly higher than POM and lower than others. There are two reasons: 1) the dynamic migration granularity of AdaptHM helps to reduce software overhead, 2) the migration frequency controller effectively reduces unnecessary migration.

Furthermore, in the exploration on migration overhead, we additionally compared our previous work, AGDM [15]. While both schemes share various aspects, the primary distinction lies in the introduction of the Benefit-based Migration Frequency Controller module (BMF) in AdaptHM. BMF's principal objective is to prevent heuristic policies from causing excessive migrations. Fig. 12 shows that AdaptHM reduces average migration traffic by 32.57% compared to AGDM. Meanwhile, Fig. 13 shows that AdaptHM reduces average migration time by 24.34%. These results clearly indicate that the BMD module effectively manages migration overhead. However, it's important to note that while reducing migration traffic, BMF also lead to some loss in migration benefits. Consequently, the overall performance of AdaptHM and AGDM remains quite similar.

G. Sensitivity Analysis

In this section, we perform sensitivity studies on two on-chip metadata structures: RmpBuffer and SegCache. They are critical components for AdaptHM to optimize latency and metadata operations. Therefore, their hit ratios directly impact performance. Our goal is to seek a compromise between hardware capacity cost and the hit ratio.

1) RmpBuffer Capacity: We vary the RmpBuffer capacity from 2 cachelines to 256 cachelines. Fig. 14 shows the geometric mean of RmpBuffer hit ratio over different capacities. The hit ratio increases with the capacity increasing. However, the growth trend shows a turning point at 16 cachelines, where the RmpBuffer achieves an average hit ratio of 96.84%. There is little gain from continuing to increase capacity after the turning point. Therefore, AdaptHM sets the default RmpBuffer capacity as 1KB (16 cachelines), considering both performance and hardware cost.

2) SegCache Capacity: Fig. 15 shows the results of SegCache. We vary the SegCache capacity from 8 entries to 1024 entries. Similar to Fig. 14, the growth trend of the SegCache hit ratio also shows a turning point at 256 entries, where it achieves an average hit ratio of 95.39%. A larger capacity can only bring a slight increase in the hit ratio. Therefore, the SegCache capacity is set as 4KB (16B/entry) in default.

VI. Conclusion

Prior data migration strategies for hybrid memory systems only focused on one aspect of migration, leaving others to be handled with simplified methods. Meanwhile, they ignored the importance of access patterns, resulting in sub-optimal performance and unnecessary migration. In this work, we present AdaptHM, a comprehensive adaptive data migration strategy that enables access-pattern adaptability on hot data identification, migration granularity and migration frequency. AdaptHM proposes a novel multi-level data framework to support its adaptability at different-sized memory regions. The hot block competition policy removes the necessity of per-block metadata and reduces operation overhead. The migration granularity can be dynamically set for segments based on their access patterns, maximizing the prefetching benefits. Migration frequency can be adjusted by the periodical benefit assessment, avoiding unnecessary migration overhead. We use four widely-used benchmark suites to demonstrate the adaptability of AdaptHM. The evaluation shows that, compared to RHPM, one of the state-of-the-art schemes, AdaptHM improves performance by 12.78% and reduces migration traffic by 15.44% on average. The dynamic memory energy consumption is saved by up to 37.24%.

VII. Acknowledgments

The authors thank anonymous reviewers for their meticulous reviews and insightful suggestions. This work was supported by the National Natural Science Foundation of China (No.61821003 and No.61832007), and Engineer Research Center of Data Storage Systems and Technology, MoE, China.

References

[1] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 24–33, 2009.

[2] Chia Chen Chou, Aamer Jaleel, and Moinuddin K Qureshi. Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1–12. IEEE, 2014.

[3] Jaewoong Sim, Alaa R Alameldeen, Zeshan Chishti, Chris Wilkerson, and Hyesoon Kim. Transparent hardware management of stacked dram as part of memory. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 13–24. IEEE, 2014.

[4] Mitesh R Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 126–136. IEEE, 2015.

[5] Jee Ho Ryoo, Mitesh R Meswani, Andreas Prodromou, and Lizy K John. Silc-fm: Subblocked interleaved cache-like flat memory organization. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 349–360. IEEE, 2017.

[6] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M Tullsen. Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 433–444. IEEE, 2017.

[7] Jagadish B Kotra, Haibo Zhang, Alaa R Alameldeen, Chris Wilkerson, and Mahmut T Kandemir. Chameleon: A dynamically reconfigurable heterogeneous memory system. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 533–545. IEEE, 2018.

[8] Apostolos Kokolis, Dimitrios Skarlatos, and Josep Torrellas. Pageseer: Using page walks to trigger page swaps in hybrid memory systems. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 596–608. IEEE, 2019.

[9] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. Llc-guided data migration in hybrid memory systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 932–942. IEEE, 2019.

[10] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. Hybrid2: Combining caching and migration in hybrid memory systems. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 649–662. IEEE, 2020.

[11] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 392–407, 2021.

[12] Shashank Adavally, Mahzabeen Islam, and Krishna Kavi. Dynamically adapting page migration policies based on applications' memory access behaviors. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(2):1–24, 2021.

[13] Bokyeong Kim, Soojin Hwang, Sanghoon Cha, Chang Hyun Park, Jongse Park, and Jaehyuk Huh. Supporting dynamic translation granularity for hybrid memory systems. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 25–32. IEEE, 2022.

[14] Zhouxuan Peng, Dan Feng, Jianxi Chen, Jing Hu, and Chuang Huang. Rhpm: Using relative hotness to guide page migration for hybrid memory systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[15] Zhouxuan Peng, Dan Feng, Jianxi Chen, Jing Hu, and Chuang Huang. Agdm: An adaptive granularity data migration strategy for hybrid memory systems. In *2023 Design, Automation and Test in Europe Conference (DATE)*. IEEE, 2023.

[16] Gabriel H Loh and Mark D Hill. Efficiently enabling conventional block sizes for very large die-stacked dram caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 454–464, 2011.

[17] Moinuddin K Qureshi and Gabe H Loh. Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 235–246. IEEE, 2012.

[18] Cheng-Chieh Huang and Vijay Nagarajan. Atcache: Reducing dram cache latency via a small sram tag cache. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 51–60, 2014.

[19] Yongjun Lee, Jongwon Kim, Hakbeom Jang, Hyunggyun Yang, Jangwoo Kim, Jinkyu Jeong, and Jae W Lee. A fully associative, tagless dram cache. *ACM SIGARCH Computer Architecture News*, 43(3S):211–222, 2015.

[20] Chiachen Chou, Aamer Jaleel, and Moinuddin K Qureshi. Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches. *ACM SIGARCH Computer Architecture News*, 43(3S):198–210, 2015.

[21] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas. Banshee: Bandwidth-efficient dram caching via software/hardware cooperation. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–14. IEEE, 2017.

[22] Chiachen Chou, Aamer Jaleel, and Moinuddin Qureshi. Batman: Techniques for maximizing system bandwidth of memory systems with stacked-dram. In *Proceedings of the International Symposium on Memory Systems*, pages 268–280, 2017.

[23] Evangelos Vasilakis, Vassilis Papaefstathiou, Pedro Trancoso, and Ioannis Sourdis. Decoupled fused cache: Fusing a decoupled llc with a dram cache. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4):1–23, 2019.

[24] Mark Oskin and Gabriel H Loh. A software-managed approach to die-stacked dram. In *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pages 188–200. IEEE, 2015.

[25] Neha Agarwal and Thomas F Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 631–644, 2017.

[26] Yang Li, Saugata Ghose, Jongmoo Choi, Jin Sun, Hui Wang, and Onur Mutlu. Utility-based hybrid memory management. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 152–165. IEEE, 2017.

[27] Jee Ho Ryoo, Lizy K John, and Arkaprava Basu. A case for granularity aware page migration. In *Proceedings of the 2018 International Conference on Supercomputing*, pages 352–362, 2018.

[28] Mahzabeen Islam, Shashank Adavally, Marko Scrbak, and Krishna Kavi. On-the-fly page migration and address reconciliation for heterogeneous memory systems. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 16(1):1–27, 2020.

[29] HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A Harding, and Onur Mutlu. Row buffer locality aware caching policies for hybrid memories. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pages 337–344. IEEE, 2012.

[30] Hoda Aghaei Khouzani, Chengmo Yang, and Jingtong Hu. Improving performance and lifetime of dram-pcm hybrid main memory through a proactive page allocation strategy. In *The 20th Asia and South Pacific Design Automation Conference*, pages 508–513. IEEE, 2015.

[31] Moinuddin K Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran. Phase change memory: From devices to systems. *Synthesis Lectures on Computer Architecture*, 6(4):1–134, 2011.

[32] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. Platform storage performance with 3d xpoint technology. *Proceedings of the IEEE*, 105(9):1822–1833, 2017.

[33] Joe Jeddeloh and Brent Keeth. Hybrid memory cube new dram architecture increases density and performance. In *2012 symposium on VLSI technology (VLSIT)*, pages 87–88. IEEE, 2012.

[34] JEDEC Standard. High bandwidth memory (hbm) dram. *Jesd235*, 2013.

[35] Chi F Chen, S-H Yang, Babak Falsafi, and Andreas Moshovos. Accurate and complexity-effective spatial pattern prediction. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 276–287. IEEE, 2004.

[36] Vaibhav Gogte, William Wang, Stephan Diestelhorst, Aasheesh Kolli, Peter M Chen, Satish Narayanasamy, and Thomas F Wenisch. Software

wear management for persistent memories. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, pages 45–63, 2019.

[37] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[38] Clinton W Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R Stan. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 50–61. IEEE, 2011.

[39] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[40] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 3, 2009.

[41] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pages 3–14. IEEE, 2001.

[42] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis. *The Role of Reactor Physics toward a Sustainable Future (PHYSOR)*, 2014.

Zhouxuan Peng received the BE degree in computer science and technology from Huazhong University of Science and Technology (HUST), China, in 2018. He is currently working towards the PhD degree majoring in computer system architecture at HUST. His current research interests include hybrid/heterogeneous memory systems and low-power embedded memory systems.

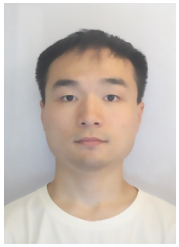
Dan Feng (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively. She is a Professor and the Dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, Non-Volatile memory technology, distributed and parallel file system, and massive storage system. She has more than 200 publications in major journals and international conferences, including IEEE TC, IEEE TPDS, IEEE TCAD, ACM-TOS, FAST, USENIX ATC, ISCA, EuroSys, HPDC, SC, ICS, DAC and DATE. She has served as the reviewer of multiple journals, including IEEE TC, IEEE TPDS, et al, and the program committees of multiple international conferences, including FAST 2022, SC 2011 & 2013, MSST 2012 &2015, SRDS 2020, et al. She is a member of IEEE and ACM (the Association for Computing Machinery).

Jianxi Chen (Member, IEEE) received the BE degree from Nanjing University, China, in 1999, the MS degree in computer architecture from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2002 and the PhD degree in computer architecture from HUST, in 2006. He is currently an associate professor with HUST, China. His research interests include computer architecture and massive storage systems. He published more than 20 papers in major journals and conferences. He is a member of the IEEE and CCF.

Jing Hu received the BE degree in computer science and technology from China University of Geosciences (Wuhan), china, in 2017. He is pursuing a Ph.D. at Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST). His current research interests include non-volatile memory technologies, key-value stores, and parallel and distributed systems.

Yachun Liu obtained her BE degree in network engineering from Henan University (Kaifeng), China, in 2015. She is currently pursuing a PhD at Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST). Her research interests include storage systems, big data, and resource scheduling.

Jinlei Hu received the BE degree from University of Electronic Science and Technology of China in 2020. He is pursuing a Ph.D. at School of Computer Science and Technology, Huazhong University of Science and Technology (HUST). His current research interests include persistent memory technologies, high-performance index, and Zone-namespaced storage.



Jintong Zhang received the BE degree in network engineering from Southwest University (Chongqing), china, in 2021. He is pursuing a Ph.D. at Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST). His current research interests include non-volatile memory technologies, ZNS storage technologies, and hybrid storage systems.

Tianyu Wan received the BE degree of Wuhan University of Technology, in 2022. He is pursuing a Ph.D. at Wuhan National Laboratory for Optoelectronics (WNLO), Huazhong University of Science and Technology(HUST). His current research interests includes disaggregated memory technologies and hybrid storage systems.

Zuoning Chen received the PhD degree from the Department of Computer Application Technology, Zhejiang University, China. She is currently with the National Research Centre of Parallel Computer Engineering and Technology. Her research interests include high-performance computer system architecture and operating system. She is also a member of the Chinese Academy of Engineering.