Phoenix: A Dynamically Reconfigurable Hybrid Memory System Combining Caching and Migration

Yifan Hua, *Student Member, IEEE*, Shengan Zheng, Weihan Kong, Cong Zhou, and Linpeng Huang, *Senior Member, IEEE*

***Abstract*—With the growing memory requirements of modern data-intensive applications for high performance and large capacity, building hybrid memory systems with different memory technologies has become a dominant trend to satisfy these demands. For better system performance, frequently accessed hot data is fetched into the fast and capacity-limited near memory (NM) while cold data is evicted to the slow and large far memory (FM). In prior works, NM is used as a cache of FM (cNM), part of OS-visible memory (mNM), and a fixed capacity of cNM and mNM. This article presents Phoenix, a novel hybrid memory architecture that harnesses the advantages of both cNM and mNM. The ratio of cNM to mNM is adjustable during runtime to better exploit both temporal and spatial locality benefits for different memory access patterns. All cNM and mNM space is multiplexed to mitigate the data movement overhead for the mode switch between cNM and mNM. In our evaluations, Phoenix outperforms state-of-the-art designs by an average of 18.2% and consumes orders of magnitude less metadata storage space.**

***Index Terms*—Cache mode, data migration policy, hybrid memory system, memory mode, spatial and temporal locality.**

I. Introduction

MANY modern big-data applications have vast datasets that dwarf capacity-limited SRAM caches, resulting in excessive cache miss requests and severe bandwidth pressure to off-chip DRAM modules [1]. Meanwhile, memory capacity has become scarce for data-intensive applications [6], [58]. However, the widely used memory technology, DRAM, suffers from device scalability problems [57]. Consequently, DRAM is unable to meet the growing demand for either memory bandwidth or capacity.

Manuscript received 12 January 2024; revised 14 May 2024; accepted 2 September 2024. Date of publication 5 September 2024; date of current version 21 February 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500303; in part by the National Natural Science Foundation of China (NSFC) under Grant 62227809 and Grant 62302290; in part by the Fundamental Research Funds for the Central Universities; in part by the Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0102; and in part by the Natural Science Foundation of Shanghai under Grant 22ZR1435400. This article was recommended by Associate Editor M. Zapater. *(Corresponding authors: Shengan Zheng; Linpeng Huang.)*

Yifan Hua, Weihan Kong, Cong Zhou, and Linpeng Huang are with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: huahuahuahua@sjtu.edu.cn; weihan@sjtu.edu.cn; cong258258@sjtu.edu.cn; lphuang@sjtu.edu.cn).

Shengan Zheng is with the School of Electronic Information and Electrical Engineering and the MoE Key Laboratory of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: shengan@sjtu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2024.3455237

Various memory technologies have been developed to meet the memory requirements in terms of capacity, bandwidth, access latency, and cost for modern applications. The emergence of persistent memory (PM) [43], [44], [45] and compute express link (CXL) [46], [55], [56] technologies exhibit the potential to fulfill the demand for memory capacity at a lower cost. Besides, the advancement of die-stacked technologies have given rise to high-bandwidth memories, such as hybrid memory cube (HMC) [36] and high-bandwidth memory (HBM) [34]. Unfortunately, none of these technologies can independently meet the diverse memory demands across various application domains. Consequently, the construction of hybrid memory systems utilizing different memory technologies has become the mainstream focus of recent research [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27].

In general, the hybrid memory system comprises different types of memory technologies: one has a limited capacity with better performance (higher bandwidth or lower latency), and another has a larger capacity but relatively poorer performance. For instance, hybrid memory systems [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27] consist of on-chip HBM and off-chip DRAM, local memory and remote memory connected by CXL, or DRAM and PM. To avoid constraints imposed by specific memory technologies in subsequent discussions, we call the former near memory (NM) and the latter far memory (FM) as in previous works [5], [6], [23]. Recent works have employed NM as a cache of FM (cNM) [10], [11], [12], [20], part of OS-visible memory (mNM) [3], [7], [8], [9], and both cNM and mNM (hybrid mode) [23], [28]. cNM designs react fast to hotness changes by fetching all requested data from FM to NM, but take the NM capacity away from the memory system and have a bad performance for workloads with weak temporal locality. mNM designs enhance the OS-visible memory capacity and bandwidth efficiency, but make the migration decision slower since only data with potential for future reuse is migrated. Hybrid mode designs aim to combine the advantages of cNM and mNM designs.

Unfortunately, existing hybrid mode designs [23], [28] suffer from four limitations.

1) Incapability of supporting an adjustable ratio of cNM to mNM during runtime. Fixed cNM and mNM capacities lack the flexibility to match different memory access patterns.

2) Unnecessary data migration overhead for mode switch between cNM and mNM stemming from the separate cNM and mNM space. For example, for evicting a page from cNM to mNM, a victim page in mNM is swapped out to FM, and then cached for the subsequent access, which brings unnecessary migration cost.
3) Inability to efficiently swap data between hybrid memories with lightweight data remapping overhead. Data swap between mNM and FM either sacrifices the swap efficiency for less remapping metadata or requires a large amount of metadata to track the data migration trajectories for high swap efficiency.
4) Large metadata storage overhead for hybrid memory management. Their space-inefficient metadata structures, such as pointers to index pages in mNM and tags to manage cache lines in cNM, consume significant storage space. Besides, they employ small metadata management granularity to reduce over-fetching. Not only is the potential spatial locality not fully exploited but also the metadata consumes large NM space.

In this article, we propose a novel hybrid memory architecture named Phoenix to preserve the advantages of both cNM and mNM designs while overcoming the above four limitations, by adding a hardware-based hybrid memory management controller (HMMC). All NM can be utilized as cNM or OS-visible mNM. NM can be dynamically switched between cNM and mNM over time to better exploit both temporal and spatial locality benefits for different memory access patterns. cNM can be compelled to become mNM for high memory footprint condition to maximize the total system memory capacity and reduce page faults. The mode switch between cNM and mNM only requires updating metadata in HMMC. Caching (in block granularity) and migration (in page granularity) decisions are made based on workloads' temporal and spatial locality features, as well as the system memory footprint. Phoenix employs a unified set-associative mechanism for page location entry remapping table (PRT) to track page migration and record blocks' caching information in an NM page for evaluating the spatial locality. The page occupied information is kept in a page occupied bit vector. A hotness tracker is designed to track data hotness changes and provide the temporal locality information. To minimize the migration cost between cNM and mNM, all cNM and mNM space is not separate but multiplexed to enable the mode switch process to move only necessary data. For data swap between heterogeneous memories, a fast&slow swap mechanism is proposed to mitigate the data remapping overhead while maintaining high swap efficiency. To limit the metadata storage space, Phoenix employs space-efficient metadata structures and suitable cNM and mNM management granularities. Metadata for hybrid memory management is placed in NM and cached in HMMC. A lightweight metadata prefetcher is adopted to improve the hit rate of the metadata buffer. Concisely, this article makes the following contributions.

1) We present Phoenix, a novel hybrid memory architecture in which NM can serve as both cNM and mNM. The ratio of cNM to mNM is adjustable during runtime to better exploit both temporal and spatial locality benefits for different memory access patterns, without rebooting.
2) All cNM and mNM space in Phoenix is not separate but multiplexed, which minimizes the data movement overhead for mode switch between cNM and mNM. The mode switch only requires updating metadata in HMMC.
3) For data swap between hybrid memories, we propose a fast&slow swap mechanism to mitigate the data remapping overhead while maintaining high swap efficiency.
4) Phoenix greatly reduces the storage space for metadata by 1–2 orders of magnitude through employing space-efficient metadata structures and appropriate migration and caching granularities with a low over-fetching risk.
5) In our evaluations, Phoenix outperforms state-of-the-art designs by 18.2%, and incurs 12.7% less NM traffic and 9.3% less FM traffic on average.

II. Background and Motivation

In order to provide fast and large capacity memory for applications, many works combine NM with FM in hybrid memory systems [2], [3], [4], [7], [8], [9], [10], [11], [12], [20], [21], [22], [23], [28], [29]. In this section, we give the background of the hybrid memory system, three types of NM utilization in prior hybrid memory management designs, fast swap and slow swap approaches for data migration, and our motivations.

A. Hybrid Memory System

Various memory technologies exhibit different tradeoffs in terms of capacity, bandwidth, access latency, and cost. A promising direction toward a more efficient memory system design is to combine two memory technologies with complementary characteristics in a hybrid memory system: 1) NM has a limited capacity with better performance (higher bandwidth or lower latency) and 2) FM has a larger capacity but relatively poorer performance. In DRAM and PM hybrid memory systems [15], [47], [48], [49], PM provides a larger memory capacity but has higher read and write latency than traditional DRAM. Emerging CXL technologies [16], [26], [46], [55], [56] can attach remote byte-addressable memory into the physical address space of the host machine, which appears to the program as a CPU-less NUMA node. Remote memory provides a larger memory capacity than local memory. However, accessing remote memory requires additional CXL round-trip latency. In on-chip HBM and off-chip DRAM hybrid memory systems [2], [21], [23], [25], HBM offers substantially higher memory bandwidth while DRAM provides a larger memory capacity.

B. Three Types of NM Utilization

State-of-the-art designs leverage the NM in cache mode (cNM), memory mode (mNM), and hybrid mode (both cNM and mNM) in hybrid memory systems.

NM Used as cNM: A large body of works have utilized NM as cNM. They can be divided into two classes: 1) block-based [10], [11], [12], [13], [14], [17], [18] and 2) page-based [20], [21], [22]. To enhance the caching capacity

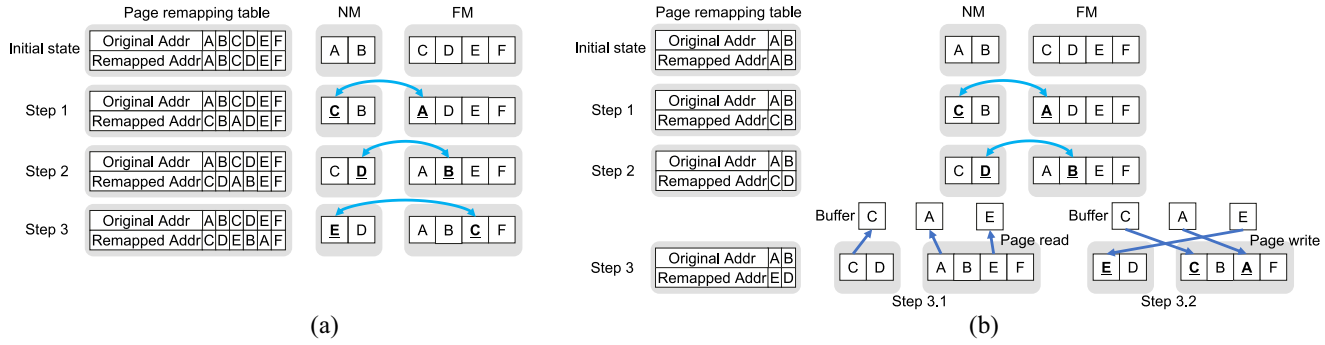

Fig. 1. (a) Fast swap and (b) slow swap.

and better exploit the temporal locality, common block-based cNM manages data at 64-B cache line granularity. Unfortunately, tags may occupy 12.5% of the NM capacity [11]. Besides, block-based cNM has a poor hit rate for workloads with weak temporal and strong spatial locality [12]. Page-based cNM reduces the tag overhead by caching 1–8 kB pages. However, many pages contain data that is not accessed prior to the pages' eviction from the cNM, wasting memory bandwidth [20]. In general, small cache lines better exploit the cache but have higher tag overhead. Large cache lines reduce the tag overhead but may cause over-fetching.

NM Used as mNM: Contrary to cNM designs, mNM designs make all NM capacity visible to OS and have the potential to utilize the bandwidth of all memories for serving memory requests. As a result, they can potentially reap the benefits of both higher aggregate memory bandwidth and larger memory capacity. However, the high remapping overhead [2], [3], [4], [5], [6], [26] and over-fetching issue caused by coarse migration granularity [7], [8], [9], [27] still plague modern mNM designs.

NM in Hybrid Mode: Some research aims to combine the advantages of both cNM and mNM, and they show that hybrid mode can gain more performance benefits than single mode. State-of-the-art hybrid mode designs adopt statically reconfigurable mechanisms to manage the two modes. In particular, KNL [28] supports 25% or 50% NM as cNM. Hybrid2 [23] and Baryon [24] fix a small cNM capacity of 64 MB. These OS-invisible cNM reduce the total memory capacity presented to the OS. All of these designs require a system reboot to switch from one hybrid configuration to another. The caching granularity in cNM is smaller than the migration granularity in mNM, and the caching scheme is more aggressive than the migration scheme, enabling cNM to cache data with changeable hotness in finer granularity and mNM to store data with stable hotness in coarser granularity. Metadata for managing the cNM and mNM cannot be accommodated in SRAM. Hundreds of kilobytes of SRAM are used as a metadata cache to store information about those hot pages.

C. Fast Swap and Slow Swap

Memory mode and hybrid mode designs take two approaches for swapping data between FM and mNM: 1) fast swap [2], [3], [23], [54] and 2) slow swap [4], [5], [6], [7], [8], [24].

Fast swap tracks the migration trajectories of all memory pages to permit them to be remapped to any memory space. Fig. 1(a) gives an example to illustrate the fast swap process. In the initial state, the six pages reside in their original locations, with physical addresses matching those in the OS page tables and TLBs. With the workload running, page C in FM becomes hot and is swapped with page A in step 1. After that, page D becomes hot and is swapped with page B in step 2. Later, page E becomes hotter than page C, where page E should be migrated from FM to NM and page C should be evicted from NM to FM. Since all memory pages are permitted to be remapped to any memory space, page C does not need to return to its original position in FM and is directly swapped with page E in step 3. Each swap requires two page reads and two page writes.

Slow swap only keeps track of the pages remapped to NM and requires remapped pages to return to their original position. The positions of all pages in FM can be inferred from their physical address in page tables, with the exception of pages currently remapped to NM. Fig. 1(b) gives an example to illustrate the slow swap process. Compared to fast swap, the first two steps in slow swap are the same while step 3 requires additional page reads and writes and consumes more data movement bandwidth. In step 3, since the remapped page C should be evicted to its original position in FM, page C and page E cannot be directly swapped. The three pages, C, A, and E are read to three page buffers in step 3.1 and written to their target positions in step 3.2, requiring three page reads and writes. The slow swap generally consumes more memory bandwidth than the fast swap to replace a page in NM with another page in FM, but requires less metadata to track the migration trajectories of remapped pages.

D. Motivation

As described before, hybrid mode design provides an opportunity to combine the advantages of both cNM and mNM designs. However, state-of-the-art hybrid mode designs experience four limitations as follows.

First, they fix the cNM and mNM capacity, which cannot always meet the memory requirements for different memory access patterns. Fig. 2 shows the memory access patterns of

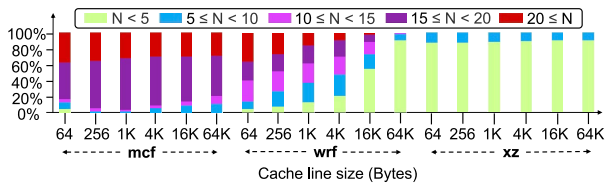

Fig. 2. Percentage of cache lines with different access numbers before eviction in 1-GB cNM. *N* represents the average access number for each 64-B data in different sizes of cache lines.

three representative SPEC2017 [40] workloads' slices selected by Simpoint [59] as an example: we collect the average access number for each 64-B data in different sizes of cache lines before eviction in 1-GB cNM. For the slice of mcf (strong spatial and strong temporal locality), both the large and small cache lines achieve a high access number. To reduce the high tag overhead and better leverage the spatial locality and memory bandwidth, most NM is preferable to be used as mNM with large management granularity without causing over-fetching. For the slice of wrf (weak spatial and strong temporal locality), with the cache line size increasing, the number of hot cache lines decreases, which means large cache lines may bring over-fetching. Thus, a small part of NM is better to serve as mNM to store those large hot data while the other NM is recommended to be utilized as cNM with small management granularity. For the slice of xz (strong spatial and weak temporal locality), most data is rarely reused. A large amount of data movement may generate significant bandwidth cost and frequent hot data evictions. Thus, most NM is favored as mNM with large management granularity for better bandwidth efficiency and spatial locality, with a nonaggressive migration scheme. To sum up, for different categories of workloads, a fixed cNM capacity cannot fully exploit the temporal and spatial locality and utilize all the memory bandwidth. Worse still, they may cause over-fetching and bring unnecessary data movement cost. Furthermore, these fixed OS-invisible cNM reduce the total memory capacity presented to OS and contribute to more page faults under high memory footprint conditions. *The ratio of cNM to mNM should be dynamically adjustable over time.*

Second, the statically reconfigurable designs consume more memory bandwidth for data movement between cNM and mNM due to the separate cNM and mNM space. For example, for evicting a page from cNM to mNM, another mNM page as a victim is swapped out to FM, and then cached for the subsequent access. The data migration from one NM page space to another NM page space brings extra unnecessary migration costs. Thus, *the space of cNM and mNM needs to be multiplexed in order to minimize the data movement overhead for switching the two modes*.

Third, prior designs employ either fast swap or slow swap mechanisms for data swap between mNM and FM. Fast swap designs incur high metadata storage overhead for data remapping while slow swap designs result in low swap performance. Considering the locality in memory access patterns exhibited by the operating system, wherein the OS tends to access memory within a specific address range over a period, only a small portion of memory pages are potentially subject to remapping during a period of workload's running time. Tracking the migration trajectories of a subset of memory pages offers an opportunity to reduce the data remapping overhead while maintaining high swap efficiency. Thus, *the mechanism for data swap between hybrid memories should be redesigned.*

Fourth, the metadata storage overhead in modern hybrid mode designs cannot be ignored. They employ small caching and migration granularities to lower the over-fetching risk. Not only is the potential spatial locality not fully exploited but also the metadata consumes tens of megabytes of NM memory space in a hybrid memory system with a capacity of several gigabytes. These metadata cannot be accommodated in on-chip SRAM and takes a large NM capacity away from the memory system. Worse still, emerging PM [47], [48], [49] and CXL [46], [55], [56] technologies can provide TB-level memory in a hybrid memory system, and the metadata storage space can reach tens of gigabytes. Space-inefficient metadata structures, such as pointers to index pages and tags to manage cNM cache lines in state-of-the-art designs, contribute to the large metadata storage space as well. As a result, *the metadata size should be minimized as small as possible.*

TABLE I
ABBREVIATIONS IN THIS ARTICLE

Abbreviation	Description
FM	Far memory
NM	Near memory
cNM	NM used as a cache of FM
mNM	NM used as OS-visible memory
HMMC	Hybrid memory management controller
PLE	Page location entry
PRT	Page location entry remapping table.
BLE	Block location entry
ADQ	Address difference queue
ADE	Address difference entry

TABLE II
VARIABLES IN THIS ARTICLE

Variables in each remapping set for hotness tracking	
Variable	**Definition**
R_h	NM occupied ratio
T	Hotness threshold for page migration from FM to NM
N_c	Number of cNM pages
N_a	Number of mNM pages where most blocks are accessed
N_n	Number of mNM pages where most blocks are not accessed
Variables for metadata prefetching	
Variable	**Definition**
D_{addr}	Address difference between two memory accesses
R_{addr}	Prefetching range of memory address

III. PHOENIX ARCHITECTURE

Phoenix is a hybrid memory architecture in which NM can be used as either cNM or mNM. Data can be fetched from FM to cNM in block size and migrated in page size between FM and mNM. The ratio of cNM to mNM is dynamically adjustable during runtime to match different memory access patterns, without rebooting. Abbreviations and variables in this article are summarized in Tables I and II, respectively.

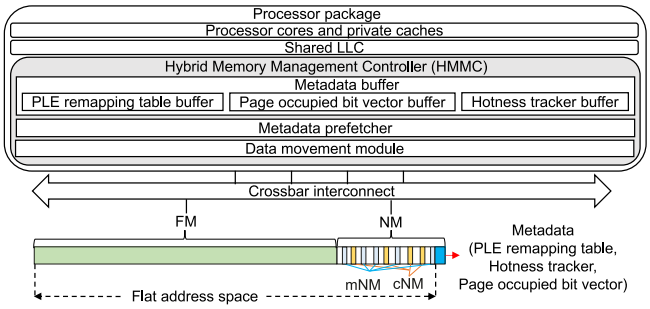

Fig. 3. System overview.

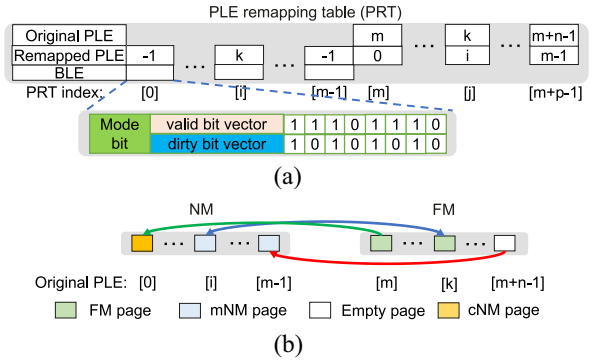

Fig. 4. Example of the unified set-associative mechanism for caching and migration. The data movement indicated by arrows in (b) corresponds to the values of PRT in (a).

A. Phoenix System Overview

Fig. 3 presents the overall architecture of our system. A HMMC is added between the shared LLC and the memory layer to facilitate the data caching and migration functionalities. Phoenix does not modify the page tables, TLBs, and memory controllers. Thus, there are no modifications to the address translation between virtual and physical addresses, nor to the mapping between physical addresses and actual memory locations. Phoenix remaps the physical addresses from the OS to new physical addresses by using the HMMC, and then maps the new physical addresses to actual memory locations. Metadata of Phoenix is stored in NM and consists of three components: 1) the page location entry (PLE) remapping table (PRT); 2) the hotness tracker; and 3) the page occupied bit vector. The metadata is only a few megabytes in size (detailed in Section IV-B), which is negligible compared to the NM size. An on-chip SRAM buffer of a few kilobytes is employed in HMMC to cache frequently utilized metadata (detailed in Section III-H). For each LLC miss memory request, the PRT is responsible for querying the actual location of requested data, either in FM or NM. The PRT records the information of remapped pages and cached blocks, and provides the spatial locality information. The hotness tracker keeps track of the most recently requested hot pages to provide the temporal locality information and monitors the memory footprint, determining the migration and caching logic in Phoenix. The occupied information for all memory pages is recorded in the page occupied bit vector. To lower the miss rate of the metadata buffer in HMMC, a metadata prefetcher prefetches metadata from NM to HMMC asynchronous to memory access requests. Similar to previous works [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [19], [20], [21], [22], [23], [24], [25], data movement between NM and FM is executed through a data movement module. The data movement module records the pages being migrated. For data movement from one memory location A to another location B, Phoenix first reads data from A to the data movement module and then writes to B. For a regular request requesting for a page being migrated, the request retrieves the needed data from the data movement module and returns it to applications. The request does not need to access memory. By using the HMMC, NM is logically, not physically, partitioned into cNM and mNM.

For high memory footprint condition, in order to provide more OS-visible memory and reduce page faults, part of cNM may be compelled to become mNM. The remaining NM can still serve as cNM. For low memory footprint condition, since the data migration granularity of mNM (page size) is larger than the fetching granularity of cNM (block size), data with strong spatial locality is preferred to be placed into mNM. This approach can ensure the hit rate of NM, better utilize all the memory bandwidth, and lower the metadata query overhead for blocks. cNM is better suited to pages with weak spatial locality. Only hot blocks are cached in cNM to reduce overfetching. In this way, the entire NM except for the space of metadata, is not only visible as a flat address space for OS but also can flexibly serve as cNM or mNM most of the time to match different workloads' memory access patterns.

B. Memory Space Layout and Metadata

Since cNM can be flexibly switched to mNM, all the FM and NM except for the space of metadata, are visible as a flat address space for OS, as shown in Fig. 3. The mode switch does not affect the physical memory visible to the OS. The space of cNM and mNM in Phoenix is multiplexed. The mode switch (detailed in Section III-F) between cNM and mNM only modifies the metadata buffer in HMMC and requires moving necessary data blocks.

For metadata management, direct remapping has a bad performance [6], [23], [30], [31] due to poor flexibility, uneven NM utilization, and frequent page swaps. Besides, the hardware query overhead for a fully associative remapping table is unacceptable on chip [23], [32], [33]. Therefore, for a balance between the hardware overhead and performance, Phoenix adopts a unified set-associative mechanism to manage metadata both for caching and migration, as Fig. 4 illustrates. An FM page is only allowed to be cached or migrated to another NM page location in the same remapping set.

PRT: As shown in Fig. 4(a), in each remapping set, the set-associative PRT holds the original PLEs for tracked FM pages, the remapped PLEs for all NM pages and tracked FM pages, and block location entry (BLEs) for all NM pages. To limit the metadata size in PRT, in each remapping set, Phoenix tracks the migration and caching trajectories of a few memory pages: all NM pages along with some hot FM pages [e.g., m NM and p FM pages in Fig. 4(a)]. Only these NM and FM

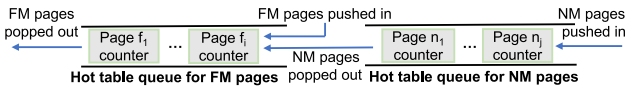

Fig. 5. Hot table in a remapping set.

pages tracked by PRT are permitted to be remapped to another memory space. Note that if the tracked FM pages become cold during runtime, they are replaceable by other hot FM pages. The original PLE refers to the original page index [i.e., offset from the first page in this set, as shown in Fig. 4(b)] in the remapping set, decided by the OS memory allocator and the virtual to physical address mapping mechanism in OS. Since all NM pages are tracked by PRT all the time while the tracked FM pages are changeable during runtime, PRT only records the original PLEs for tracked FM pages. The remapped PLE combines the functions of page address remapping and page allocation (-1: the page has not been allocated). For example, the blue arrow in Fig. 4(b) indicates a swap between the ith and kth pages in the remapping set, recorded by the corresponding ith and jth PRT entries. The green arrow and red arrow represent page caching and migration, respectively. One PLE requires less storage space ($\lceil \log_2 (m+n) \rceil$ bits, where m and n are the numbers of NM pages and FM pages in a remapping set, respectively) than a traditional tag or pointer (a few bytes).

Since each NM page can serve as either cNM or mNM, Phoenix utilizes a BLE to indicate the mode of an NM page as well as the valid and dirty information of blocks in the NM page. One BLE contains a mode bit, a valid bit vector, and a dirty bit vector, as Fig. 4(a) shows. The mode bit denotes whether the NM page is in cache mode or memory mode. For a cNM page, the corresponding BLE indicates if the cached blocks are valid and dirty. For an mNM page, the valid bit vector records all accessed blocks in the page for evaluating the spatial locality.

Page Occupied Bit Vector: The page occupied bit vector records the occupied information for all memory pages in each remapping set. An occupied bit indicates if the memory page space has been occupied, queried by the page allocation and data movement process.

Hotness Tracker: In each remapping set, the hotness tracker includes a hot table and five parameters: the NM occupied ratio (R_h), a hotness threshold (T) to decide if an FM page should be brought in NM for high R_h condition, the number of cNM pages (N_c), and the number of mNM pages in which most blocks have/have not been accessed (N_a/N_n). To reduce the metadata storage overhead and get the temporal locality information, the hot table only monitors the hottest pages, including all NM pages and the recently accessed FM pages. The hot table includes two queues with LRU replacement strategy, one for NM pages and the other for FM pages, as shown in Fig. 5. Each entry in the queue serves as a counter to record the access number for a page before popped out from the queue. When a memory request accesses a page, the counter for the page is incremented by one. Recently requested page entries are pushed back into the queue, and page entries that have not been accessed for a long time are popped out of the queue. Therefore, the hot table can always identify recently requested hottest pages during runtime. The popped-out NM page entries are pushed back into the FM queue and incur page evictions from NM to FM. The five parameters are used for making data movement decisions and detailed in Section III-F.

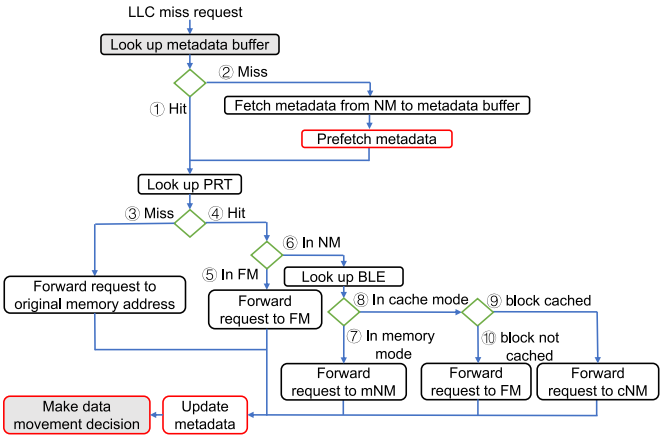

Fig. 6. Memory access flow. Operations within the black closed boxes are on the critical path. Operations within the red boxes are asynchronous to memory accesses for LLC miss requests.

C. Memory Access Flow

Fig. 6 illustrates the memory access flow. Using the requested memory address, the LLC miss request first checks if the required metadata has been cached in the metadata buffer in HMMC. For a metadata buffer hit (①), Phoenix looks up the PRT buffer to determine the actual memory address for the requested data. For a metadata buffer miss (②), Phoenix fetches the required metadata from NM to the metadata buffer and asynchronously prefetches metadata that might be utilized later. After looking up the PRT, a PRT miss (③) indicates that the requested page is not remapped to other memory space, and the LLC miss request directly accesses the original requested memory address. In the case of a PRT hit (④), the target page may be remapped to FM (⑤) or NM (⑥). For ⑤, the memory request goes to FM. For ⑥, HMMC checks the BLE if the NM page is in memory mode (⑦) or cache mode (⑧). If the page is in memory mode, the memory request directly goes to mNM. If the page is in cache mode, HMMC further checks the valid bit vector to determine whether the target block is cached. For block cached (⑨), the requested data is in cNM. For block not cached (⑩), the memory request goes to FM. All memory accesses update the corresponding metadata and may incur data movement asynchronously.

D. Page Allocation Process

Few previous studies discuss the page allocation process. However, simply allocating pages to FM or NM for all workloads by the OS memory allocator does not fully exploit the benefits of combining hybrid memory technologies. A suitable page allocation mechanism can reduce the migration cost for hot pages migrated from FM to NM and cold pages evicted from NM to FM. Since the PRT records the

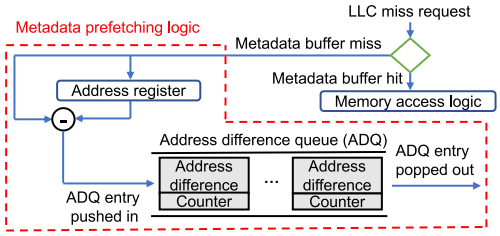

Fig. 7. Metadata prefetching mechanism.

remapping information of all NM pages and the page occupied information of all memory pages, the pages whose original PLEs are within the NM address range could be allocated to any free memory space in a remapping set. Based on the observation that adjacent allocation requests intend to have similar memory access patterns [24], [39], [40], [41], a hotness-based remapping allocation mechanism is applied to Phoenix. If the recently allocated pages still reside in the hot table queue for NM pages, the page is allocated in NM. Otherwise, Phoenix allocates the page in FM.

E. Metadata Prefetching

Since metadata for managing the hybrid memory system is placed in NM and cached in an on-chip SRAM, Phoenix incorporates a lightweight metadata prefetcher to improve the hit rate of the metadata cache and lower the metadata access latency on the critical path, as shown in Fig. 7. A recent work [42] reports that the address difference between two consecutive memory accesses is effective for metadata prefetching. For each memory access through HMMC, Phoenix employs a 64-bit address register to record the requested memory address. The address difference is calculated by

$$D_{\text{addr}} = |\text{Addr}_{\text{last}} - \text{Addr}_{\text{curr}}| \tag{1}$$

where D_{addr} is the address difference between two consecutive memory accesses, $\text{Addr}_{\text{last}}$ is the last requested memory address recorded in the register, and $\text{Addr}_{\text{curr}}$ is the currently requested memory address. An address difference queue (ADQ) keeps the recently generated address differences along with their respective counts. Each address difference entry (ADE) in the ADQ includes an address difference (8 bytes) and a counter (1 byte). Every time an address difference is calculated, the counter of the corresponding ADE is incremented by one. If the address difference is not in the ADQ, it is pushed into the ADQ and its counter value is set to one. ADQ entries unused for a long time are evicted from the ADQ to catch the changes in memory access patterns. For a metadata miss in HMMC whose page address is Addr, Phoenix prefetches the metadata in ranges of $\text{Addr} \pm R_{\text{addr}}$ (R_{addr} is the prefetching range), $\text{Addr} + D_{\text{addr}} \pm R_{\text{addr}}$, and $\text{Addr} - D_{\text{addr}} \pm R_{\text{addr}}$, asynchronous to memory access logic.

F. Data Movement Decision

In this section, Phoenix makes the data movement decision based on the temporal and spatial locality features of different memory access patterns and the system memory footprint to support flexible cNM and mNM capacities, lower the over-fetching risk, maximize the OS-visible memory capacity, and remove the eviction latency of cNM pages from the critical path in the case of memory shortage. Memory accesses bring changes to the hotness tracker and as a result, incur data movement. Besides, for high memory footprint, data movement is triggered to meet OS memory requirements.

For spatial locality, the BLE tracks the accessed blocks in both cNM and mNM pages. If more than (1/2) of all blocks in a page have been fetched into cNM, that means the page has a strong spatial locality and should be switched to an mNM page. More requested pages should be migrated to mNM if most NM pages have strong spatial locality. mNM pages are switched from cNM pages (most blocks in the page have been accessed) or migrated from FM (not sure if most blocks in the page have been accessed). Thus, mNM pages with a high access ratio reflect a strong spatial locality degree (SL), while those with a low access ratio and the remaining cNM pages reflect a weak SL. The SL in a remapping set is evaluated by

$$\text{SL} = N_a - N_n - N_c. \tag{2}$$

For $\text{SL} > 0$ (strong spatial locality), more hot data should be brought in mNM to better exploit the spatial locality and utilize the memory bandwidth. For $\text{SL} \leq 0$ (weak spatial locality), hot data should be cached in cNM to reduce over-fetching.

For temporal locality, Phoenix uses the hot table to track hot and recently accessed pages. It is hard to benefit from bringing data with low access frequency into NM, which wastes memory bandwidth and may cause frequent hot page evictions for high NM footprint condition, resulting in performance degradation. The threshold T in the hotness tracker can alleviate this issue. If R_h is high, for $\text{SL} > 0$, only pages whose hotness value is larger than T are permitted to be migrated to mNM, and for $\text{SL} \leq 0$, only blocks in a page whose hotness value is larger than T are permitted to be cached in cNM. In this way, for weak temporal locality workloads, a large amount of data with low access frequency is not brought into NM, which reduces the data movement overhead and eviction frequency of hot pages and better utilizes the FM bandwidth. For strong temporal locality workloads, despite some hot data that may benefit from caching and migration not brought in NM, the eviction frequency of those hottest pages is reduced to make them serve more memory requests before eviction. The FM bandwidth is better utilized, and the data movement overhead is mitigated.

Based on the above analysis to better exploit both temporal and spatial locality benefits for different memory access patterns, the data caching and migration from FM to NM are shown as follows.

Data Caching and Migration From FM to NM: Data caching and migration are triggered by the conditions shown in Fig. 8(a) and detailed as follows.

1) For accessing an FM page, ❶ if $\text{SL} > 0$ with a low R_h, the page is migrated to mNM due to the strong spatial locality. ❷ If $\text{SL} > 0$ with a high R_h, the page is migrated to mNM only if its hotness value is larger than T. ❸ If $\text{SL} \leq 0$ with a low R_h, the page is cached to cNM and only the requested block is fetched. ❹ If

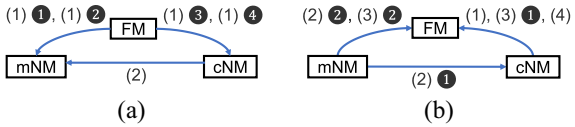

Fig. 8. Data caching, migration, and eviction. (a) Data caching and migration. (b) Data eviction.

SL ≤ 0 with a high R_h, the page is cached to cNM only if its hotness value is larger than T.

2) For accessing a cNM cached page, if the target block is not cached in cNM, Phoenix caches the block. If most blocks in the page have been cached, the cNM page turns into an mNM page. Since the cNM and mNM space is multiplexed in Phoenix, only blocks not cached are fetched from FM, minimizing the data movement overhead for the mode switch.

Data eviction from NM to FM, aimed at providing more free NM space, evicting zombie pages in NM, and removing the eviction latency of cNM pages from the critical path in the case of memory shortage, is detailed below.

Data Eviction From NM to FM: Data eviction from NM to FM is triggered by the conditions shown in Fig. 8(b) and detailed as follows.

1) If a cNM page entry is popped out from the hot table queue for NM pages, dirty blocks in the corresponding cNM page are evicted to FM.
2) Since evicting an mNM page to FM consumes much higher bandwidth (at least 2×) than a cNM page, ❶ if there is a free FM page in the remapping set, mNM pages to be evicted have one more chance to reside in NM. The mNM page to be evicted from the hot table queue is switched to cNM mode without data migrated to FM. A free FM page is marked as occupied and all blocks in the mNM page are marked as dirty. The mode switch from mNM to cNM is designed as a buffering mechanism for increasing colder pages in mNM, aiming to lower the migration cost resulting from hotness fluctuations and extend the residency in NM to serve more memory requests. There is no data movement cost for mNM switched to cNM due to our multiplexed space design. In this way, if these pages then become hotter, no data movement is required. ❷ If there is no free FM page in the remapping set, mNM pages to be popped from the hot table queue are swapped with FM pages to be migrated to NM.
3) For high R_h, in case of both the head page in the hot table queue for NM pages and its counter value remain unchanged over a long time, the page (zombie page) should be evicted to FM since there is no other page able to evict the zombie page from NM. ❶ If the zombie page is a cNM page, dirty blocks in the cNM page are flushed to FM. ❷ If the zombie page is an mNM page, the mNM page is evicted from NM to FM.
4) If the system memory footprint is high in general (i.e., the memory address in LLC miss request is larger than the FM capacity), cNM pages in multiple remapping sets are flushed to FM. This batching mechanism provides more OS-visible memory to reduce page faults. For pages to be allocated in these remapping sets later, there is no need to wait for the eviction from cNM to make free page space, which removes the eviction latency from the critical path. In these remapping sets, all NM is not permitted to be used as cNM until the OS memory footprint drops.

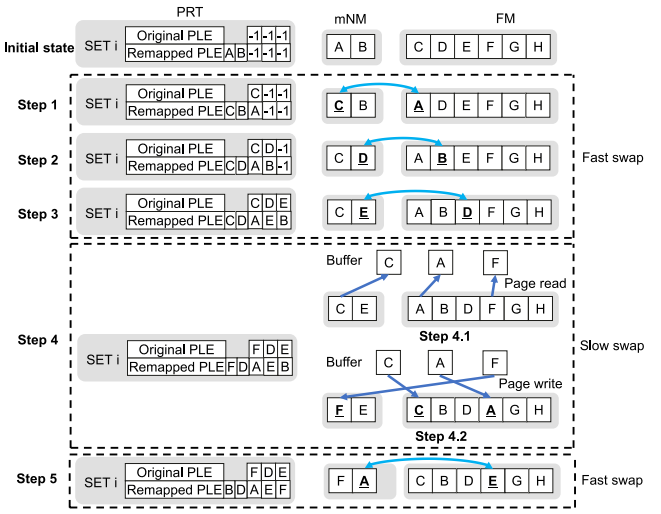

Fig. 9. Fast&slow swap.

To sum up, the migration and caching mechanism in Phoenix addresses the limitations of state-of-the-art hybrid mode designs for different kinds of workloads in Fig. 2. For workloads with strong spatial and strong temporal locality, most NM is used as mNM for better spatial locality and memory bandwidth efficiency without over-fetching. For workloads with strong spatial and weak temporal locality, most NM is used as mNM with a nonaggressive migration scheme to better utilize all the memory bandwidth and exploit spatial locality benefits. Data with a low access frequency does not bring significant memory bandwidth waste and frequent evictions. For workloads with weak spatial and strong temporal locality, most NM is used as cNM to better exploit the temporal locality benefit and reduce over-fetching. Compared to the fixed cNM capacity in existing hybrid mode designs, more cNM capacity contributes to a lower eviction frequency for data in cNM. For workloads with weak spatial and weak temporal locality, few data are moved. Several optimizations, such as advanced footprint caching [14], [19] and software-assisted metadata management [2], [13], are directly applicable to Phoenix. However, such options are orthogonal to our contributions and require modifications to existing operating systems and software. We do not include them in our base design to clearly attribute the performance gains to our proposed techniques.

G. Fast&Slow Swap

For data swap between FM and mNM in Section III-F, Phoenix adopts a fast&slow swap mechanism to combine the advantages of fast swap (high swap efficiency) and slow swap (low metadata overhead for data remapping). Considering the locality in memory access patterns exhibited by the operating system, wherein the OS tends to access memory within

a specific address range over a period [50], [51], [52], the majority of memory pages remain inactive. Consequently, only a small portion of memory pages are potentially subject to remapping during a period of workload execution. In each remapping set, the PRT tracks the migration trajectories of all NM pages and a few FM pages, rather than tracking all FM pages or none of the FM pages. For data swap between two pages whose migration trajectories have been both recorded in PRT, Phoenix performs fast swap and updates the remapped PLEs in PRT. Otherwise, Phoenix performs slow swap and page entry replacement in PRT. Fig. 9 gives an example of the fast&slow swap mechanism, where each remapping set includes two NM pages and six FM pages. The PRT tracks the migration trajectories of two NM pages and three FM pages. Phoenix performs fast swap until there is no free remapped page entry in PRT in steps 1–3. In step 4, page F whose migration trajectory has not been recorded in PRT, is swapped with page C. Phoenix performs slow swap to ensure that the evicted page C returns to its original position in FM. In step 5, the migration trajectories of both page A and page E have been recorded in PRT. The two pages are directly swapped through the fast swap approach. In Section IV-B, we evaluate the performance of Phoenix with different numbers of tracked remapped FM pages.

H. Hardware Cost

Phoenix is a hardware-based design with core components and processing logic integrated into the on-chip HMMC. Compared to software-managed schemes, hardware design provides faster address translation. The processing logic of Phoenix can also be implemented in hardware without software or OS interference. The hardware cost of Phoenix comes from the on-chip capacity requirement and peripheral logic circuits. The processing logic of HMMC mainly includes the address resolution by querying the metadata buffer, the mode switch between cNM and mNM by updating the metadata buffer, and a few simple logic for data movement and metadata prefetching, which is negligible compared to the on-chip capacity overhead. The on-chip capacity of HMMC (evaluated in Section IV-B) consists of 1) 1320-B PRT buffer (PLE: 640 entries, each entry is 10 bits in size; mode bit: 64 entries, each entry is 1 bit in size; valid bit vector: 64 entries, each entry is 4B in size; dirty bit vector: 64 entries, each entry is 4 B in size); 2) 512-B page occupied bit vector (4096 entries, each entry is 1 bit in size); 3) 216-B hotness tracker buffer (hot table: 192 entries, each entry is 1 B in size; parameters: 8 entries, each entry is 3 B in size); 4) 144-B metadata prefetcher (16 entries, each entry is 9 B in size); and 5) 16 kB+32 B data movement buffer (four 4-kB-sized pages with their corresponding 8-B-sized starting physical addresses). The total on-chip capacity required by Phoenix is only about 18 kB+176 B (2-kB metadata buffer, 144-B metadata prefetcher, and 16 kB+32 B data movement buffer). Compared to hundreds of kilobytes of on-chip metadata buffer in previous works [14], [23], [24], the hardware cost of Phoenix is significantly reduced.

TABLE III
SYSTEM CONFIGURATION

Processor parameters [25]	
Cores	ARM A72(AArch64), 3.6GHz, 24 cores
IL1/DL1 cache	private 64 KB per core, 4-way, LRU
L2 Cache	private 256 KB per core, 8-way, SRRIP
L3 Cache	shared 8 MB, 16-way, DRRIP
DRAM-PM parameters [14], [24]	
DDR3_2000	512 MB, 1 64-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 11-10-24-10
PCM	32 GB, 1 64-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 11-58-80-11
Local-remote memory parameters [46], [55], [56]	
DDR4_3200	512 MB, 1 64-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 22-22-38-22
DDR4_3200	32 GB, 1 64-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 22-22-38-22
Average CXL round-trip latency	100 ns
HBM-DRAM parameters [34], [35]	
HBM2	512 MB, 8 128-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 7-7-17-7
DDR4_3200	32 GB, 1 64-bit channels, 8 banks, tCAS-tRCD-tRAS-tRP: 22-22-38-22

TABLE IV
BENCHMARK CHARACTERISTICS (BM: BENCHMARK, MPKI: LLC MISSES PER KILO INSTRUCTIONS, AND FP: FOOTPRINT)

	BM	MPKI	FP(GB)	BM	MPKI	FP(GB)
High MPKI	mcf	51.2	32.3	cloverleaf	29.9	31.7
	Stream	35.1	32.4	bwaves	27.3	31.9
	roms	32.7	31.9	xalancbmk	22.1	28.2
	lbm	30.8	32.3			
Medium MPKI	cam4	18.9	32.2	hpccg	11.7	29.5
	wrf	17.5	27.4	fotonik3d	11.1	28.2
	cactuBSSN	14.3	32.1	x264	10.2	22.3
	xz	14.1	32.0			
Low MPKI	namd	8.3	31.3	SP	1.1	28.6
	leela	5.6	23.6	comd	0.9	30.2
	miniAMR	3.6	32.4	miniFE	0.8	32.1
	nab	2.9	31.8	miniGhost	0.5	29.8

IV. EVALUATION

A. Experimental Setup

We use the gem5 simulator [60] and DRAMSim2 [61] to model Phoenix and other control schemes. We compare Phoenix against a baseline configuration and nine state-of-the-art designs.

1) *A baseline system without NM:* BASE.
2) *Two hybrid mode designs:* Hybrid2 [23] and Baryon [24].
3) *Three memory mode designs:* Chameleon [2], AGDM [5], and RHPM [6].
4) *Four cache mode designs:* Banshee [22], Alloy Cache (AC) [11], Unison Cache (UC) [21], and NOMAD [13].

In order to evaluate the effectiveness of our proposed Phoenix design comprehensively, we implement all systems in three different hybrid NM and FM architectures (DRAM-PM, local–remote memory connected with CXL, and HBM-DRAM) as shown in Table III. Page faults in our system are assumed to be serviced by a solid-state disk with a latency of 36 microseconds (100K cycles) [2], [54]. Since state-of-the-art designs achieve the best performance under different on-chip metadata sizes ranging from several kilobytes to 512 kB, for a

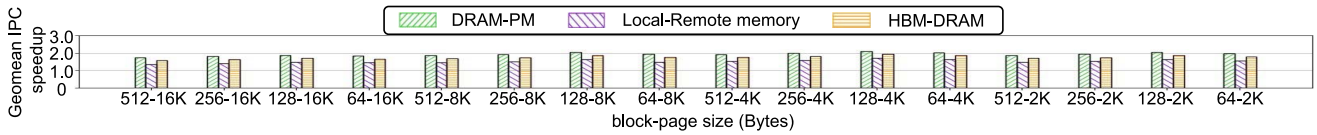

Fig. 10. Geometric mean of the IPC speedup for Phoenix with different block and page sizes, normalized to BASE, in the three hybrid memory systems.

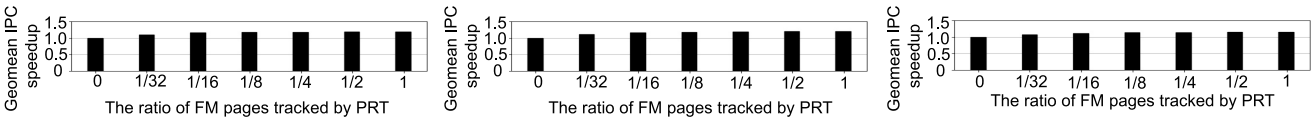

Fig. 11. Geometric mean of the IPC speedup for Phoenix with different ratios of FM pages tracked by PRT, normalized to that for Phoenix with none of the FM pages tracked by PRT, in the three hybrid memory systems. (a) DRAM-PM hybrid memory system. (b) Local–remote hybrid memory system. (c) HBM-DRAM hybrid memory system.

fair comparison among all evaluated designs, we allow 512-kB on-chip SRAM in the memory controller to buffer requested metadata and the rest unused SRAM to buffer frequently requested data.

The configurations for Phoenix are shown as follows. Both cNM and mNM are managed with 8-way associativity. In each remapping set, the hot table monitors sixteen recently accessed FM pages for a balance between the metadata overhead and data migration efficiency. We set T as the smallest hotness value of NM pages in each remapping set to dynamically match the data hotness pattern during runtime. The R_h is defined as high if its value reaches 1 to maximize the NM utilization rate. The ADQ size is set to 16, which is effective for metadata prefetching in Phoenix and does not incur excessive overhead. For metadata prefetching, the prefetching range R_{addr} is set to 8 kB, where more than 95% of requested metadata is prefetched from NM to HMMC in our experiments. We use the SPEC2017 [40], Mantevo [53], NAS [37], and Stream [38] benchmarks in Table IV to evaluate our design. We simulate a minimum of 24 billion instructions for each benchmark by using the Simpoint [59].

B. Design Space Exploration and Metadata Overhead

Phoenix can be configured with any block and page size, which affects the system performance and metadata size. We explore some possible page size (ranging from 2 to 16 kB) and block size (ranging from 64 to 512 B) configurations in the three hybrid memory systems, and their performance results are shown in Fig. 10. We use the average normalized instruction per cycle (IPC) for all benchmarks in Table IV during execution as a performance metric for the speedup comparison. Smaller blocks miss the opportunity to exploit spatial locality while larger blocks cause over-fetching. Thus, a block of 128 B is a good compromise between the spatial locality exploitation and bandwidth consumption. For the same block size, 4 kB pages demonstrate the best performance among evaluated page size configurations. Our design achieves the best performance at 128 B blocks and 4 kB pages, and for the rest of experiments in this article, we present our results in this configuration.

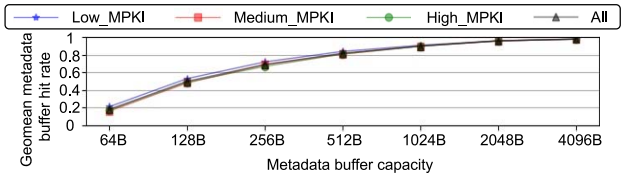

Fig. 12. Geometric mean of the metadata buffer hit rate for Phoenix with different metadata buffer capacities in HMMC.

Fig. 11 illustrates the performance of Phoenix with different ratios of FM pages tracked by PRT. It is predictable that Phoenix achieves the highest IPC by tracking all FM pages (the ratio = 1) and the lowest IPC by tracking none of the FM pages (the ratio = 0), due to the performance gap between the fast swap and slow swap. The metadata overhead is proportional to the number of tracked FM pages. However, the system's performance gradually increases with the enlargement of the ratio and tends to saturate. Only a small portion of memory pages are potentially subject to remapping during a period of workload's running time. In the three hybrid memory systems, the IPC achieved by tracking (1/16) of all FM pages is, on average, 97.2% of that by tracking all FM pages. Therefore, in each remapping set, Phoenix tracks the caching and migration trajectories of (1/16) of all FM pages, considering both performance and metadata overhead. The total metadata storage space in NM (3960 kB: 2496-kB PRT, 1040-kB page occupied bit vector, and 424-kB hotness tracker) is reduced by 1–2 orders of magnitude compared to prior designs.

Fig. 12 demonstrates the geometric mean of the metadata buffer hit rate over different metadata buffer capacities in HMMC. When the capacity is larger than 2 kB, the metadata buffer hit rate is close to 1. Phoenix sets the default metadata buffer capacity as 2 kB, where the metadata buffer has an average hit rate of 96.5% without incurring much hardware cost in HMMC.

Fig. 13 gives the sensitivity results for different variables in Phoenix running on the DRAM-PM hybrid memory system. In Fig. 13(a), with the increasing number of monitored FM pages in each remapping set, Phoenix achieves higher IPC speedup

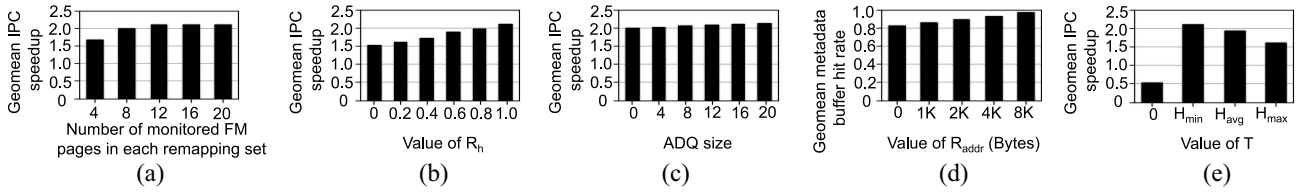

Fig. 13. Sensitivity analysis for different variables in Phoenix running on the DRAM-PM hybrid memory system. (a) IPC speedup for Phoenix with different numbers of monitored FM pages, normalized to BASE. (b) IPC speedup for Phoenix with different values of R_h defined as high, normalized to BASE. (c) IPC speedup for Phoenix with different ADQ sizes, normalized to BASE. (d) Metadata buffer hit rate for Phoenix with different values of R_{addr}, normalized to BASE. (e) IPC speedup for Phoenix with different values of T, normalized to BASE.

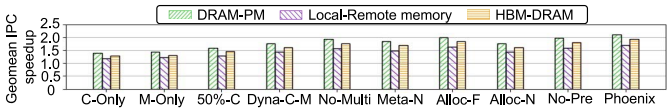

Fig. 14. Performance factors breakdown, normalized to BASE, in the three hybrid memory systems.

and tends to saturate. More monitored FM pages in the hot table enable Phoenix to catch hotness changes of more recently requested pages. The metadata overhead for hotness monitoring is proportional to the number of monitored FM pages, while the system's performance tends to saturate. Monitoring sixteen FM pages in each remapping set reduces the metadata overhead without degrading performance. Fig. 13(b) illustrates the performance of Phoenix with different values of R_h defined as high. As described in Section III-F, if the R_h value in a remapping set is low, requested FM pages are directly cached/migrated to NM. If the R_h value in a remapping set is high, only requested FM pages whose hotness value is larger than T are cached/migrated to NM. With the increasing value of the R_h threshold defined as high, more requested FM pages are brought to NM, contributing to a higher NM utilization rate and more performance benefits. The performance of Phoenix with different ADQ sizes and R_{addr} values is shown in Fig. 13(c) and (d), respectively. Larger ADQ size and R_{addr} value improve the metadata buffer hit rate. The performance tends to saturate when the ADQ size is larger than 12. Prefetching 8 kB R_{addr} ranges contributes to a 97.8% metadata buffer hit rate. We evaluate Phoenix's performance with different T values in Fig. 13(e). Phoenix has a bad performance with a fixed threshold of 0. Not only can the fixed threshold not dynamically match the data hotness pattern during runtime, but it also consumes a significant amount of bandwidth for data movement. H_{min} (the minimum hotness value of NM pages in each remapping set) achieves higher IPC speedup than H_{avg} (the average hotness value of NM pages in each remapping set) and H_{max} (the maximum hotness value of NM pages in each remapping set). The reason is that a higher T value prevents a large number of requested FM pages from being brought into NM, contributing to fewer memory requests served by NM.

C. Performance Breakdown

Fig. 14 illustrates the geometric mean of the IPC speedup for all benchmarks in Table IV to show the effect of our proposed optimizations. The performance speedup of Phoenix can be attributed to the cNM and mNM combination, the dynamically adjustable cNM and mNM capacities, the multiplexed cNM and mNM space, the metadata buffer in HMMC, the page allocation mechanism, and the metadata prefetcher. Each of these optimizations contributes to 22.7%, 21.4%, 13.9%, 18.1%, 8.8%, and 15.1% of Phoenix's overall performance gain, respectively. Each entry on the x-axis represents Phoenix removing one of our proposed optimizations. From left to right: C-Only and M-Only represent all the NM used as cNM and mNM, respectively. The more benefits achieved by M-Only than C-Only mainly stem from the better memory bandwidth efficiency for NM and FM and more OS available memory space. We also evaluate the performance of fixing the cNM capacity at 50% of total NM capacity (50%-C), which outperforms the single-mode designs. Dynamically adjustable cNM and mNM capacity design (Dyna-C-M) better exploits temporal and spatial locality benefits for different memory access patterns and outperforms fixed cNM and mNM capacity designs. A hybrid mode design without the multiplexed cNM and mNM space (No-Multi) brings more data movement overhead for mode switch, wasting both NM and FM bandwidth. Placing all the metadata in NM without the metadata buffer in HMMC (Meta-N) degrades the performance mainly due to the performance gap between SRAM and NM as well as more NM traffic. Alloc-F and Alloc-N represent allocating all pages to FM and NM, respectively. Compared to our hotness-based page remapping allocation, Alloc-F consumes more bandwidth for hot data migration from FM to NM. Alloc-N reduces the migration cost for workloads with low memory footprint since all the data can be placed in NM while incurs significant bandwidth waste for high memory footprint workloads due to a large amount of data evicted from NM. Removing the metadata prefetcher from Phoenix (No-Pre) incurs a lower metadata buffer hit rate and higher metadata miss latency on the critical path.

D. Performance Comparisons

Figs. 15–17 illustrate the performance of Phoenix and state-of-the-art designs for high, medium, low and all MPKI benchmarks in the three hybrid memory systems, respectively: Phoenix outperforms the best previous designs by 27.1%, 20.5%, 8.4%, and 18.2% on average in each benchmark suit. Phoenix has the best performance on all the 22 benchmarks in Table IV. Compared to the two hybrid mode designs Hybrid2 and Baryon, Phoenix makes better use of NM for locality exploration due to the adjustable capacities for cNM and

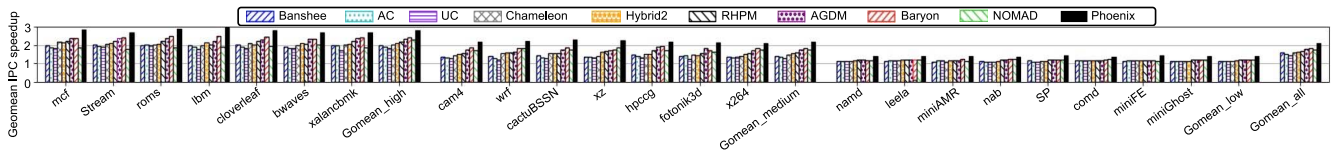

Fig. 15. Performance of Phoenix and state-of-the-art designs, normalized to BASE, in the DRAM-PM hybrid memory system. The benchmarks are sorted by MPKI and the geometric mean of an MPKI class is presented at the right of the class.

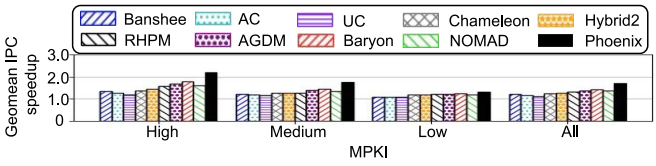

Fig. 16. Performance of Phoenix and state-of-the-art designs, normalized to BASE, in the local–remote hybrid memory system.

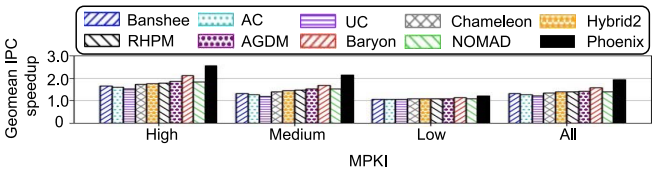

Fig. 17. Performance of Phoenix and state-of-the-art designs, normalized to BASE, in the HBM-DRAM hybrid memory system.

mNM. Besides, since the metadata storage space is greatly minimized and the cNM and mNM space is multiplexed, Phoenix provides more NM space for workloads and consumes less memory bandwidth for mode switch between cNM and mNM. Furthermore, the dynamic mode switch design in Phoenix provides more available memory for the OS and reduces page faults for workloads with high memory footprint. Chameleon, RHPM, and AGDM are designed based on POM [3] with the added option to economize on migration bandwidth. They restrict only one NM sector in each remapping set, which leads to uneven NM utilization rates in different remapping sets and frequent sector migration and eviction. Furthermore, RHPM and AGDM adopt the slow swap mechanism for data migration between FM and NM, consuming more memory bandwidth to swap data between hybrid memories and degrading the system performance. Cache designs fail to utilize the aggregate memory bandwidth of both NM and FM. The high eviction frequency for hot data in NM and more page faults caused by the OS invisible NM space architecture give rise to a significant amount of migration bandwidth and page fault latency, degrading the system performance. Moreover, state-of-the-art designs do not consider both the spatial and temporal locality of the running workloads, incurring unnecessary memory bandwidth consumption and data fetching latency due to the over-fetching issue and the inability to premigrate upcoming requested data from FM to NM. Significant metadata storage overhead in state-of-the-art designs reduces the available NM space for running workloads, degrading the system performance as well.

E. Over-Fetching Analysis

Over-fetching is a common issue that plagues modern hybrid memory management designs and degrades system performance. By collecting the percentage of data brought in NM but unused, we analyze the over-fetching for Phoenix and two state-of-the-art hybrid mode designs: 10.4% in Phoenix (128 B blocks and 4 kB pages), 16.8% in Baryon (256 B sub-blocks, 2 kB blocks, and 16 kB super-blocks), and 18.5% in Hybrid2 (256 B blocks and 2 kB pages). The reasons for the over-fetching reduced by Phoenix are as follows.

1) The over-fetching is mainly caused by workloads with weak spatial locality. The adjustable design in Phoenix can provide more cNM capacity (up to 512 MB) than Hybrid2 (64 MB) and Baryon (64 MB) for workloads with weak spatial locality, which greatly reduces the eviction frequency in cNM and enables blocks in cNM to serve more memory requests before eviction.
2) Our data movement mechanism only permits data in a page that reaches a certain hotness level (i.e., T) to be brought in NM for high memory footprint condition, which prevents pages with low hotness into NM and reduces the eviction frequency of pages in NM. Hybrid2 and Baryon bring all requested blocks into cNM, which causes significant over-fetching for workloads with weak spatial locality.
3) The buffering mechanism for mNM page eviction extends the pages' residency in NM, enabling data brought in NM to serve more memory requests before eviction.
4) The granularity for data caching and eviction in Phoenix is smaller than that in Hybrid2 and Baryon, mitigating the over-fetching issue.

F. Memory Traffic Analysis

Fig. 18(a) and (b) illustrates the normalized NM and FM traffic for each benchmark group, respectively. Phoenix economizes memory bandwidth for both NM (12.7% less traffic than the best, i.e., AGDM) and FM (9.3% less traffic than the best, i.e., Banshee), which is one of the reasons why Phoenix has the best performance. The two hybrid mode designs, Hybrid2 and Baryon, both incur extra unnecessary NM traffic for data migration from one space in NM to another space in NM, which can be mitigated by the multiplexed NM space design in Phoenix. The over-fetching issue results in excessive NM and FM traffic as well. In general, cache mode designs incur higher NM traffic and lower FM traffic than memory mode designs. This is because cache mode designs need to fetch all requested data from FM to NM while memory mode

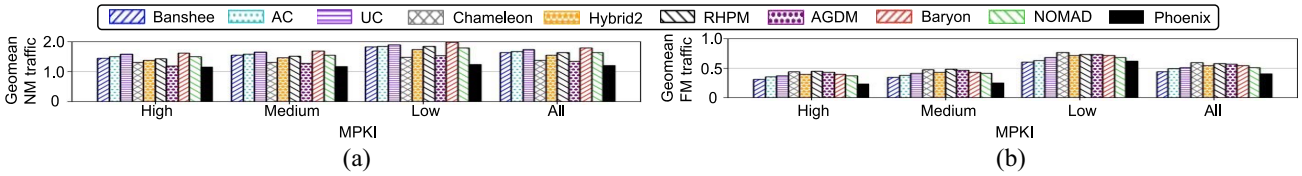

Fig. 18. (a) NM traffic and (b) FM traffic of Phoenix and state-of-the-art designs, normalized to BASE.

designs only migrate data for future reuse. RHPM and AGDM employ the slow swap mechanism for data migration in hybrid memory systems, which generates more data migration traffic. The memory traffic reduction in Phoenix is mainly attributed to lowering the eviction frequency for pages in cNM and mNM, preventing data with a low access frequency into NM, alleviating the over-fetching risk, and reducing the data movement overhead for mode switch between cNM and mNM. Apart from the system performance gains achieved by the memory traffic reduction, the write durability issue [47], [48] for PM products in DRAM-PM hybrid memory systems is alleviated as well.

V. Related Work

In general, prior works can be divided into three categories: utilizing NM in cache mode (cNM), memory mode (mNM), and hybrid mode (both cNM and mNM).

NM in Cache Mode: Prior cNM designs aim to mitigate the metadata overhead for data caching and enhance the hit rate of cNM. AC [11] utilizes a direct mapped design with 64-byte cache lines and eliminates the tag serialization delay by streaming tag and data together in a single burst. A simple and highly effective memory access predictor is employed to service cache misses faster without waiting for a cache miss detection. UC [21] is a page-based four-way set-associative cache design with an LRU replacement policy that uses a footprint predictor to improve the hit rate of cNM. Moreover, it uses a way predictor to avoid the serialization latency of tags and data accesses. Banshee [22] leverages OS page tables and TLBs to locate data in cNM and proposes a bandwidth-aware replacement policy to balance bandwidth utilization. Information of recently inserted or replaced pages in cNM is cached in an added SRAM structure called Tag Buffer. NOMAD [13] utilizes OS page tables and TLBs to manage tags and added hardware to perform data caching. By decoupling the tag and data management, on a DC miss, the OS updates a tag and immediately resumes an application thread without waiting for the cache fill to complete. Instead, the added hardware handles the cache fill without blocking the application thread.

NM in Memory Mode: State-of-the-art mNM designs intend to mitigate the metadata overhead for data remapping and make predictions for potentially reused data. Chameleon [2] modifies the instruction set architecture (ISA) to enable the operating system to inform the added hardware of page allocations and frees. The same as POM [3], Chameleon adopts the segment restricted remapping mechanism for data remapping, resulting in uneven NM utilization rates in different remapping sets and frequent segment migration. Chameleon utilizes fast swap for data migration. RHPM [6] is a hardware-based page remapping design adopting the segment restricted remapping mechanism. Page migration in a remapping set is decided by the relative page hotness. Metadata is placed in NM and cached in added hardware. In the case of metadata cache misses, RHPM makes predictions for the location of requested data. AGDM [5] is designed based on RHPM with the added option to migrate data in two granularities. Memory footprint predictions are made for potentially recurring discrete memory access patterns during runtime. Both RHPM and AGDM employ slow swap for data migration.

NM in Hybrid Mode: The aim of hybrid mode designs is to combine the advantages of both cNM and mNM. Hybrid2 [23] is a hardware-based design using a fixed small fraction of NM as cNM and the remaining NM as mNM. Metadata for hybrid memory management is stored in NM and cached in added hardware. The cNM performs as a staging area to select the data most suitable for migration. Data in cNM is evicted to mNM or FM based on the migration traffic overhead. Hybrid2 performs fast swap for data migration between FM and mNM. Baryon [24] is a hardware-based design leveraging memory compression and data sub-blocking techniques to improve the utilization of NM capacity and FM bandwidth. A small NM area is reserved to efficiently manage and stabilize the irregular and frequently varying data layouts. Data migration between FM and mNM is performed by using slow swap. In both Hybrid2 and Baryon, the ratio of cNM to mNM is fixed and the space of cNM and mNM is separate.

VI. Conclusion

This article presents Phoenix, a novel hybrid memory system that combines caching and migration. The ratio of cNM to mNM is adjustable to exploit both temporal and spatial locality benefits. Phoenix lowers the over-fetching risk and reduces the data movement cost for mode switch between cNM and mNM. A lightweight metadata prefetcher is employed to improve the hit rate of the metadata cache and a fast&slow swap mechanism is adopted to mitigate the metadata overhead while maintaining high swap efficiency. In our evaluations, Phoenix outperforms state-of-the-art designs by an average of 18.2% and consumes orders of magnitude less metadata storage space.

References

[1] L. Song, Y. Chi, L. Guo, and J. Cong, "Serpens: A high bandwidth memory based accelerator for general-purpose sparse matrix-vector multiplication," in *Proc. 59th ACM/IEEE Design Autom. Conf.*, 2022, pp. 211–216.

[2] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "CHAMELEON: A dynamically reconfigurable heterogeneous memory system," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2018, pp. 533–545.

[3] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked DRAM as part of memory," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 13–24.

[4] J. H. Ryoo, M. R. Meswani, A. Prodromou, and L. K. John, "SILC-FM: Subblocked interleaved cache-like flat memory organization," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 349–360.

[5] Z. Peng, D. Feng, J. Chen, J. Hu, and C. Huang, "AGDM: An adaptive granularity data migration strategy for hybrid memory systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2023, pp. 1–6.

[6] Z. Peng, D. Feng, J. Chen, J. Hu, and C. Huang, "RHPM: Using relative hotness to guide page migration for hybrid memory systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 8, pp. 2514–2526, Aug. 2023.

[7] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "MemPod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 433–444.

[8] A. Kokolis, D. Skarlatos, and J. Torrellas, "PageSeer: Using page walks to trigger page swaps in hybrid memory systems," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 596–608.

[9] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "LLC-guided data migration in hybrid memory systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2019, pp. 932–942.

[10] P. Behnam and M. N. Bojnordi, "RedCache: Reduced DRAM caching," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[11] M. K. Qureshi and G. H. Loh, "Fundamental latency tradeoff in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 235–246.

[12] M. N. Bojnordi and F. Nasrullah, "ReTagger: An efficient controller for DRAM cache architectures," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[13] Y. Kim, H. Kim, and W. J. Song, "NOMAD: Enabling non-blocking OS-managed DRAM cache via tag-data decoupling," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2023, pp. 193–205.

[14] F. Zhou, S. Wu, J. Yue, H. Jin, and J. Shen, "Object fingerprint cache for heterogeneous memory system," *IEEE Trans. Comput.*, vol. 72, no. 9, pp. 2496–2507, Sep. 2023.

[15] Y. Tan et al., "GATLB: A granularity-aware TLB to support multi-granularity pages in hybrid memory system," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2022, pp. 903–908.

[16] T. Lee, S. K. Monga, C. Min, and Y. I. Eom "MEMTIS: Efficient memory tiering with dynamic page classification and page size determination," in *Proc. 29th Symp. Oper. Syst. Princ.*, 2023, pp. 17–34.

[17] M. Babaie, A. Akram, and J. Lowe-Power, "Enabling design space exploration of DRAM caches for emerging memory systems," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2023, pp. 340–342.

[18] J. Hong, S. Cho, G. Park, W. Yang, Y.-H. Gong, and G. Kim, "Bandwidth-effective DRAM cache for GPUs with storage-class memory," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2024, pp. 139–155.

[19] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM caches for servers: Hit ratio, latency, or bandwidth? Have it all with footprint cache," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 404–415, 2013.

[20] Y. Lee et al., "A fully associative, tagless DRAM cache," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 3S, pp. 211–222, 2015.

[21] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 25–37.

[22] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient DRAM caching via software/hardware cooperation," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 1–14.

[23] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Hybrid2: Combining caching and migration in hybrid memory systems," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 649–662.

[24] Y. Li and M. Gao, "Baryon: Efficient hybrid memory management with compression and sub-blocking," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2023, pp. 137–151.

[25] Y. Hua, S. Zheng, J. Yin, W. Chen, and L. Huang, "Bumblebee: A MemCache design for die-stacked and off-chip heterogeneous memory systems," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[26] Y. Sun et al. "Demystifying CXL memory with genuine CXL-ready systems and devices," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2023, pp. 105–121.

[27] W. Liu, X. He, and Q. Liu, "Exploring memory access similarity to improve irregular application performance for distributed hybrid memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 797–809, Mar. 2023.

[28] A. Sodani et al., "Knights landing: Second-generation Intel Xeon Phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar./Apr. 2016.

[29] C. Giannoula et al. "DaeMon: Architectural support for efficient data movement in fully disaggregated systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 7, no. 1, pp. 1–36, 2023.

[30] B. Cook, T. Kurth, B. Austin, S. Williams, and J. Deslippe, "Performance variability on Xeon Phi," in *Proc. Int. Conf. High Perform. Comput.*, 2017, pp. 419–429.

[31] G. Kaur, R. Arora, and S. S. Panchal, "Implementation and comparison of direct mapped and 4-way set associative mapped cache controller in VHDL," in *Proc. 8th Int. Conf. Signal Process. Integr. Netw. (SPIN)*, 2021, pp. 1018–1023.

[32] N. Jain, S. Mittal, and P. Ahlawat, "Reducing conflict misses using fraction associative mapping," in *Proc. 2nd IEEE Int. Conf. Parallel, Distrib. Grid Comput.*, Solan, India, 2012, pp. 349–354.

[33] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "A cache design for probabilistically analysable real-time systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2013, pp. 513–518.

[34] *High Bandwidth Memory (HBM) DRAM*, JEDEC Standard JESD235D. Accessed: Dec. 26, 2023. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd235a.

[35] (Micron Technol., Inc., Boise, ID, USA). *DDR4 Datasheet*. Micron. Accessed: Dec. 26, 2023. [Online]. Available: https://www.micron.com/products/dram/ddr4-sdram/part-catalog/mt40a1g8sa-062e

[36] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. IEEE Hot Chips 23 Symp. (HCS)*, 2011, pp. 1–24.

[37] "NAS benchmark." NAS. Accessed: Nov. 2, 2023. [Online]. Available: https://goo.gl/jQvMKbl

[38] "Stream benchmark." STREAM. Accessed: Nov. 2, 2023. [Online]. Available: https://www.cs.virginia.edu/stream/

[39] S. Singh and M. Awasthi, "Memory centric characterization and analysis of SPEC CPU2017 suite," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2019, pp. 285–292.

[40] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did SPEC CPU 2017 broaden the performance horizon?" in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 271–282.

[41] S. Song et al. "Experiments with SPEC CPU 2017: Similarity, balance, phase behavior and SimPoints," Dept. Electr. Comput. Eng., Univ. Texas Austin, Austin, TX, USA, Rep. TR-180515-01, 2018.

[42] S. Tsukada, H. Takayashiki, M. Sato, K. Komatsu, and H. Kobayashi, "A metadata prefetching mechanism for hybrid memory architectures," *IEICE Trans. Electron.*, vol. 105, no. 6, pp. 232–243, 2022.

[43] (Intel, Santa Clara, CA USA). *Intel Optane DC Persistent Memory*. (2020). [Online]. Available: Dec. 25, 2023. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

[44] G. W. Burr et al., "Phase change memory technology," *J. Vac. Sci. Technol. B*, vol. 28, no. 2, pp. 223–262, 2010.

[45] J. Condit et al., "Better I/O through byte-addressable, persistent memory," in *Proc. ACM SIGOPS 22nd Symp. Oper. Syst. Princ.*, 2009, pp. 133–146.

[46] H. Li et al., "Pond: CXL-based memory pooling systems for cloud platforms," in *Proc. 28th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, vol. 2, 2023, pp. 574–587.

[47] Y. Hua, K. Huang, S. Zheng, and L. Huang, "PMSort: An adaptive sorting engine for persistent memory," *J. Syst. Archit.*, vol. 120, Nov. 2021, Art. no. 102279.

[48] Y. Chen, J. Shu, J. Ou, and Y. Lu, "HiNFS: A persistent memory file system with both buffering and direct-access," *ACM Trans. Storage*, vol. 14, no. 1, pp. 1–30, 2018.

[49] Y. Hua, K. Huang, S. Zheng, and L. Huang, "Redesigning the sorting engine for persistent memory," in *Proc. 26th Int. Conf. Database Syst. Adv. Appl.*, 2021, pp. 393–412.
[50] E. H. M. Cruz, M. Diener, M. A. Z. Alves, L. L. Pilla, and P. O. A. Navaux, "Optimizing memory locality using a locality-aware page table," in *Proc. IEEE 26th Int. Symp. Comput. Archit. High Perform. Comput.*, 2014, pp. 198–205.
[51] Y. Feng and E. D. Berger, "A locality-improving dynamic memory allocator," in *Proc. Workshop Memory Syst. Perform.*, 2005, pp. 68–77.
[52] B. Goglin, "Memory footprint of locality information on many-core platforms," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2018, pp. 1283–1292.
[53] M. A. Heroux et al., "Improving performance via mini-applications," Sandia Nat. Lab., Albuquerque, NM, USA, Rep. SAND2009-5574X, 2009.
[54] C. C. Chou, A. Jaleel, and M. K. Qureshi, "CAMEO: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2014, pp. 1–12.
[55] A. Akram, "The feasibility of utilizing low-power DRAM in disaggregated memory systems," in *Proc. Int. Symp. Memory Syst.*, 2024, pp. 1–3.
[56] S. Van Doren, "HOTI 2019: Compute express link," in *Proc. IEEE Symp. High-Perform. Interconnects (HOTI)*, 2019, p. 18.
[57] J. A. Mandelman et al., "Challenges and future directions for the scaling of dynamic random-access memory (DRAM)," *IBM J. Res. Develop.*, vol. 46, no. 2.3, pp. 187–212, Mar. 2002.
[58] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay "AIFM: High-Performance, Application-Integrated far memory," in *Proc. 14th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2020, pp. 315–332.
[59] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.
[60] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[61] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan.–Jun. 2011.

Yifan Hua (Student Member, IEEE) is currently pursuing the Ph.D. degree with Shanghai Jiao Tong University, Shanghai, China.

His research interests include nonvolatile memory systems, processing-in-memory systems, and hybrid memory management.

Shengan Zheng received the B.S. and Ph.D. degrees from Shanghai Jiao Tong University, Shanghai, China, in 2014 and 2019, respectively.

He is currently an Assistant Professor with Shanghai Jiao Tong University. His research interests include memory systems, storage systems, and distributed systems.

Weihan Kong is currently pursuing the Ph.D. degree with Shanghai Jiao Tong University, Shanghai, China.

His research interests include hybrid memory system and near data processing.

Cong Zhou is currently pursuing the Ph.D. degree with Shanghai Jiao Tong University, Shanghai, China.

His research interests include near-memory computing and distributed memory systems.

Linpeng Huang (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from Shanghai Jiao Tong University, Shanghai, China, in 1989 and 1992, respectively.

He is a Professor of Computer Science with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include distributed systems and service-oriented computing.