# Exploring Memory Access Similarity to Improve Irregular Application Performance for Distributed Hybrid Memory Systems

Wenjie Liu, *Student Member, IEEE*, Xubin He, *Senior Member, IEEE*, and Qing Liu, *Member, IEEE*

**Abstract**—With the increasing problem complexity, more irregular applications are deployed on high-performance clusters due to the parallel working paradigm, and yield irregular memory access behaviors across nodes. However, the irregularity of memory access behaviors is not comprehensively studied, which results in low utilization of the integrated hybrid memory system compositing of stacked DRAM and off-chip DRAM. To address this problem, we devise a novel method called *Similarity-Managed Hybrid Memory System* (*SM-HMS*) to improve the hybrid memory system performance by leveraging the memory access similarity among nodes in a cluster. Within *SM-HMS*, two techniques are proposed, *Memory Access Similarity Measuring* and *Similarity-based Memory Access Behavior Sharing*. To quantify the memory access similarity, memory access behaviors of each node are vectorized, and the distance between two vectors is used as the memory access similarity. The calculated memory access similarity is used to share memory access behaviors precisely across nodes. With the shared memory access behaviors, *SM-HMS* divides the stacked DRAM into two sections, the *sliding window section* and the *outlier section*. The shared memory access behaviors guide the replacement of the *sliding window section* while the *outlier section* is managed in the LRU manner. Our evaluation results with a set of irregular applications on various clusters consisting of up to 256 nodes have shown that *SM-HMS* outperforms the state-of-the-art approaches, *Cameo*, *Chameleon*, and *Hyrbid2*, on job finish time reduction by up to $58.6\%$, $56.7\%$, and $31.3\%$, with $46.1\%$, $41.6\%$, and $19.3\%$ on average, respectively. *SM-HMS* can also achieve up to $98.6\%$ ($91.9\%$ on average) of the ideal hybrid memory system performance.

**Index Terms**—Cluster, irregular application, memory system, DRAM, hybrid memory system

## 1 Introduction

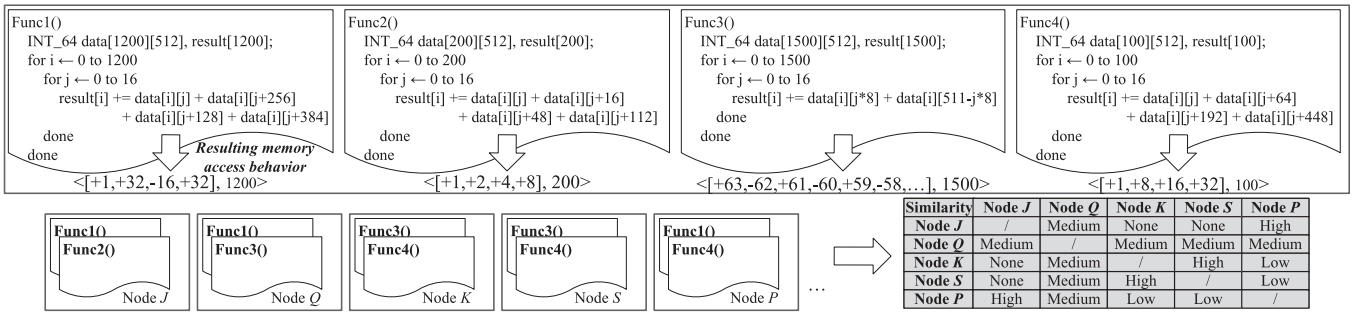
High-performance clusters are widely deployed to process massive data with the parallel working paradigm. A common yet simplistic cluster workload is that all work nodes are assigned to the same job executing the same binary code with a different yet typically equal portion of data [1]. However, for a large portion of irregular applications (e.g., adaptive mesh refinement [2]), the application is divided into smaller tasks and the assigned task for each work node can be interpreted as the combination of various functions and data structures, exhibiting more complex and heterogeneous memory access behaviors (*MAB*s) across nodes. Despite different tasks running among nodes, some common functions and data structures are shared among work nodes, and either execution shared functions or accessing shared data structures yield similar *MAB*s across nodes [3], [4], which indicates various levels of memory access similarity among work nodes running the same irregular application.

Targeting the well-known "memory wall" problem [5], the stacked DRAM is integrated within the processor package via the 3D-stacking technology, providing substantially higher bandwidth and lower access latency than the off-chip DRAM. However, limited by the processor's die size and heating problem, the stacked DRAM usually works cooperatively with the off-chip DRAM, and the stacked DRAM utilization determines the overall performance of the hybrid memory system [6]. To utilize the stacked DRAM efficiently, two challenges exist. The first challenge is to identify *hot* data. Existing approaches identify the *hot* data by exploiting either the access history of each data block [7], [8], [9] or hints provided by the operating systems [10], [11], which ignore the shared *MAB*s enabled by the parallel working paradigm of the cluster and may not perform well in the cluster environment. The second challenge is to manage metadata efficiently. Large bodies of work address the metadata management by using different data granularities for data movement [7], [8], [9], [12], [13], managing the stacked DRAM as a cache [12], [13] or part of the memory space [14], or dynamically shifting between as-a-cache and part-of-memory [7], [8].

To improve the hybrid memory system utilization, we devise a *Similarity-Managed Hybrid Memory System* (*SM-HMS*) by taking advantage of the memory access similarity of irregular applications in the cluster. When running an irregular application, the task assigned to each node is represented as the combination of different functions, and each

- *Wenjie Liu and Xubin He are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA. E-mail: {wenjie.liu, xubin.he}@temple.edu.*
- *Qing Liu is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07670 USA. E-mail: qliu@njit.edu.*

*Manuscript received 8 May 2022; revised 7 November 2022; accepted 3 December 2022. Date of publication 8 December 2022; date of current version 19 January 2023.*
*This work was supported in part by the US National Science Foundation under Grants 2134203, 1828363, 1812861, 2134202, and 2144403.*
*(Corresponding author: Wenjie Liu.)*
*Recommended for acceptance by D. Li.*
*Digital Object Identifier no. 10.1109/TPDS.2022.3227544*




| Similarity | Node *J* | Node *Q* | Node *K* | Node *S* | Node *P* |
|---|---|---|---|---|---|
| **Node *J*** | / | Medium | None | None | High |
| **Node *Q*** | Medium | / | Medium | Medium | Medium |
| **Node *K*** | None | Medium | / | High | Low |
| **Node *S*** | None | Medium | High | / | Low |
| **Node *P*** | High | Medium | Low | Low | / |

Fig. 1. An example of memory access similarity among work nodes. Each work node executes a different portion of the shared binary code, creating various combinations of *MAB*s. As shown, the similarity between two nodes is estimated by the overlapped *MAB*s along with the number of pages covered by each *MAB*, e.g., node $K$ and node $S$ share the same *MAB*s and yield high similarity.

function yields different *MAB*s. Thus, nodes with more shared functions exhibit higher similarity in *MAB*s. Sharing such *MAB*s enables accurate and informed data placement in the hybrid memory system. Based on this, two techniques are included in the *SM-HMS*, *Memory Access Similarity Measuring* and *Similarity-based Memory Access Behavior Sharing* (*SMABS*). *SM-HMS* quantifies the memory access similarity by vectoring *MAB*s of each node and measures the distance between vectors as the memory access similarity, while *SMABS* shares *MAB*s according to the calculated memory access similarity and uses the shared *MAB*s as "recipes" to perform accurate and informed cache replacement for the stacked DRAM cache. In doing so, the stacked DRAM utilization can be significantly improved, alleviating the "memory wall" problem and thus increasing the cluster performance. In summary, we make the following contributions in this study:

- We propose a *Similarity-Managed Hybrid Memory System* (*SM-HMS*) method to address the under-utilized hybrid memory systems by leveraging the various degree of memory access similarity observed in the irregular applications running on the cluster.
- Inside *SM-HMS*, two techniques, *Memory Access Similarity Measuring* (*MASM*) and *Similarity-based Memory Access Behavior Sharing* (*SMABS*), are included to obtain and utilize the memory access similarity among nodes. *MASM* quantifies the memory access similarity by measuring the distance among *MAB* vectors, while the *SMABS* shares *MAB*s across nodes according to the memory access similarity and utilizes the shared *MAB*s as "recipes" to perform accurate and informed stacked DRAM cache replacement.
- We conduct extensive experiments to evaluate the efficacy of *SM-HMS* in a cluster of various configurations consisting of up to 256 nodes with a variety of workloads and the results have demonstrated that *SM-HMS* can effectively improve the stacked DRAM utilization among work nodes and improve overall performance. Moreover, our *SM-HMS* outperforms state-of-the-art approaches *Cameo*, *Chameleon*, and *Hybrid2*, in terms of stacked DRAM utilization and overall performance.

## 2 Background

In this section, we first investigate memory access characteristics of irregular HPC applications, followed by the design challenges of the hybrid memory system.

### 2.1 Memory Access Behaviors in Clusters

A simplistic cluster workload is that all work nodes assigned to the same job are executing the same binary code with a different equal portion of data [1]. As observed in a recent study [3], MABs are identical across nodes since all nodes execute the same binary code. However, memory access is more complex for a large portion of irregular applications (e.g., adaptive mesh refinement [2]), where the submitted job is divided into multiple tasks and tasks running on each node are different. As shown in Fig. 1, four functions are mixed among five work nodes compositing different tasks, and different mixed functions executed on each work node lead to various combinations of *MAB*. By capturing the *MAB* at the page level, multiple pages related to the same function show identical access behavior in terms of cache line offset deltas [15], [16], e.g., 1200 pages related to $Func1$ exhibit an access behavior $[+1, +32, -16, +32]$. Then the *MAB* of $Func1$ can be represented with a tuple $< Sequence,\ Sequence\ coverage >$, where $Sequence$ is a list of the frequently used cache line offset deltas, and $Sequence\ coverage$ is the number of pages following such a sequence. From the view of each work node, the *MAB* of a node can be viewed as the combined *MAB* of each page since the memory page is the basic memory allocation unit, e.g., the *MAB* for $Node\ J$ can be described as the combined *MAB*s from both $Func1$ and $Func2$. Moreover, more overlapped combinations of functions result in higher similarity as more pages across nodes share the same *MAB*.

*A Case Study on the Adaptive Mesh Refining.* The Adaptive Mesh Refining workload represents a typical irregular application, as different tasks are running on each node despite the shared exactly same binary code across work nodes. At the initial stage, the entire dataset is divided into equal portions which are later assigned to each work node along with the shared binary code to maximum the parallelism among all nodes. During the run-time, each work node traverses the assigned dataset, and performs finer granularity of refinement once a region of interest detected. With the changes from both data access granularity and different processing methods invoked, the *MAB* changes accordingly. With the finer data access granularity, the strides exhibited between consecutive memory requests can be different, as the region of interest requires a mesh refinement at a higher density. Moreover, different algorithms may be used to handle the mesh refinement with different granularity, which could changes the *MAB*s of each node fundamentally. Additional, the job scheduler is not aware of the data variance

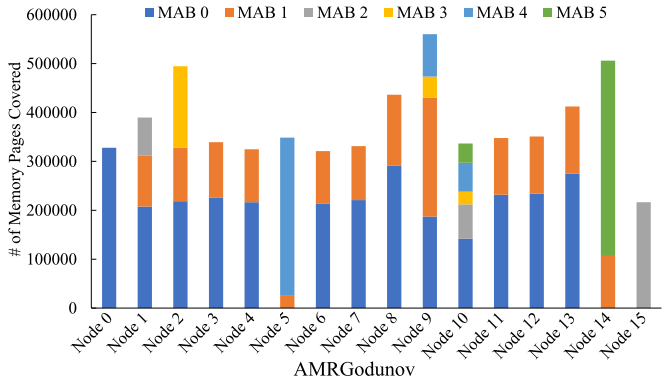



Fig. 2. Breakdown on memory pages covered by the most frequently used 6 *MAB*s. Three observations can be drawn. First, an *MAB*, e.g., $MAB\ 0$, can be shared by all nodes as executing the same binary code. Second, identical $MAB$ combination is observed on node $3, 4, 6, 7, 8, 11, 12$ and 13, as such nodes may execute the same task. Third, $MAB\ 5$ only shows on node 14 and a large amount of pages are covered by $MAB\ 5$, implying the similarity not only exists among nodes but also across multiple pages within a node.

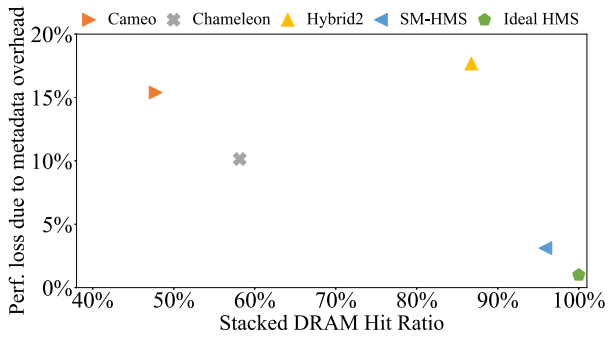



Fig. 3. Comparison on performance overhead and stacked DRAM utilization among state-of-the-art hybrid memory system designs. As shown, existing approaches either yield low utilization of the integrated stacked DRAM or suffer from significant performance caused by metadata querying.

and distributes tasks among nodes by assigning dataset with equal portions, indicating that it is hard to address the irregular *MAB*s via an advanced scheduling policy.

On the other hand, since functions included in the shared binary code is finite in a given Adaptive Mesh Refining workload, some of the *MAB*s commonly exist in multiple nodes as the overlapped functions are executed among nodes, which is observed in Fig. 2. Fig. 2 gives a breakdown of the number of memory requests covered by the most frequently used 6 *MAB*s of each work node running an irregular application. As shown, different combinations of *MAB*s can be observed among work nodes, indicating the data assigned on each node requires various levels of refinement and different functions are called, which leads to irregular memory access behaviors. However, a portion of *MAB*s are shared among nodes, e.g., $MAB\ 0$ and $MAB\ 1$, which evidences that the level of memory access similarity still exists at a random level between any two nodes in the cluster. It is worth mentioning that the number of pages covered by a shared *MAB* varies among nodes since the corresponding function processes different amounts of data among such nodes.

### 2.2 Challenges in Hybrid Memory Systems

To exploit the high performance enabled by the stacked DRAM, two challenges are at the core of designing an efficient hybrid memory system.

The first challenge is to determine the optimal data placement at run-time. Due to the imbalanced performance between stacked DRAM and off-chip DRAM, an intuitive approach to utilizing the high-performance stacked DRAM is to put the soon-to-be-accessed data in the stacked DRAM. The more memory requests served by stacked DRAM, the higher performance can be achieved. A majority of works have focused on using historical memory accesses to guide the data placement for future memory accesses [7], [8], [9]. However, a large portion of HPC applications' memory footprint is allocated to memory pages holding data, which is only accessed for a limited time [17]. Furthermore, the observed access history may be either too short to guide data placement accurately or useless since pages are seldomly accessed afterward.

The second challenge is to reduce the overhead caused by metadata. Since the data is distributed between stacked DRAM and off-chip DRAM, it is vital to determine the data location before access, and the metadata is used to track the location of the requested data. As accessing the metadata stands in the critical path of memory access, the performance overhead caused by metadata access significantly impacts the performance of hybrid memory systems. On the other hand, the metadata storage overhead can be as huge as several hundreds of megabytes, and the recorded access history aggregates the storage overhead depending on the management data granularity [17]. The two challenges lead to a trade-off between the data granularity and metadata overhead, as more precise data placement incurs data movement at a finer granularity between stacked DRAM and off-chip DRAM, which exacerbates the metadata overhead. Fig. 3 compares the stacked DRAM hit ratio and the accordingly performance degradation caused by metadata for the state-of-the-arts. As shown, the state-of-the-art approaches keep improving the stacked DRAM hit ratio. At the same time, the performance overhead caused by metadata increases dramatically for the irregular HPC applications, which diminishes the performance gain enabled by the stacked DRAM.

## 3 Design of the Similarity-Managed Hybrid Memory System

This section presents the detailed design of *Similarity-Managed Hybrid Memory System* (*SM-HMS*). First, we give a high-level overview of *SM-HMS* along with the two workflows, *Memory Access Similarity Measuring* (*MASM*) and *Similarity-based Memory Access Behavior Sharing* (*SMABS*). Then, we dive into the discussion of the three components that enable the *SM-HMS*. For readability, Table 1 summarizes the frequently used abbreviations.

### 3.1 *SM-HMS* Overview

Fig. 4 provides a topological view of an *SM-HMS* enabled cluster. To enable *SM-HMS*, three components, namely *Similarity Monitor*, *Access Behavior Buffer*, and *Swap Executor*, are added to each node of the cluster at different levels. All three components work cooperatively to enable two core

TABLE 1
Frequently Used Abbreviations

| Abbreviation | Description |
|---|---|
| *MAB* | Memory Access Behavior |
| *GMABV* | Global Memory Access Behavior Vector |
| *PMABV* | Per-node Memory Access Behavior Vector |

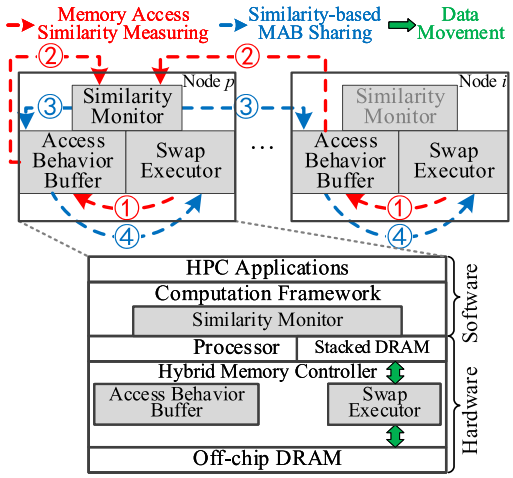



Fig. 4. A schematic view of *SM-HMS* with $n$ nodes (Node $p$ is selected to maintain memory access similarity). As shown, three components work cooperatively to perform *Memory Access Similarity Measuring* (*MASM*) and *Similarity-based Memory Access Behavior Sharing* (*SMABS*).

functions, *Memory Access Similarity Measuring* and *Similarity-based Memory Access Behavior Sharing*.

*Memory Access Similarity Measuring.* To characterize the task assigned on each work node, the memory access similarity is used to describe the relationship between two nodes from the perspective of memory access behavior. Furthermore, the *Swap Executor* is used to monitor the frequently used *MAB*s for each work node (step ①). Then, the observed *MAB*s are submitted to the *Similarity Monitor* for updating the memory access similarity among nodes (step ②). By converging *MAB*s across nodes, the *Similarity Monitor* calculates the similarity between nodes and maintains a *Similarity Matrix* for the similarity-based *MAB* sharing.

*Similarity-Based Memory Access Behavior Sharing.* With the similarity among nodes calculated, the *Similarity Monitor* groups work nodes with high similarity and enable accurate *MAB* sharing (step ③), which alleviates the unnecessary data movement caused by inaccurate sharing in the scenario of the irregular application. The shared *MAB*s are buffered by the *Access Behavior Buffer*, and guide the data placement by the *Swap Executor* (step ④).

## 3.2 *Similarity Monitor*

To investigate the irregularity behavior of the irregular applications, the *Similarity Monitor* is added into the cluster and interprets the similarity of tasks performed on each node as the similarity among *MAB*s shown on each node. To enable this, the responsibility of *Similarity Monitor* is divided into two major workflows, *Memory Access Similarity Measuring* (*MASM*) and *Similarity-based Memory Access Behavior Sharing* (*SMABS*).

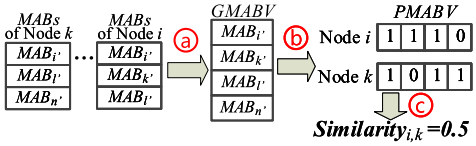



Fig. 5. Procedure of memory access similarity calculation. Each step is described in Section 3.2.1.

### 3.2.1 *Memory Access Similarity Measuring*

To discover the relationship among tasks assigned to each node, we use memory access similarity to represent the difference of tasks among work nodes and propose a quantitative method to measure the memory access similarity among nodes which is described in Fig. 5. According to Fig. 5, *MASM* measures similarity by three steps, converging *MAB*s from all nodes as a *Global Memory Access Behavior Vector* (*GMABV*), generating *Per-node Memory Access Behavior Vector* (*PMABV*), and calculating distance among *PMABV*s as the similarity.

*Step ⓐ: Converging MABs Sent by All Nodes as a GMABV.* Since each work node can execute a portion of the job's binary code, the *MAB*s of one node represent a fraction of the overall memory characteristics of the running job. To accurately compare the similarity among nodes, the *MAB*s obtained from each work node are converged as a *GMABV*, and each element in the *GMABV* represents a unique *MAB*. By doing so, the running job is profiled from the aspect of *MAB*, and the *GMABV* can be used to generate the memory access behavior vector for each work node.

*Step ⓑ: Generating the PMABV for Each Node.*With the converged *GMABV*, *MAB*s generated by each node can be mapped with the *GMABV* and built as a *MAB* vector consisting of all *MAB*s encountered during run-time, namely a *Per-node MAB Vector* (*PMABV*). To generate the *PMABV* of a node, *MASM* cross-references *MAB*s observed on the corresponding node with the generated *GMABV*, and marks each *PMABV* entry as '1' for the existence of the corresponding *MAB* while '0' for absence. It is worth mentioning that multiple *PMABV* designs are applicable to extend the usability, e.g., requests covered by the*MAB*, and using the boolean existence bit can enable quickly determine nodes to share for the *SMABS*. On the other hand, the *Access Behavior Buffer* maintains a capacity-limited buffer to hold the most useful *MAB*s, which guarantees the less effective *MAB*s will soon be evicted. After the *PMABV*s generated, the memory access behavior of each node can be represented as a discrete binary vector, and the similarity calculation can be performed.

*Step ⓒ: Using the Distance Between Two PMABVs as the Similarity.*With *PMABV* generated for each node, the memory access similarity between two nodes can be represented as the distance between two vectors, and various distance measuring methods can be employed to measure the distance between two vectors (e.g., Jensen-Shannon Divergence [18], KL-Divergence [19], and Edit Distance [20]). A larger distance between two vectors indicates lower similarity as fewer *MAB*s are shared between the two nodes, while a smaller distance for higher similarity. The calculated memory access similarity among nodes is maintained by a *Similarity Matrix* (as shown in Fig. 6) within the *Similarity Monitor*. The *Similarity Matrix* is updated during run-

| | Node *1* | Node *2* | Node 3 |
|---|---|---|---|
| Node *0* | 0.42 | 0.51 | 0.59 |
| Node *1* | | 0.58 | 0.68 |
| Node *2* | | | 0.83 |

Fig. 6. An example of *AMRGodunov*'s *Similarity Matrix*. Each entry denotes the similarity between two corresponding nodes.

time to accommodate the changing workset on each node. Fig. 7 gives the workflow of the *Similarity Matrix* update process. When a "pending" *MAB* arrives, the *GMABV* is first checked to verify if such a *MAB* has been shown before. Suppose a *MAB* is found, then the *MAB* is appended to the *GMABV*, which updates the *PMABV* for each node and performs similarity calculation between the sender node and other nodes. Note that the similarity among other nodes will not change as the non-zero part of the corresponding *PMABV* stays the same. Otherwise, the sender node's *PMABV* and the similarity between other nodes will be updated depending on the previous existence of such a *MAB*.

Fig. 8 shows the memory access similarity for both irregular and regular applications. The darker color indicates higher memory access similarity between the corresponding two nodes. As shown, the regular application achieves high similarity among nodes due to the same binary code running on all nodes and yields highly similar access behaviors across nodes. Unlike the regular application, the irregular application exhibits different similarities among nodes. Only a portion of nodes has shown high similarity, while low similarity observed on other nodes.

### 3.2.2 Similarity-Based MAB Sharing

As illustrated in Section 2, the data placement policies of the state-of-the-art hybrid memory system designs heavily rely on the observed *MAB*s, which leads to non-trivial metadata overhead for both storage and access. Moreover, keeping the metadata for all memory pages of the running job is not efficient for both memory system performance and metadata storage, as only the memory pages within the current working set matter. An intuitive method to amortize the metadata overhead is to share *MAB*s among nodes, and the overhead caused by *MAB* monitoring can be amortized among nodes sharing the same *MAB*. However, nodes running the same job may exhibit different combinations of *MAB* in the scenario of irregular applications. Sharing *MAB*s with a random node can result in unnecessary data movement, severely reducing the performance gain enabled by *MAB*s sharing and hurting the system performance.

To better understand the effectiveness of different sharing schemes, Fig. 9 compares two intuitive sharing schemes, *Global Sharing* (*GS*) and *Selective Sharing* (*SS*), in terms of performance overhead and sharing accuracy (defined as the percentage of shared *MAB*s is actually helpful). *Global Sharing* shares all *MAB*s to all nodes in the same job, while *Selective Sharing* only shares *MAB*s with nodes already exhibited such *MAB*s. Comparing both sharing schemes, *Global Sharing* fails to provide high sharing accuracy for the irregular application since not every *MAB*s is helpful to other nodes. In contrast, the *Selective Sharing* shares *MAB*s on demand which incurs performance overhead for the node deployed

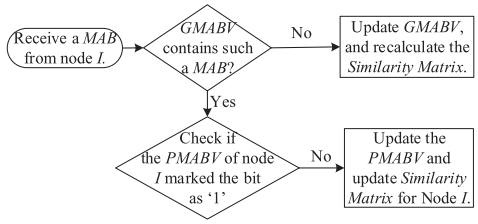



Fig. 7. Procedure of *Similarity Matrix* update. Depending on the existence of the received *MAB*, the *Similarity Monitor* updates a *Similarity Matrix* accordingly.

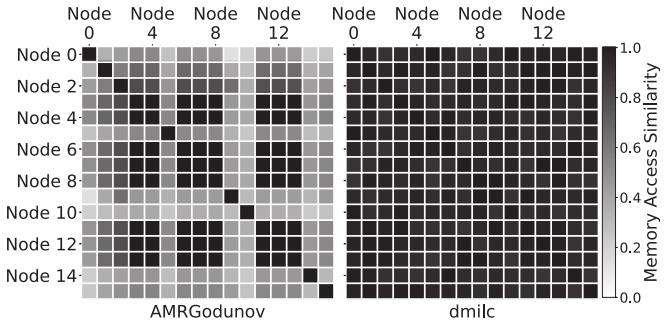



Fig. 8. *Similarity Matrix* of irregular (*AMRGodunov*) and regular (*dmilc*) applications. As shown, *AMRGodunov* shows various memory access similarities among nodes, while *dmilc* exhibits high similarity across nodes.

the *Similarity Monitor*. Due to the inaccurate *MAB*s sharing, both methods suffer from performance loss, with up to $13.1\%$ for *GS* and $15.4\%$ for *SS*, indicating an efficient *MAB* sharing scheme is needed in the scenario of irregular applications.

**Algorithm 1.** Similarity-Based Node Grouping

```
Un-group all nodes
while find an un-grouped node i having similarity larger than SimThreshold with at least one node do
    Mark node i as High Similarity node
    if node i has high similarity with existing groups then
        Assign node i to the found group
    else
        Create a new group G_i
    end if
end while
for AccBhv in GMABV do
    Create a behavior group B_AccBhv
    Assign nodes with access behavior AccBhv and labeling as Low Similarity to group B_AccBhv
end for
```

To accurately share *MAB*s among nodes, the memory access similarity maintained by the *Similarity Matrix* is used to guide the sharing among nodes, and a *Similarity-based Memory Access Behavior Sharing* (*SMABS*) scheme is proposed to achieve both high sharing accuracy and low metadata overhead. Since the memory access similarity marks the percentage of overlapped *MAB*s between two nodes, the accuracy of *MAB* sharing can be estimated by the similarity between sharing/receiving nodes, and sharing *MAB*s among nodes with high similarity yields high sharing accuracy. Based on this, *SMABS* groups nodes with high similarity together and applies *GS* among such nodes, while *Selective*

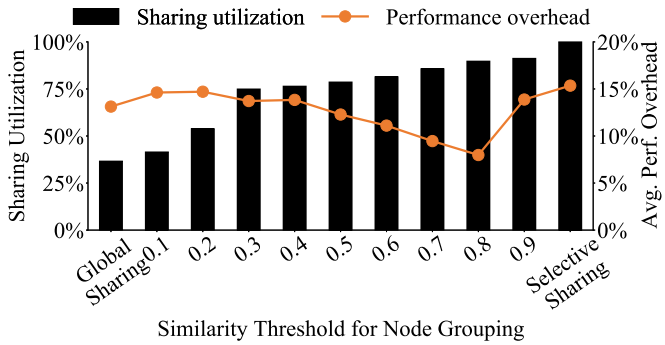



Fig. 9. Comparison between different sharing schemes in terms of sharing utilization and performance overhead. The sharing utilization marks the percentage of useful shared *MAB*s, and can be calculated by $\frac{Useful\ MABs}{Shared\ MABs}$.

*Sharing* is used for the rest of the nodes. With fewer nodes involved in *Selective Sharing*, the overhead caused by *MAB*s querying can be ignored for the node with *Similarity Monitor* enabled. Algorithms 1 and 2 give the pseudo-code of *SMABS* for both *Similarity-based Node Grouping* and *Similarity-based Sharing*. As illustrated, a grouping threshold $SimThreshold$ is defined to balance the sharing accuracy and performance overhead, and the *SMABS* falls into *Global Sharing* when the grouping threshold is set to 0 while 1.0 for *Selective Sharing*. To find a suitable grouping threshold, *SMABS* adjusts $SimThreshold$ dynamically according to feedbacks of the recipient nodes. *SMABS* increases $SimThreshold$ to improve the effectiveness of shared *MAB*s for decreased sharing accuracy while decreasing $SimThreshold$ to maximize performance improvement when the sharing accuracy is increased or unchanged.

**Algorithm 2.** Similarity-Based Sharing

```
Input: Receiving access behavior AccBhv from node k
if AccBhv in GMABV then
  if node k belongs to a Similarity Group then
    Forward AccBhv to nodes within the group with global sharing
  else
    Perform selective sharing with all Low Similarity nodes
  end if
  Update metadata
else
  Update PMABV, GMABV and Similarity Matrix
end if
```

*Scalability of the SM-HMS.* With more work nodes assigned to the workload with a large amount of data, the *SM-HMS* can suffer from performance degradation caused by both increased network traffic from the *MAB* sharing and heavier performance overhead from the *Similarity Monitor*. To extend the *SM-HMS* with more nodes, all work nodes assigned to the same job are divided on a rack basis, and one node in each rack is selected as a local *Similarity Monitor*, which is only responsible for the similarity management of nodes within the same rack. Since the job scheduler is unaware of the data variance on each node, nodes with high similarity can be distributed among multiple racks, which requires the synchronization of memory access similarity among racks to utilize the potential performance improvement enabled by the *SM-HMS*. To sync across racks, each local *Similarity Monitor* treats other ranks as a special node and uses the aggregated *MAB*s to represent the characteristic of all nodes within the corresponding rack. Also, the sharing of *MAB*s will only be issued and received by the local *Similarity Monitor*, which significantly reduces the network traffic caused by the *MAB* sharing among nodes. Moreover, the number of nodes involved in similarity calculation and maintenance is greatly reduced, easing the performance overhead for nodes deployed the *Similarity Monitor*. By doing so, the *SM-HMS* can support more nodes without causing tremendous network traffic and ensure the enabled performance improvement benefits applications with the demand of high scalability.


Fig. 10. An entry of the *Access Behavior Buffer*. As shown, both the *Status* and *Coverage* bits determine if the newly observed *MAB* will be sent to the *Similarity Monitor* for similarity update. And the *Starting Address* and *Covered Pages* are used to quickly identify memory requests can be served by the *sliding window section*.

### 3.3 *Access Behavior Buffer*

According to the design of *SMABS*, only the *MAB*s of the current workset are shared among nodes, which alleviates the overhead caused by *MAB* monitoring on each node. Additionally, multiple memory pages across nodes share the same *MAB* due to the extensive structured data processed by each node and the abundant loops in the HPC applications. Thus, even a single shared *MAB* can describe the access behavior of multiple pages within the corresponding node. Based on such an idea, an *Access Behavior Buffer* is added in the memory controller of each work node to buffer the shared *MAB*s and use the buffered *MAB*s to guide data placement in the hybrid memory system.

Fig. 10 gives the structure of the *Access Behavior Buffer*. As shown, the *MAB* stores a cache line access sequence to guide cache replacement for the covered pages. The *Covered Page* is used to aid the *Swap Executor* in determining if the corresponding *MAB* covers the targeting page. Since the array is widely used in HPC applications, memory pages with identical access behavior are often located closely [21]. The *Starting Address* records the address of the first page covered by the corresponding *MAB*. The uncovered page compares its address with the corresponding *Starting Address* to determine if it is covered or not. The *Status* bits and a *coverage* count are used to mark the metadata of each *Access Behavior Buffer* entry. Depending on the origination of each *Access Behavior Buffer* entry, different behaviors are performed. For *MAB*s shared by other nodes, the status bits are marked as "00". The coverage count provides a feedback to the *Similarity Monitor* for the corresponding *MAB*. The status bits are set to "01" for a *MAB* generated locally, indicating a "pending" *MAB* discovered. Suppose more memory pages are linked to a "pending" *MAB* entry, then the status bits are set to "10" indicating that such a *MAB* is sent to the *Similarity Monitor* for sharing. Additionally, the *Access Behavior Buffer* is managed in an LRU manner, and *MAB*s not used for a long time are evicted to maximize the effectiveness of newly shared or discovered *MAB*s.

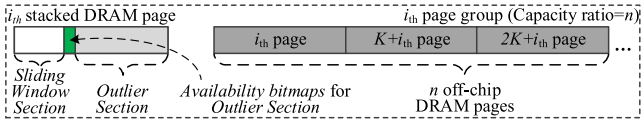



Fig. 11. Memory layout of *SM-HMS*. Every $n$ ($n$ denotes the capacity ratio between off-chip DRAM and stacked DRAM) off-chip pages is mapped with one stacked DRAM page, and the total number of page groups is $K$ ($K$ marks the number of pages within the stacked DRAM). Each stacked DRAM page is divided into two parts, *sliding window section* and *outlier section*.

### 3.4 *Swap Executor*

To utilize *MAB*s buffered in Access Behavior Buffer, a *Swap Executor* is used to perform data placement for the underlying hybrid memory system by using the shared *MAB*s as "recipes".

#### 3.4.1 Memory Layout and Data Placement of SM-HMS

With *MAB*s buffered in the *Access Behavior Buffer*, the *Swap Executor* is designed to perform cache replacement for the stacked DRAM. The stacked DRAM is used as a cache between the last-level cache and the off-chip DRAM in the core of *SM-HMS*. Fig. 11 shows the memory layout of *SM-HMS*. To utilize the stacked DRAM efficiently, $n$ interleaved off-chip DRAM pages are mapped with one stacked DRAM page to form a page group ($n$ denotes the capacity ratio between off-chip DRAM and stacked DRAM). As a shared *MAB* marks the access sequence of cache lines in each covered page, memory requests targeting such pages can be well forecasted and served with stacked DRAM by performing timely cache replacement. Each stacked DRAM page is divided into the *sliding window section* and the *outlier section*. Within a page group, each off-chip page is linked with a cache line within the corresponding stacked DRAM page's *sliding window section*. The *Swap Executor* performs accurate and timely swapping for the *sliding window section* based on the recently accessed cache line and the *MAB* covering the target off-chip page. For requests that do not follow the given *MAB*, the *outlier section* serves such requests and performs replacement with LRU. Since $n$ off-chip pages share the *outlier section*, the capacity assigned to each off-chip page is determined by the Coverage segment of the corresponding entry in the *Access Behavior Buffer*. Pages with a higher amount of requests benefit more from the *outlier section*. Additionally, cache lines retired from the *sliding window section* are swapped to the *outlier section* to increase the chance of stacked DRAM hit [22], and metadata of the *outlier section* is updated accordingly.

#### 3.4.2 Serving Memory Requests

The *Access Behavior Buffer* and *Swap Executor* work cooperatively to quickly determine the data location and reduce unnecessary traffic between stacked DRAM and off-chip DRAM.

The *Access Behavior Buffer* is accessed for each memory request to determine if the targeting page is linked to a buffered *MAB*. Suppose the requesting page is linked with an existing *MAB*. In that case, the corresponding *MAB* and the most recently accessed cache line offset are used to determine whether the about to be accessed cache line resides in the *sliding window section*. For a hit in the *sliding window section*, the *Swap Executor* retires the accessed cache line and moves it to the *outlier section* for future access. The *Swap Executor* chooses and buffers the next to be accessed cache line from either the *outlier section* or the off-chip DRAM following the given *MAB*. Suppose the *sliding window section* cannot satisfy the incoming request. Then, the *outlier section* is accessed to serve the request with the LRU policy.

In case the requesting page is not covered by any buffered *MAB*s, the *Swap Executor* monitors the access behavior of such a page for *MAB* discovering and adds the observed *MAB* to the *Access Behavior Buffer* with status bits of "01" (represents a pending *MAB*). If more pages share the same *MAB*, the *Swap Executor* updates the status bits to "10", and the *Access Behavior Buffer* sends such a *MAB* to the *Similarity Monitor* for both similarity update and *MAB* sharing. For a requesting page not linked with any *MAB* entries, the page address is compared with the *starting address* of entries in the *Access Behavior Buffer*, since memory pages sharing identical MAB are located near each other. Suppose an entry with a nearby starting address is founded. In that case, the *Swap Executor* proactively performs data replacement according to the corresponding *MAB* while keeps monitoring the effectiveness of the preemptive applied *MAB*. Thus, the *MAB* monitoring process is shortened, and pages with fewer accesses can still benefit from the integrated stacked DRAM.

## 4 Evaluation

In this section, we conduct experiments to evaluate the effectiveness of the *SM-HMS*. We first introduce our evaluation methodology, including experimental testbed, related simulation parameters, and workloads. Then we present and discuss the evaluation results using various metrics.

### 4.1 Experimental Methodology

To evaluate the proposed *SM-HMS*, we implement its design in the cycle-accurate DRAMSim2 [23] memory simulator. We use the Simics [24] full-system simulator as the front-end to simulate the cluster and integrate the modified DRAM simulator with Simics. To calculate the memory access similarity, we represent memory access behavior with frequently used cache line offset delta sequence, and employs JS Divergence [18] to calculate the distance between two *PMABVs*. For the computation framework, OpenMPI [25] is used as the computation framework, and some necessary modifications are made to the framework in order to implement the functions provided by the *Similarity Monitor*. For cluster configuration, we use Simics to simulate a cluster with 16 nodes (the node with the smallest node index is selected to enable the *Similarity Monitor*), where each node shares the same configuration as shown in Table 2. The network of the simulated cluster is configured with 10 $GB/s$ bandwidth and 5 $\mu s$ latency. For access overhead of the *Access Behavior Buffer*, CACTI tool [26] estimates the access latency, which assumes a 3.5 $ns$ latency for each *Access Behavior Buffer* access.

For benchmarks, a wide range of irregular HPC applications are selected, including Chombo [27], NASA Parallel Benchmark [28], Graph500 [29], and Coral2 [30]. During the run-time, OpenMPI is used as the computation framework.

TABLE 2
Work Node Configurations

| | |
|---|---|
| Processor | 8 cores, L1 cache 32KB/32KB L2 cache 512KB per core, L3 cache 64MB |
| Memory Controller | 64/64-entry read/write request queue FR-FCFS, writes are scheduled in batches |
| Stacked DRAM | HBM2 with 4GB capacity |
| Off-chip DRAM | DDR4-1600, 2 channels, 1 rank per channel Capacity: 32GB |
| Access Behavior Buffer | Capacity: 512KB Read/Write Latency 3.5ns/3.5ns |
| Network | Bandwidth 10Gb/s with latency 5 $\mu$s |
| Framework | OpenMPI-2.1.6 with OpenMP |

TABLE 3
Benchmark Characteristics

| Benchmark | Benchmark Suite | Memory Footprint (GB) | MPKI |
|---|---|---|---|
| **AMR-G** | Chombo | 31.35 | 25.86 |
| **AMR-P** | Chombo | 30.24 | 11.90 |
| **EBAMR** | Chombo | 19.68 | 28.38 |
| **CG** | NPB | 22.36 | 11.77 |
| **MG** | NPB | 18.91 | 9.04 |
| **UA** | NPB | 14.85 | 14.68 |
| **Graph500** | Graph | 20.61 | 7.38 |
| **LAMMPS** | CORAL 2 | 28.64 | 9.30 |
| **AMG** | CORAL 2 | 24.61 | 5.22 |

Within each node, all processor cores are utilized with the OpenMP to provide parallelism and accelerate the execution of the running application. Table 3 summarizes the ground truth of selected benchmarks, including Missed per Kilo Instructions (MPKI) of the LLC and the size of memory footprints.[1] Before starting the simulation, we fast forward one billion memory accesses for cache warm-up. We then simulate 100 million memory accesses for statistics. To evaluate the effectiveness of *SM-HMS*, a cluster with only off-chip DRAM enabled is used as the baseline (denoted as *Off-chip DRAM only*), and we compare the *SM-HMS* with the state-of-the-art hybrid memory system designs, *Cameo* [7], *Chameleon* [8], and *Hybrid2* [9], to justify whether the single node-based solution is able to improve the performance in a cluster scenario. In addition, we compare the *SM-HMS* with the ideal hybrid memory system (*Ideal HMS*) that all memory requests are served by the stacked DRAM to show the exceptional efficiency of *SM-HMS*.

The evaluation of *SM-HMS* is conducted as follows. For performance improvement, both job finish time and IPC are used to compare the performance improvement of *SM-HMS* with clusters armed with the state-of-the-art hybrid memory system designs along with a cluster with the ideal hybrid memory system deployed. Then, results on stacked DRAM hit ratio, memory access latency, and traffic between stacked DRAM and off-chip DRAM are presented to reason the performance improvement enabled by *SM-HMS*. Also, sensitivity studies are performed to show the adaptiveness of the proposed *SM-HMS* among various cluster configurations. Moreover, experiments on regular applications are conducted to show that the *SM-HMS* can also achieve performance improvement in regular applications.

### 4.2 Job Finish Time & IPC

A reasonable metric to measure performance improvement of a cluster is the finish time of the running application, which is also the finish time of the slowest node in the corresponding cluster. Fig. 12 compares the finish time of each benchmark, where the upper edge of each bar represents the benchmark's finish time while the lower edge for the finish time of the fastest node for each run. As shown, the proposed *SM-HMS* delivers a finish time reduction for up to 67.4% with 61% on average, and

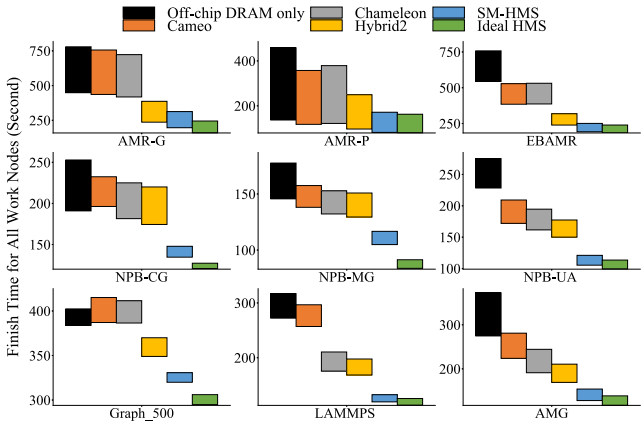



Fig. 12. Comparison on job finish time among the six memory schemes. *SM-HMS* outperforms the three state-of-the-art approaches by 46.1%, 41.6%, and 19.3% on average, respectively.

outperforms the state-of-the-art approaches, *Cameo*, *Chameleon*, and *Hyrbid2*, by up to 58.6%, 56.7%, and 31.3%, with 46.1%, 41.6%, and 19.3% on average, respectively. Moreover, *SM-HMS* achieves a finish time reduction efficiency (91.9% on average) compared with the cluster with an ideal hybrid memory system. The reason behind this lies in three aspects. First, more memory requests can be served by the stacked DRAM, which contributes to an overall performance improvement of the entire cluster. By sharing *MAB*s across nodes, the *sliding window section* serves memory requests following a given *MAB* with a *MAB*-guided cache replacement policy. By doing so, the cache replacement for the stacked DRAM cache is performed accurately, which also reduces unnecessary traffic between the stacked DRAM and off-chip DRAM. Moreover, the shared *MAB*s alleviate the cost of *MAB* discovering for pages with fewer accesses. Memory requests targeting such pages can also benefit from the stacked DRAM, which is hard to realize in the existing state-of-the-art approaches due to insufficient historical memory accesses. For benchmarks with fewer memory requests following the discovered *MAB*s, e.g., *AMRGodunov*, the *outlier section* takes over and serves memory requests with LRU policy. Second, overhead caused by metadata querying is significantly reduced, contributing to the overall cluster performance improvements. Since the *Access Behavior Buffer* only buffers *MAB*s used by each node during run-time, the buffered *MAB*s can deliver an average coverage of 84.1% of the current working set, which indicates that only the *Access Behavior Buffer* will be accessed to acquire the metadata of pages within the current working set. Also, the linkage between a page and a

1. We also present the sensitivity studies with different memory footprint sizes and capacity ratios in Section 4.6 to demonstrate that *SM-HMS* is adaptive to various memory configurations.

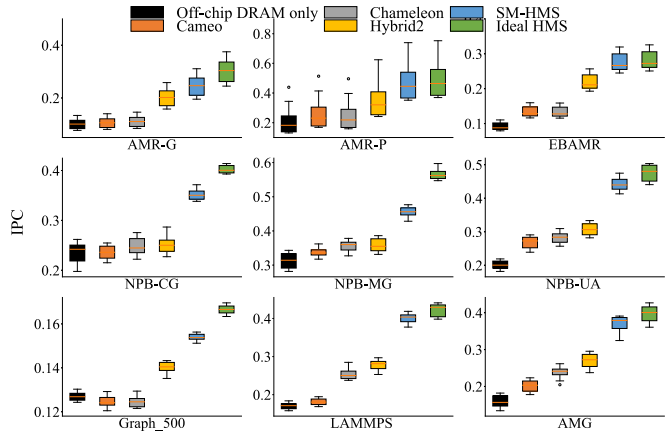



Fig. 13. IPC comparison for the six memory schemes. *SM-HMS* achieves up to 98.6% (91.9% on average) of the performance improvement of the ideal hybrid memory system.

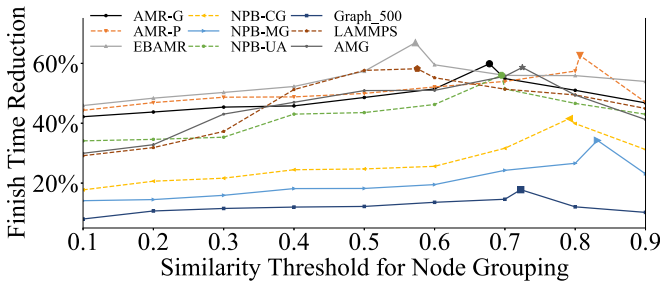



Fig. 14. Finish time reduction with various $SimThreshold$ configurations. As shown, the point with larger marker on each line identifies the corresponding average $SimThreshold$ and the finish time reduction, which outperforms the fixed $SimThreshold$ and indicates the effectiveness of the dynamically adjusting $SimThreshold$.

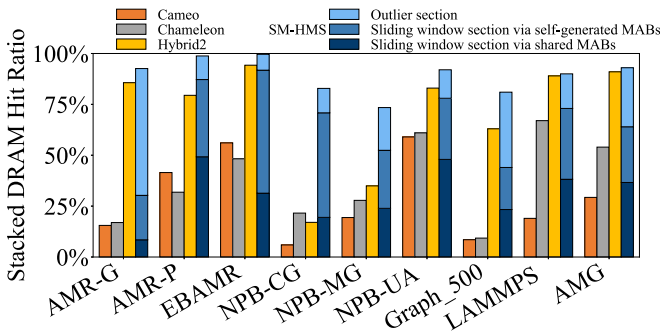



Fig. 15. Stacked DRAM hit ratio for the four hybrid memory system designs. For our *SM-HMS*, we further breakdown the stacked DRAM hit ratio achieved, and *sliding window section* contributes to an average hit ratio of 65.6% while 23.5% for the *outlier section*.

*MAB* is recorded in the *Access Behavior Buffer* to accelerate the *Access Behavior Buffer* querying and minimize the performance overhead caused by metadata querying. On the other hand, the address of an unknown page (the page that does not link to any *MAB*s) is cross-referenced with the *Starting Address* of existing entries in the *Access Behavior Buffer*, and the matched *MAB* will be applied preemptively to accelerate the process of *MAB* matching. By doing so, more memory requests can be quickly benefited by the stacked DRAM, contributing to both stacked DRAM hit ratio and overall performance. Fig. 13 compares the performance improvements for the six memory schemes in terms of IPC. As shown, *SM-HMS* outperforms the three state-of-the-art approaches for up to 2.22 ×, 2.12 ×, and 1.25 ×, with 1.62 ×, 1.55 ×, and 1.16× on average, respectively. Additionally, the *SM-HMS* significantly reduces the metadata overhead (2.1% on average) compared with 7.9%, 6.8%, and 13.1% for the three state-of-the-art solutions. With the combination of the two factors, *SM-HMS* can effectively utilize the stacked DRAM and improve the cluster performance. Last, the dynamically adjusted node grouping tunes the similarity grouping at a finer granularity, which reduces the network traffic caused by *MAB* sharing and relaxes the performance degradation caused by similarity monitoring. To demonstrate the effectiveness of the dynamic adjustment of $SimThreshold$, we compare the dynamically adjusted $SimThreshold$ with fixed values in the term of finish time reduction as shown in Fig. 14. As shown, each line represents the finish time reduction with various fixed $SimThreshold$, and the point with a bigger marker indicates the finish time reduction for the dynamically adjusted $SimThreshold$, which outperforms the fixed $SimThreshold$ configurations. The reason behind this is that the dynamically adjusted $SimThreshold$ can quickly adjust the node grouping according to the changes in memory access characteristics during run-time, which yields a good balance between the sharing accuracy and the performance overhead caused by sharing.

### 4.3 Stacked DRAM Hit Ratio

To better understand the performance improvement of *SM-HMS*, Fig. 15 compares the stacked DRAM hit ratio among *Cameo*, *Chameleon*, *Hybrid2*, and our proposed *SM-HMS*. With the guidance provided by the shared *MAB*s, *SM-HMS* delivers a hit ratio of up to 98.8% with 95.9% on average, which outperforms the three state-of-the-art approaches by 48%, 36.6%, and 8.9% on average. The reason behind is that the shared *MAB*s exploit the connections among memory pages across nodes to enable highly accurate data replacement, which is ignored by existing solutions as they heavily rely on the *MAB* observation for each separate page. The high stacked DRAM hit ratio of *SM-HMS* can be attributed to two aspects. First, using accurately shared *MAB*s as "recipes" guides the cache replacement. The *sliding window section* fulfills such requests with the accurate and informed cache replacement with the corresponding *MAB* for memory requests covered by an *Access Behavior Buffer* entry. Meanwhile, the *outlier section* is allocated to pages based on the number of outlier cache lines within each page, and more outlier cache lines can be cached in the *outlier section*, maximizing the possibility of stacked DRAM hit. Second, pages with fewer accesses can also benefit from the stacked DRAM by observing that *MAB* can be similar among pages located near each other. With the *starting address* of each *Access Behavior Buffer* entry, the incoming memory request may find a match in the *Access Behavior Buffer* and presumably apply the *MAB* to serve more memory requests. By doing so, the previously unable to be benefited memory pages can be served by the stacked DRAM, which enlarges the beneficiaries of the stacked DRAM and improves the overall stacked DRAM hit ratio. Among the three existing approaches, *Hybrid2* achieves a similar stacked DRAM hit ratio as *SM-HMS*, as the dual-layer design keeps the hot data in stacked DRAM longer and yields more stacked DRAM hits. However, the improved hit ratio of *Hybrid2* comes with a penalty caused by metadata

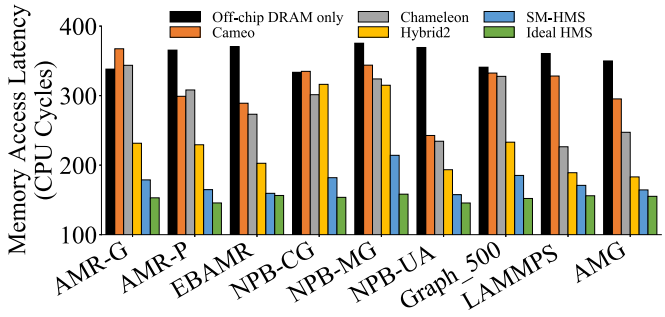



Fig. 16. Comparison on memory access latency for the six memory schemes. *SM-HMS* reduces the memory access latency by up to 51.3%, 47.9%, and 28.1%, with 38.3%, 35.2%, and 20.4% on average compared with the three state-of-the-art approaches, *Cameo*, *Chameleon*, and *Hybrid2*, respectively.

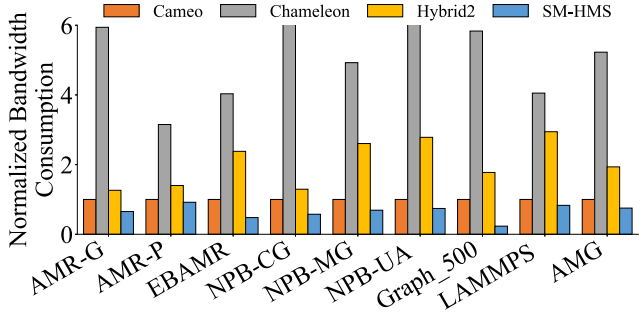



Fig. 17. Normalized bandwidth consumption among *Cameo*, *Chameleon*, *Hybrid2*, and *SM-HMS*.

querying, which prolongs the memory access latency, as shown in Fig. 16.

### 4.4 Memory Access Latency

Fig. 16 gives the memory access latency comparison among the six memory schemes. Compared with the three state-of-the-art approaches, *SM-HMS* reduces the memory access latency by up to 51.3%, 47.9%, and 28.1%, with 38.3%, 35.2%, and 20.4% on average, respectively. The low memory access latency of *SM-HMS* comes from the efficacy of the Access Behavior Buffer. As the *Access Behavior Buffer* only keeps the currently in-use *MAB*s, *MAB*s in the *Access Behavior Buffer* cover 84.1% of memory pages on average, and only 3.2% of memory requests incurring additional metadata queries, while on average 23.7%, 14.8%, and 21.3% incurred by the three state-of-the-art approaches respectively, which explains the high memory access latency for the three existing approaches since more metadata querying sits on the critical path of memory accessing. Additionally, compared with the ideal hybrid memory system, *SM-HMS*'s average memory access latency is higher than the ideal hybrid memory by 12.6%, while 86.7%, 78.2%, and 41.7% for the three state-of-the-art approaches, respectively.

### 4.5 Bandwidth Consumption

Fig. 17 compares the bandwidth consumption (normalized against *Cameo*) between the stacked DRAM and off-chip DRAM among the four hybrid memory system designs. *SM-HMS* achieves the least bandwidth consumption in all applications and reduces the memory traffic compared with the three state-of-the-art approaches by up to 52.1%, 89%, and 79.8%, with 7.1%, 80.5%, and 60.8% on average, respectively. The reduced traffic comes from two aspects. First, cache replacement can be performed accurately with the help of *SMABS*, and the *sliding window section* only caches soon-be-accessed cache lines according to shared *MAB*s. Also, the cache line retired from the *sliding window section* does not get evicted immediately, while it is swapped into the *outlier section* and can be used to serve future requests. Second, stacked DRAM miss will not trigger a cache replacement immediately, while the *Swap Executor* and *Access Behavior Buffer* collaboratively decide the replacement candidate and ensure only useful cache lines will be cached.

### 4.6 Sensitivity Study

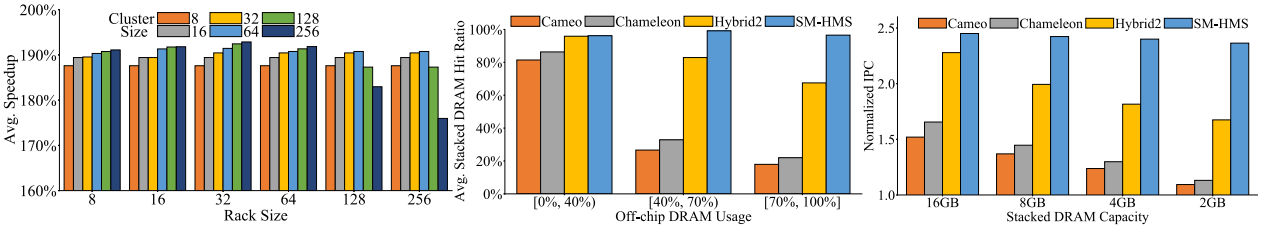
To better understand how *SM-HMS* behaves with various cluster configurations, the sensitive studies on the cluster size, memory footprint and stacked DRAM capacity are conducted to investigate the effectiveness of *SM-HMS*.

#### 4.6.1 Cluster Size

Fig. 18a compares the average performance improvement with various numbers of nodes in the cluster and the rack. For a fair comparison, the total job size is proportional to the number of nodes within the cluster. As shown, the proposed *SM-HMS* scales well in most cluster configurations, evidenced by the performance improvements with a reasonable number of nodes in each rack. For rack sizes varying from 8 to 64 nodes, the performance of *SM-HMS* keeps increasing with the number of nodes within the cluster, which is due to more nodes can be benefited from the shared *MAB*, enabling high-performance hybrid memory system. Also, the scalability of *SM-HMS* comes from the collaboration among multiple local *Similarity Monitor*s, dramatically reducing the intra-rack network traffic. In the extreme case that all nodes are located within a rack, e.g., 256 nodes, the performance delivered by the *SM-HMS* begins to drop due to either increased network traffic or heavier similarity computation overhead.

#### 4.6.2 Memory Footprint

In the design of *Cameo*, *Chameleon*, and *SM-HMS*, each stacked DRAM page is shared by multiple off-chip DRAM pages. A concern is that multiple pages may compete for a stacked DRAM page, leading to unnecessary performance overhead. To investigate the effectiveness of *SM-HMS* under various sizes of memory footprint, the input parameters of the selected benchmarks are adjusted to generate different memory footprints. Fig. 18b compares the stacked DRAM hit ratio with various memory footprints. *SM-HMS* maintains a relatively high stacked DRAM hit ratio (97.2% on average) with various memory footprints, indicating that *SM-HMS* can benefit a broader range of applications with different memory usage. Two main reasons take the credit. First, "hot" cache lines from multiple pages are co-existed within each stacked DRAM page and unnecessary cache replacement is significantly reduced, since *SM-HMS* performs cache replacement at cache line granularity. Second, the *sliding window section* serves memory requests with a single cache line, enabling more spaces of the stacked DRAM page are used to serve the hard-to-predicted outlier




(a) Various cluster size configurations. (b) Various memory footprints. (c) Various stacked DRAM capacities.

Fig. 18. Sensitivity study on the cluster size, memory footprint and stacked DRAM capacity. Three conclusions can be drawn. First, the local *Similarity Monitor* alleviates performance degradation caused by *MAB* sharing with reasonable rack size. Second, *SM-HMS* maintains the performance improvement with various sizes of memory footprint. Third, with the shared *MAB* and *sliding window section*, *SM-HMS* can benefit the system performance with various sizes of the stacked DRAM.

cache lines. On the contrary, *Cameo* and *Chameleon* suffer from dramatically decreased stacked DRAM hit ratio, on average from 81.4% and 86.3% (for low memory usage) to 17.9% and 22% (for high memory usage), respectively. The stacked DRAM hit ratio decreases slowly for *Hybrid2*, due to the dual-layer design keeping the hot data in the stacked DRAM longer and results in more stacked DRAM hits.

#### 4.6.3 Stacked DRAM Capacity

Fig. 18c compares the normalized average IPC across all work nodes with various stacked DRAM capacities. With decreasing stacked DRAM capacity, fewer memory requests can be served via the stacked DRAM, which leads to degraded performance improvements. Among the four hybrid memory system designs, the performance gain of *SM-HMS* slightly decreases from 2.45× to 2.36 ×, compared to the state-of-the-art approaches dramatically drops from 1.52 ×, 1.65 ×, 2.27× to 1.10 ×, 1.14 ×, 1.67 ×, respectively. The reason behind is that more off-chip pages are competing with the stacked DRAM in the state-of-the-art designs, which aggregates the degraded performance benefit enabled by the stacked DRAM. On the other hand, *SM-HMS* utilizes the *MAB* of each memory page and maximizes the stacked DRAM hit ratio by utilizing both *sliding window section* and *outlier section*, which is proven to be more adaptive with various stacked DRAM capacities than existing approaches.

### 4.7 Effectiveness on Regular Applications

Since the *SM-HMS* heavily relies on memory access similarity for sharing *MAB*s across nodes, regular applications also benefit from the proposed *SM-HMS* due to the high memory access similarity across nodes. Fig. 19 compares the finish time of the fastest node (the lower edge of each bar) and the slowest node (the upper edge of each bar) for four regular applications. As shown, *SM-HMS* delivers a finish time reduction of up to 54.7% (with 30.7% on average) compared with the baseline cluster. At the same time, the three state-of-the-art approaches achieve 5.9%, 11.8%, and 15.9% on average, respectively. Compared with irregular applications, the finish time difference among nodes is significantly smaller, indicating that each node runs similar tasks and can be well balanced among nodes by the job scheduler. Additionally, pages sharing the same MAB benefit from the reduced metadata overhead and the effectiveness of both *sliding window section* and *outlier section*.

## 5 Related Work

*Optimization for Irregular Applications.* Existing researches target at the irregular applications by either exposing the irregularity to the job scheduler for adaptively scheduling [31], [32] or utilizing the irregularity for better resource allocation in the heterogeneous clusters [33], [34]. Nozal et al. propose to balance the loads assigned to different hardware when executing irregular applications with the OneAPI framework [31]. Also, Dai et al. detect the unevenly distributed irregular workload during run-time, and migrate workloads of the overwhelmed nodes to nodes with less workload assigned [32]. On the other hand, the irregularity is utilized to refine resource allocation in the cluster. Yang et al. observe the random and irregular network traffic caused by the irregular applications running in the cluster, and propose to utilize data compression techniques to minimize the network bandwidth consumption [34]. Also, Shin et al. analyze the performance degradation caused by the irregular applications running on the GPU, and propose to accelerate the address translation processes to minimize the data preparation time for the SIMD instructions [33].

*Similarity in the System.* The similarity exists in many aspects of the computing systems, and prior works explore the similarity to improve the system performance. Koller et al. propose an I/O deduplication method based on the observed highly-similar I/O content for both stored and accessed data [35]. In the meanwhile, Xiao et al. utilize the abundant self-similar in the high-level programs to accelerate the data communications among chips [36]. By taking advantage of the parallel working paradigm of the clusters, Liu et al. propose to improve the performance of straggling

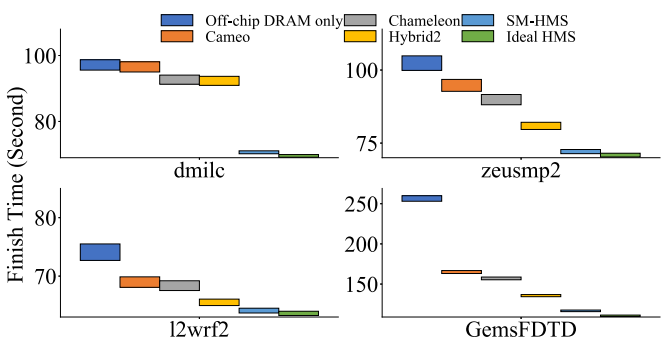



Fig. 19. Comparison on job finish time for the regular applications. *SM-HMS* outperforms the three state-of-the-art approaches by up to 29.5%, 25.9%, and 24.3%, with 22.5%, 19.7%, and 11.3% on average, respectively.

nodes in the cluster with the observed high similarity among nodes running the same job [3].

*Hybrid Memory Systems.* Existing works identify the hot data by either monitoring the memory access behavior of each data block or relying on the information discovered by the operating system. By monitoring the memory access behavior of each data block, the obtained access behavior can be used to identify the following accessed cache line, and accurate data migration/caching can be performed [7], [8], [9]. On the other hand, Prodromou et al. propose to leverage the memory access behavior observed by the operating systems and use Majority Element Algorithm to predict the hot pages[11]. Moreover, researchers integrate computation capability to the stacked DRAM which furthermore reduces the memory traffic and enables higher efficiency for memory-intensive applications [37], [38].

Fundamentally, *SM-HMS* distinguishes itself from existing works by sharing memory access behavior among work nodes according to the quantified memory access similarity and using the shared memory access behavior as the "recipe" to perform accurate and informed cache replacement, which improve the stacked DRAM utilization and reduce the metadata overhead accordingly.

## 6 Conclusion

In this paper, We propose a *Similarity-Managed Hybrid Memory System* (*SM-HMS*) method to address the under-utilized hybrid memory systems by leveraging the various degree of memory access similarity observed in the irregular applications running on the cluster. Inside *SM-HMS*, two techniques, *Memory Access Similarity Measuring* (*MASM*) and *Similarity-based Memory Access Behavior Sharing* (*SMABS*), are included to obtain and utilize the memory access similarity among nodes, where *MASM* quantifies the memory access similarity by measuring the distance among *MAB* vectors, while the *SMABS* shares *MAB*s across nodes according to the memory access similarity and utilizes the shared *MAB*s as "recipes" to perform accurate and informed stacked DRAM cache replacement. Our evaluation results with a set of irregular applications on various cluster configurations consisting of up to 256 nodes have shown that *SM-HMS* outperforms the state-of-the-art approaches, *Cameo*, *Chameleon*, and *Hyrbid2*, on finish time reduction by up to 58.6%, 56.7%, and 31.3%, with 46.1%, 41.6%, and 19.3% on average, respectively. Comparing with the ideal hybrid memory system, *SM-HMS* achieves up to 98.6% (91.9% on average) efficiency. Moreover, experiments on regular applications show that the proposed *SM-HMS* is beneficial to a broader range of HPC applications.

## Acknowledgments

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

## References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] W. Zhang et al., "AMReX: A framework for block-structured adaptive mesh refinement," *J. Open Source Softw.*, vol. 4, no. 37, pp. 1370–1370, 2019.

[3] W. Liu, P. Huang, and X. He, "StragglerHelper: Alleviating straggling in computing clusters via sharing memory access patterns," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2020, pp. 602–611.

[4] M. Diener, S. White, L. V. Kale, M. Campbell, D. J. Bodony, and J. B. Freund, "Improving the memory access locality of hybrid MPI applications," in *Proc. 24th Eur. MPI Users' Group Meeting*, 2017, pp. 1–10.

[5] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Architecture News*, vol. 23, no. 1, pp. 20–24, 1995.

[6] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *SuperComput. Front. Innovations: An Int. J.*, vol. 1, no. 3, pp. 19–55, 2014.

[7] 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 47th Annu. Int. Symp. Microarchitecture*, 2014, pp. 1–12.

[8] J. B. Kotra, H. Zhang, A. R. Alameldeen, C. Wilkerson, and M. T. Kandemir, "Chameleon: A dynamically reconfigurable heterogeneous memory system," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Microarchitecture*, 2018, Art. no. 533.

[9] 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. Architecture*, 2020, pp. 649–662.

[10] 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. Architecture*, 2019, pp. 596–608.

[11] 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. Architecture*, 2017, pp. 433–444.

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

[13] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Microarchitecture*, 2012, Art. no. 235.

[14] 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. Microarchitecture*, 2014, pp. 13–24.

[15] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2019, pp. 399–411.

[16] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2015, pp. 141–152.

[17] V. Young, Z. A. Chishti, and M. K. Qureshi, "TicToc: Enabling bandwidth-efficient DRAM caching for both hits and misses in hybrid memory systems," in *Proc. IEEE 37th Int. Conf. Comput. Des.*, 2019, pp. 341–349.

[18] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *Proc. Int. Symp. Inf. Theory*, 2004, Art. no. 31.

[19] J. Goldberger et al., "An efficient image similarity measure based on approximations of KL-divergence between two gaussian mixtures," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, 2003, pp. 487–493.

[20] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, Mar. 2001.

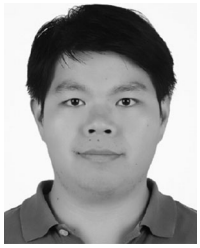
[21] B. Asgari et al., "FAFNIR: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2021, pp. 908–920.

[22] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. 17th Annu. Int. Symp. Comput. Architecture*, 1990, pp. 364–373.

[23] 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. 2011.

[24] P. S. Magnusson, M. Christensson, and J. E. A. Eskilson, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[25] E. Gabriel et al., "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proc. Eur. Parallel Virtual Mach. / Message Passing Interface Users Group Meeting*, 2004, pp. 97–104.
[26] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An integrated cache timing, power, and area model," Compaq Computer Corporation, Palo Alto, California, Tech. Rep. 2001/2, 2001.
[27] M. Adams, "Chombo software package for AMR applications design document," 2014. [Online]. Available: https://crd.lbl.gov/assets/pubs_presos/chomboDesign.pdf
[28] T. El-Ghazawi and F. Cantonnet, "UPC performance and potential: A NPB experimental study," in *Proc. IEEE/ACM Conf. Super-Comput.*, 2002, pp. 17–17.
[29] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group*, vol. 19, pp. 45–74, 2010.
[30] S. S. Vazhkudai et al., "The design, deployment, and evaluation of the coral pre-exascale systems," in *Proc. IEEE/ACM Conf. Super-Comput.*, 2018, pp. 661–672.
[31] R. Nozal and J. L. Bosque, "Exploiting co-execution with OneAPI: Heterogeneity from a modern perspective," in *Proc. Eur. Conf. Parallel Process.*, 2021, pp. 501–516.
[32] G. Dai, P. K. Paluri, A. M. Cheng, and P. Wu, "A virtualization platform designed for irregular multi-process applications," in *Proc. Int. Conf. Parallel Process.*, 2021, pp. 9–12.
[33] S. Shin et al., "Scheduling page table walks for irregular GPU applications," in *Proc. 17th Annu. Int. Symp. Comput. Architecture*, 2018, pp. 180–192.
[34] Y. Yang, J. S. Emer, and D. Sanchez, "SpZip: Architectural support for effective data compression in irregular applications," in *Proc. 17th Annu. Int. Symp. Comput. Architecture*, 2021, pp. 1069–1082.
[35] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," *ACM Trans. Storage*, vol. 6, no. 3, pp. 1–26, 2010.
[36] Y. Xiao, S. Nazarian, and P. Bogdan, "Plasticity-on-chip design: Exploiting self-similarity for data communications," *IEEE Trans. Comput.*, vol. 70, no. 6, pp. 950–962, Jun. 2021.
[37] X. Xie et al., "SpaceA: Sparse matrix vector multiplication on processing-in-memory accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2021, pp. 570–583.
[38] Y. Xiao, S. Nazarian, and P. Bogdan, "Prometheus: Processing-in-memory heterogeneous architecture design from a multi-layer network theoretic strategy," in *Proc. Des. Automat. Test Eur.e Conf. Exhib.*, 2018, pp. 1387–1392.



**Wenjie Liu** (Student Member, IEEE) is currently working toward the PhD degree with the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania. His main reserach interests include DRAM, distributed systems, and nonvolatile memory, etc. He has published papers in various international conferences and journals, including IPDPS, MASCOTS, ICPP, IPCCC, Sigmetrics, the *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, etc.

**Xubin He** (Senior Member, IEEE) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, China, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from the University of Rhode Island, Kingston, RI, in 2002. He is currently a professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. His research interests include computer architecture, data storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the Sigma Xi Research Award (TTU Chapter) in 2005 and 2010. He is a member of USENIX.

**Qing Liu** (Member, IEEE) received the BS and MS degrees from the Nanjing University of Posts and Telecom, China, in 2001 and 2004, respectively, and the PhD degree in computer engineering from the University of New Mexico, in 2008. He is an assistant professor with the Department of Electrical and Computer Engineering, NJIT and Joint Faculty with Oak Ridge National Laboratory. Prior to that, he was a staff scientist with Computer Science and Mathematics Division, Oak Ridge National Laboratory for 7 years.