AceMiner: Accelerating Graph Pattern Matching using PIM with Optimized Cache System

Liang Yan*†, Xiaoyang Lu‡
Xiaoming Chen*†¶, Sheng Xu§, Xingqi Zou*†, Yinhe Han*†¶, Xian-He Sun‡¶
*Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences
†University of Chinese Academy of Sciences, Beijing
‡Department of Compute Science, Illinois Institute of Technology, Chicago, IL
§School of Computer and Information, Anhui Normal University, Wuhu
¶ Corresponding authors: {chenxiaoming, yinhes}@ict.ac.cn, sun@iit.edu

***Abstract*—Graph pattern matching (GPM), a critical algorithm for discovering specific patterns within complex structures, is becoming increasingly important in the data-driven world. GPM applications are memory-bound and can be accelerated by memory-centric computing systems, such as processing-in-memory (PIM). However, there are three primary challenges when it comes to accelerating GPM applications with PIM: (1) difficulty in utilizing locality, (2) heavy data movement, and (3) heavy comparison overhead due to pruning. To address these challenges, we propose AceMiner, a framework to accelerate GPM applications with a software and hardware co-design perspective using PIM. In AceMiner, we embed *hybridCache*, a novel in-DRAM cache system with lower access latency and optimized replacement policy, to leverage the potential locality and reduce data movement in PIM. Additionally, we introduce a comparison unit to address the huge pruning overhead. Experimental results show that AceMiner outperforms the state-of-the-art, achieving speedups of 40.2% and 13.3% over NDMiner and DIMMining respectively, with less energy consumption and design overhead.**

***Index Terms*—Graph pattern matching, Processing-in-memory, Cache system**

I. Introduction

Graph Pattern Matching (GPM) algorithm involves identifying subgraphs (a.k.a embeddings) within a larger graph that conform to specific patterns (as shown in Fig. 1(a)). GPM algorithms are widely used in various domains, including bioinformatics [1], cheminformatics [2], web spam detection [3], and social sciences [4]. However, the GPM algorithms are always memory-bound [5]–[7]. The intricate nature of GPM algorithms, characterized by increased algorithmic complexity and irregular memory accesses, renders it more challenging than standard graph processing [8]–[11]. Consequently, optimizing GPM applications on traditional processor-centric systems like CPUs and GPUs poses significant challenges [7].

This work was supported by National Natural Science Foundation of China (Nos. 62122076, 62025404, 62104230 and 62102005), by Strategic Priority Research Program of CAS (No. XDB44000000), by Key Research Program of Frontier Sciences, CAS (No. ZDBS-LY-JSC012), by Youth Innovation Promotion Association CAS and by University Synergy Innovation Program of Anhui Province (No. GXXT-2021-011).

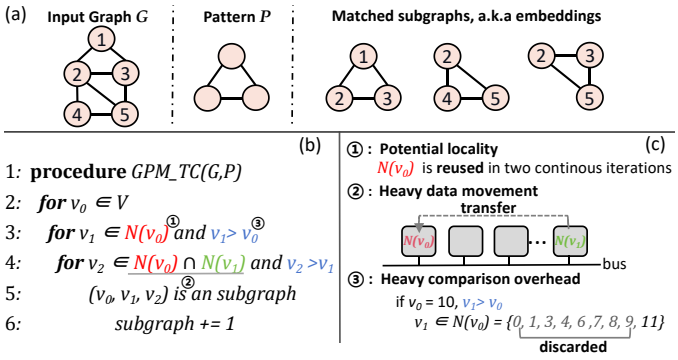

Fig. 1. Overview of GPM algorithm. (a) An example of triangle counting(TC), 3 triangles are found in input graph G. (b) Pseudo code for TC. (c) Challenges in GPM applications.

Processing-in-Memory (PIM) architectures represent a shift from traditional processor-centric systems to a memory-centric computation paradigm. PIM aims to reduce data movement costs by embedding general-purpose or specialized processing units within or near memory modules. This approach can significantly lower data access latency and increase throughput, prompting considerable research into PIM-based hardware and software co-designs, particularly for GPM applications [5]–[7]. However, the elevated computational complexity and intense memory access demands of GPM applications pose unique challenges in effectively leveraging PIM architectures. These challenges include difficulty in utilizing data locality, substantial data movement, and significant comparison overhead for avoiding duplicate subgraph findings.

Difficulty in utilizing locality. Accessing vertices and edges in GPM algorithms often results in non-sequential patterns [6]–[8]. This challenge is further exacerbated by the reliance of GPM algorithms on a set-centric programming model [12], which leads to poor data locality. While strategies such as vertex reordering or graph partitioning have been explored to enhance data regularity and improve locality, these approaches can incur overheads significantly, sometimes higher than the GPM process itself [6], [7]. Despite these challenges, there remains untapped potential for improving locality in GPM applications, particularly in current PIM-accelerated

architectures. For instance, the set-centric model typically involves repeated access to specific data sets at various loop levels. As illustrated in Fig. 1(b), the neighbor set $N(v_0)$ of vertex v_0 is repeatedly utilized in consecutive iterations. This observation implies that caching $N(v_0)$ could be a beneficial approach to leverage this locality effectively. More complex patterns, especially those involving deeper iterations, would likely result in greater reuse of neighbor sets [13]. However, the limited cache capacity in existing PIM architectures, a limitation arising from the area and heat dissipation concerns, hinders the full exploitation of this potential locality [14].

Heavy data movement. While PIM architectures are designed to minimize data transfers between DRAM and CPUs, managing substantial data movement within the PIM architecture remains a significant challenge for GPM applications. As datasets rapidly increase in size and scope, more than a single PIM module is needed to meet increasing memory storage demands, necessitating the deployment of multiple PIM modules. Multiple PIM modules, interconnected in dual in-line memory module (DIMM)-like or network-on-chip (NoC)-like configurations, experience considerable internal data movement [6], [15], as depicted in Fig. 1(c). Current GPM accelerators, such as DIMMining [7] and NDMiner [6], primarily focus on reducing data transfers between DRAM and CPUs, yet they do not fully address the optimization of data movement between PIM modules [5]–[7]. Given the irregular access patterns typical in GPM algorithms, data movement within the PIM architecture is substantial and represents a significant opportunity for enhancing efficiency.

Heavy comparison overhead. Advanced GPM algorithms frequently employ symmetry breaking (a.k.a pruning) to avoid detecting the same subgraph across multiple iterations (avoiding duplicate subgraph findings). This process involves comparing vertices to filter out those that do not meet the pattern search sequence constraints, as demonstrated in Fig. 1(b) line 3, 4. However, recent studies indicate that most vertices retrieved in one iteration are often discarded in the subsequent one (Fig. 1(c)) [6], [7]. Symmetry breaking reduces calculations but increases comparison overhead, leading to inefficient use of CPU cache and DRAM bandwidth due to unnecessary vertices loading from memory to processing unit. This trade-off between reducing computation and loading unnecessary memory presents challenges for frameworks that aim to optimize performance and energy efficiency. DIMMining [7] has implemented an index pre-comparison method to decrease the redundant vertices loading overhead, but it introduces additional storage demands and programming burdens.

In response to the identified challenges, we introduce AceMiner, a groundbreaking software and hardware co-designed framework tailored for GPM applications in multiple PIM systems. AceMiner stands out with its distinctive features. First, AceMiner presents an advanced programming interface developed to enhance efficient set-centric operations in GPM applications for PIM systems. Second, AceMiner is distinguished by its integration of *hybridCache*, an in-DRAM cache solution specifically designed as a core component for

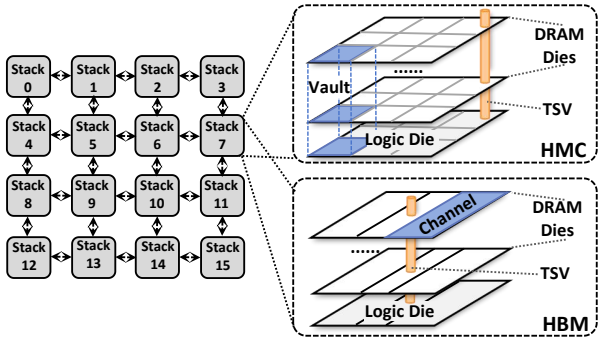

Fig. 2. Multiple PIM architecture based on 3d-stacked memory technologies.

PIM systems. To the best of our knowledge, this represents the first implementation of an in-DRAM cache tailored to boost the performance of GPM applications within a PIM framework. Our *hybridCache* provides rapid data access and utilizes data locality, which are especially beneficial in GPM applications characterized by frequent neighbor set reuse, thereby significantly reducing data movement within PIMs. Moreover, *hybridCache* incorporates an advanced replacement policy, strategically developed to manage the irregular memory access patterns typical in GPM applications, further optimizing memory efficiency and enhancing system performance. Third, AceMiner seamlessly integrates comparison units with *hybridCache*. This integration enables AceMiner to conduct highly efficient comparison processes in symmetry breaking for frequently accessed data sets, leading to considerable bandwidth savings and minimal area overhead. Comprehensive experiments on real-world graphs show that AceMiner achieves a 40.2% and 13.3% speedup over state-of-the-art frameworks NDMiner [6] and DIMMining [7], respectively.

II. Background and Motivation

A. Processing in Memory

PIM effectively addresses the memory-bound problem by relocating computation to the location of data, which leverages high internal memory bandwidth through 3D stack technology and minimizes the need for costly off-chip data transfers [16]. With the help of PIM, significant performance improvements have been observed in various applications, including data analytics, deep neural network (DNN) training, graph processing, and GPM applications [5]–[7]. Key characteristics of PIM architectures include multiple processing units within the memory chip, offering higher bandwidth and lower latency than the host processor [14]. The processing units, ranging from general-purpose cores to specialized processors, typically operate at a few hundred megahertz and are equipped with a limited number of registers and modest-sized cache or scratchpad memory [14] for local data storage and movement.

In this work, we concentrate on PIM systems constructed using 3D-stacked memories like Hybrid Memory Cube (HMC) and High-Bandwidth Memory (HBM), as they offer a balance between technological maturity and performance benefits [15]. As shown in Fig. 2, such an PIM system typically consists of multiple interconnected memory stacks, forming a memory network. Each stack comprises DRAM dies stacked vertically

atop a logic die. Through-silicon vias (TSVs) allow low-latency, high-bandwidth inter-die connectivity, e.g. hundreds of GB/s to several TB/s. HMC and HBM employ divergent organizational paradigms within stacks. HBM partitions individual DRAM dies, treating different chunks across dies as independent channels. Comparatively, HMC also divides dies but consolidates matching portions from all into a vault akin to conventional channel access. The PIM system associates computing logic with each HMC vault or HBM channel, directly placing logic on the bottom die in true 3D or via 2.5D interposer-based integration. In this work, we use general-purpose and energy-efficient cores as the PIM processing unit, although our architecture can accommodate other types of logic, such as reconfigurable logic [17], [18] and application-specific integrated circuits (ASICs) [19], [20].

B. Graph Pattern Matching

GPM algorithm aims at finding all unique subgraphs within an input graph G that are isomorphic to a specified pattern P. Isomorphism here implies a one-to-one correspondence between vertices and edges of P and the subgraph. GPM algorithm employs a search tree to enumerate subgraphs in G that match a defined pattern P. This process begins from a single vertex, expanding one vertex or edge at a time through multiple layers of nested loops, each corresponding to a vertex in P. For instance, a triangle pattern requires three such loops. Advanced GPM algorithms follow a set-centric programming paradigm, utilizing set operations like intersection and difference to manage neighbor relationships. In order to prevent duplicate subgraph findings, symmetry breaking constraints are applied based on P. For example, as shown in Fig. 1(b), lines 3 to 4, $v_1 > v_0$ and $v_2 > v_1$, are constraints of symmetry breaking to avoid counting duplicate subgraphs. Typically, GPM algorithms are classified into two primary categories: pattern-oblivious and pattern-aware. Pattern-aware GPM algorithms are generally more efficient than their pattern-oblivious counterparts, primarily due to their enhanced ability to eliminate redundant computations [21]. In this work, we focus on utilizing pattern-aware algorithms.

GPM applications typically utilize a task-based programming and execution model, which is also prevalent in many data-centric workloads to enhance parallelism [6], [9], [13]. A task-based model divides a computational task into independent subtasks, which are executed simultaneously on multiple processing units. These subtasks have clear inputs and outputs and do not rely on each other, allowing for parallel processing. The task-based model offers a flexible approach to scheduling computation loads within an application. It is particularly effective for PIM, which often encompasses hundreds of processing units. Additionally, it provides the foundation for efficiently managing remote neighbor set accesses, a crucial aspect in GPM applications. Furthermore, task abstraction facilitates the integration of data access information, such as the addresses required by a task. In this work, we adopt the task-based model, capitalizing on its inherent advantages to achieve optimal performance in GPM applications.

C. Motivation

The primary challenges in GPM applications include poor locality, heavy data movement, and significant comparison overhead associated with symmetry breaking. We identify that the set-centric programming paradigm, commonly used in GPM algorithms, results in extensive reuse of neighbor sets across different loop levels, especially in complex pattern searches. In conventional processor-centric architectures, caches are commonly utilized to enhance program locality [22], [23]. However, in current PIM architectures, cache system of PIM side which is a high-speed memory system for temporary data storage, faces capacity limitations due to area and heat dissipation constraints. Therefore, they cannot fully exploit the potential locality in GPM applications. We draw inspiration from in-DRAM cache technology to design a novel cache system and replacement policy, specifically tailored for PIM architectures. The innovative cache system is crafted to meet the locality demands of GPM applications. It also addresses the challenge of heavy data movement in GPM applications by caching remotely accessed data locally, thereby significantly enhancing overall data processing efficiency. Furthermore, as symmetry breaking in GPM applications often discards a substantial portion of vertices after initial loading, we integrate comparison units with *hybridCache* to reduce the considerable overhead of useless memory loads.

III. System Design

In this section, we first outline the hardware architecture of AceMiner. Subsequently, we delve into the details of our proposed hardware-software co-optimizations within AceMiner, aimed at addressing the key challenges of utilizing locality, reducing heavy data movement, and minimizing comparison overhead of symmetry breaking in GPM applications.

A. Hardware Architecture

Fig. 3 shows the overall architecture of AceMiner, which can be divided into the host and PIM parts. On the host side, as depicted in Fig. 3(a), we have configured 4 out-of-order (OoO) CPU cores and equipped them with three cache levels, in which L1 and L2 are private for each core, and LLC is shared. L1 is divided into the I-cache and D-cache. The PIM controller coordinates data transfers between the host and PIM side, handling the distribution and reception of data for high performance, reliable and efficient memory access. On the PIM side, as depicted in Fig. 3(b), to accommodate the growing size of graph datasets used in GPM applications and the large storage demand of intermediate results generated during the GPM process, AceMiner utilizes a mesh topology to connect multiple PIMs, which is set by default to a 4×4 scale. Each PIM stack comprises 32 processing units and 256 banks, interconnected through a crossbar-based network-on-chip (NoC). We provision a memory capacity of 4GB for each PIM stack, amounting to 64GB across the entire system. Notably, the hardware architecture of AceMiner is not dependent on any specific memory or interconnection technologies. Our design can use HMC [24], HBM [16], and

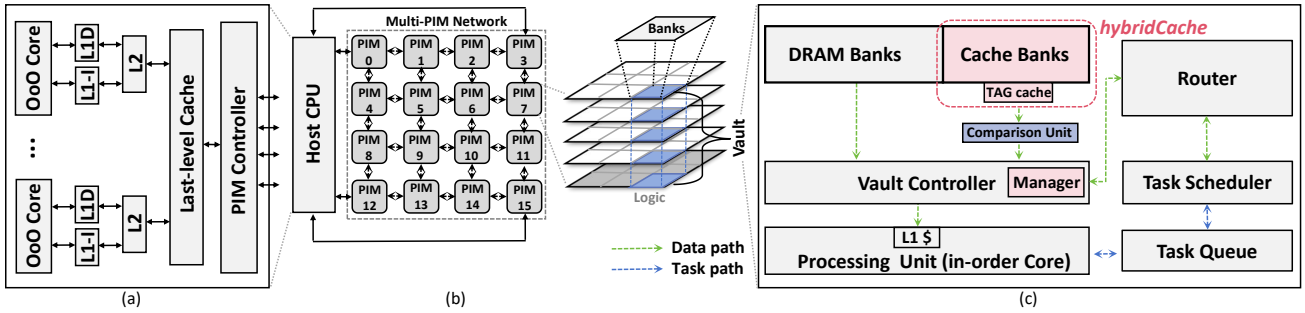

Fig. 3. Overall AceMiner system architecture. (a) Host side. (b) PIM side. (c) Each vault in the PIM.

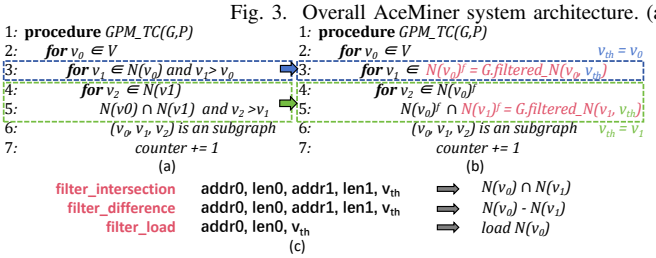

Fig. 4. Programming interface. (a) Original TC Code. (b) AceMiner TC Code. (c) Host ISA instructions to support PIM.

other 3D memories as long as they exhibit the similar structure, although HMC is used as a demonstration in the diagram.

For each vault in the PIM, as illustrated in Fig. 3(c), the processing unit is a simple in-order core, equipped with private L1 caches for instructions and data. Every core employs a local TLB to translate virtual addresses into physical addresses. In support of the task-based execution approach detailed in Sec. II-B, AceMiner integrates both a task queue and a task scheduler in each vault. The task queue maintains all enqueued tasks, and when a core completes a task, the task queue dequeues a new task for the core to execute. In addition to the task queue, AceMiner employs a task scheduler to manage tasks according to a predefined scheduling policy, which forwards newly generated tasks to suitable PIM units. Moreover, each core can access data related to tasks from local and remote memories, managed by the vault controller and router to ensure effective data retrieval and processing.

We propose *hybridCache*, a novel PIM side cache system, as depicted in Fig. 3(c) (red dashed box), which leverages in-DRAM cache technology [25]. To optimize cache management, we incorporate a dedicated manager component within the vault controller for cache replacement. We utilize a portion of the DRAM in each PIM unit as an in-DRAM cache, which offers several advantages. First, the large size of the in-DRAM cache enables us to fully exploit the locality potential from the frequently reused neighbor sets. Second, this approach significantly reduces the cost associated with accessing remote data. It can cache much remote data locally while improving the flexibility in task scheduling. The choice of in-DRAM cache for AceMiner is driven by the strict area constraints inherent in 3D-stacked memory. Details about *hybridCache* and its implementation are discussed in Sec. III-C.

B. Programming Interface

In the AceMiner system, the costly set-related operations are exclusively offloaded to PIM for acceleration, following practices established in prior works [5], [6]. In order to facilitate effective communication of set-related operations to memory, AceMiner introduces a specialized compiler that transforms original graph mining code into PIM-friendly code via the host CPU. As depicted in Fig. 4(a) and Fig. 4(b), the compiler analyzes the source code to pinpoint instructions that are suitable for PIM acceleration. These targeted instructions include computations for set operations, neighbor set loads, and symmetry-breaking constraints. The compiler then encapsulates these identified instructions with PIM-specific instructions to optimize them for efficient processing in AceMiner.

The PIM instructions, as illustrated in Fig. 4(c), are designed to complete set-centric computation operations and aid in the comparison of symmetry breaking. The threshold vertex (i.e., v_{th}), crucial for the comparison units (details to be provided in Sec. III-D), is determined at runtime by the host CPU and then communicated to the PIM units. In line with recent academic and industrial PIM implementations, we consider that data allocated for PIM is stored in physically contiguous memory blocks [26]. This approach of contiguous memory mapping ensures that PIM instructions need to translate only a base address ($addr$), with the rest of the addresses within a defined range (len) being straightforward to deduce.

C. hybridCache

In AceMiner, we introduce a novel PIM cache system named *hybridCache*. The implementation of *hybridCache* enables AceMiner to effectively tackle the challenges of utilizing locality and managing data movement when using PIM to accelerate GPM applications. Although there have been numerous studies on DRAM caches within conventional architecture, where faster on-die or in-package DRAM is used as caches for traditionally slower DDR and nonvolatile memories [25], [27], applying DRAM-based caching in PIM systems presents distinct challenges. The primary challenges in integrating an in-DRAM cache into a PIM system include reducing the latency of access to the in-DRAM cache and designing an appropriate replacement policy to enhance overall system efficiency. These challenges necessitate a unique approach tailored to the specific demands and architecture of PIM systems, differing significantly from conventional DRAM caching strategies.

1) Access Latency Optimization: Generally, the access latency for any cache, whether conventional or in-DRAM cache, primarily depends on tag searching [25]. In the context of in-DRAM caches, tag placement significantly differs from that in conventional CPU-side caches due to the larger size of

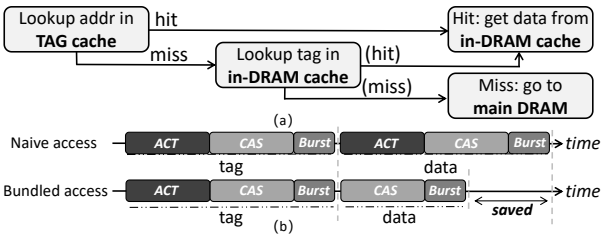

Fig. 5. Design of *hybridCache*. (a) Tag searching procedure. (b) Bundled and naive accesses of in-DRAM cache.

in-DRAM cache tags. Storing all in-DRAM cache tags in SRAM would greatly reduce latency but would also introduce substantial area overhead, making it impractical for PIM systems. Conversely, storing all tags in DRAM accommodates larger tag sizes but can considerably slow down access speeds. In AceMiner, we aim to minimize the time spent on tag searching while avoiding excessive area overhead. To achieve this, *hybridCache* employs a lightweight SRAM-based TAG cache to accelerate data access. All in-DRAM cache tags are stored in DRAM to meet size requirements, complemented by a small SRAM-based TAG cache that caches frequently accessed tags of the in-DRAM cache. This dual-layer design efficiently reduces the tag searching time for the in-DRAM cache. The procedure of tag searching within *hybridCache* is illustrated in Fig. 5(a). The TAG cache acts as a quick "lookup table" to determine if a requested data block already resides in the in-DRAM cache. On a TAG cache hit, the data can be fetched directly, drastically reducing the overall time needed to access data from the in-DRAM cache.

Nevertheless, in the event of a TAG cache miss, it becomes necessary to access the in-DRAM cache to verify the tag and data area, resulting in the costly activation of DRAM twice. To minimize the total access time to the in-DRAM cache when TAG cache misses occur, we propose the implementation of a bundled access scheme. This approach combines the in-DRAM tag access and data access within a single row activation, thereby reducing the overall time required for accessing the in-DRAM cache. This configuration enables a single, compound access operation to retrieve both tag and data, based on the premise that both tag and data are stored together in each physical DRAM row. Our *hybridCache* implements a straightforward modification to the scheduling algorithm of the memory controller, which now treats separate tag and data lookups as a unified bundled access. In practice, during an in-DRAM cache lookup, the memory controller initially issues standard activation and read commands. These commands load the requested in-DRAM cache line into the DRAM row buffer and facilitate the reading of the tag information. The pivotal advantage of this bundled access approach is that it obviates the need to reopen a row for subsequent data access, as illustrated in Fig. 5(b). This innovation leads to a reduction in the time required to activate a row. By enabling compound access, *hybridCache* significantly reduces latency and enhances overall performance.

2) Advanced Replacement Policy: The memory access patterns in GPM applications significantly differ from those in conventional applications, primarily due to the prevalence of cyclic access patterns. Such distinctive behavior reduces the effectiveness of traditional cache replacement policies like Least Recently Used (LRU) or Least Frequently Used (LFU). For instance, with a loop-like pattern, especially when it exceeds cache capacity, LRU often mistakenly evicts blocks that will soon be needed due to their recent inactivity [28]. Similarly, LFU faces challenges as it only tracks metadata over a brief access period, thus potentially overlooking important long-term patterns [29]. Neither LRU nor LFU alone can handle the unique nature of memory accesses and efficiently exploit locality in GPM applications. Consequently, there is an urgent requirement for a more comprehensive cache replacement policy in GPM applications.

In AceMiner, we manage the in-DRAM cache considering the decisions from both LRU and LFU policies. As illustrated in Fig. 6, we deploy detectors for LRU and LFU to identify two potential candidate cache lines for eviction. Subsequently, we evaluate the reuse distance [28] of the candidates to decide which cache line should be evicted. The reuse distance is a metric representing the number of distinct memory addresses accessed between two consecutive requests for the same address. Given that GPM algorithms typically exhibit regular cyclic access patterns, this reuse distance becomes a reliable indicator for predicting the distance between current and subsequent requests. By leveraging reuse distance in our cache replacement policy, we ensure that the data most likely to be accessed soon remains in the cache, enhancing the effectiveness of traditional LRU and LFU policies. Our cache replacement policy aligns with common practices in modern caching systems and does not impose additional overhead.

3) Other Considerations: The specific characteristic of GPM applications, which typically requires only read accesses [5]–[7], allows *hybridCache* to operate efficiently without necessitating performance trade-offs for maintaining coherence requirement. For other applications, like graph computing with read-write data which requires some sort of coherence between the PIM cores, our *hybridCache* only stores the read-only primary data, data writes bypass the *hybridCache* and directly go to the home memory locations, therefore greatly simplifying the coherence requirement. This simplified coherence makes *hybridCache* appropriate for more diverse applications. Additionally, it is important to note that *hybridCache* is designed to be fully compatible with existing DRAM systems. By utilizing block-based access with a size of 64 bytes, *hybridCache* is tailored to prevent fragmentation and minimize bandwidth wastage on transfers, aligning with conventional architectures.

D. Comparison Units

To minimize bandwidth wastage from loading vertices that will ultimately be discarded, AceMiner incorporates a comparison unit with the column decoder of the in-DRAM cache bank. Because this unit directly uses cache line values read from bank, it is placed at the column decoder output. As shown in Fig. 7, this unit is equipped with a comparator suite designed to compare vertex ID data with a threshold vertex ID

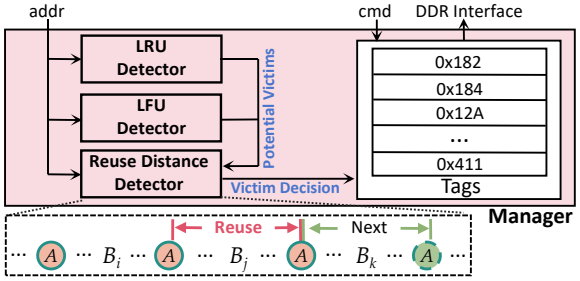

Fig. 6. The *hybridCache* manager in a vault controller for replacement policy.

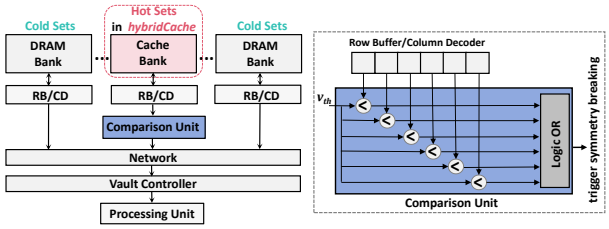

Fig. 7. Design of the comparison unit in AceMiner.

obtained from the memory request. When the data satisfies the symmetry breaking criteria, the comparison unit emits a signal to the vault controller, halting any further unnecessary vertex loads. Specifically, our unit is optimized to execute symmetry breaking primarily for hot neighbor sets identified and selected by *hybridCache*.

In AceMiner, integrating the comparison unit with the in-DRAM cache banks, rather than across all DRAM banks, is based on several considerations. While equipping each bank with comparison units could theoretically enhance bandwidth, such an expansion is not always practical or necessary. For instance, in 3-MC mining problems, the top 10% of frequently accessed neighbor sets account for nearly 44% of total accesses [7]. Therefore, the frequently accessed sets benefit most from the comparison unit. This consideration aligns with the function of *hybridCache*, which is designed to store such hot sets. Furthermore, another consideration in our design is the area cost associated with each comparison unit, which includes multiple comparator circuits. By selectively integrating these units with the in-DRAM cache banks, we achieve an efficient balance between lightweight and system effectiveness.

IV. Methodology

A. System Configuration

We utilize the cycle-accurate Ramulator simulator [30] to evaluate the performance of AceMiner. Ramulator, a DRAM simulator, has been cross-validated with real DRAM devices and is widely used in prior research for PIM evaluations [6], [26]. Table I presents the detailed system configurations. Additionally, we use CACTI 6.5 [31] to model the area and power consumption.

B. Benchmarks and Workloads

To evaluate AceMiner, we use four mining patterns: 3 Clique Finding (3CF), 4 Clique Finding (4CF), 5 Clique Finding (5CF), and 3 Motif Counting (3MC). Additionally, we utilize six real-world graph datasets, as detailed in Table II. Both the mining patterns and graph datasets selected are consistent with previous works [6], [7], [12].

TABLE I
SIMULATED SYSTEM CONFIGURATIONS.

	Parameter	Value
Host Side	CPU Cores	4 Cores, OoO, 2GHz, 192-entry ROB
	Private L1 Cache	Separated 32KB I/D-Cache per core, 8-ways, 2-cycle hit latency, 8-entry MSHR, 64B line size
	Private L2 Cache	256KB/core, 8-ways, 12-cycle hit latency, 12-entry MSHR, 64B line size
	Shared L3 Cache	shared 2MB/core, 32-ways, 35-cycle hit latency, 32-entry MSHR, 64B line size
PIM side	PIM System	4×4 HMC v2.1 [24] stacks in mesh, 32 vaults per stack 4GB and 256 banks per stack, 64GB in total
	PIM Processing Unit	500Mhz, single-issue, in-order
		L1 I/D cache: 32kB, 4-way, 64B line size, LRU
	DRAM Parameters	tCAS=tRCD=tRP=22ns 5.0 pJ/bit RD/WR, 535.8pJ ACT/PRE 4.0 pJ/bit inter-stack net, 0.4pJ/bit Intra-stack net
	hybridCache	in-DRAM cache: 256MB per stack, 16-way, 64B line size, Advanced Replacement Policy
		TAG cache(SRAM): 16kB, 4-way, 64B line size, LRU

TABLE II
GRAPH DATASETS USED FOR EVALUATION.

Graph	#Vtx	#Edge	Size
P2P(PP)	10.9K	40.0K	620K
Astro(AS)	18.8K	198K	5.3M
MiCo(MI)	100K	1.08M	18MB
com-Youtube(YT)	1.13M	2.99M	57MB
cit-Patents(PA)	3.77M	16.52M	332MB
soc-LiveJournal1(LJ)	4.85M	43.11M	1.2G

V. Results

A. Overall Performance

We compare AceMiner with state-of-the-art PIM frameworks for GPM accelerating, NDMiner [6] and DIMMining [7]. The main differences between the three frameworks are outlined in Table III. Notably, both NDMiner and DIMMining utilize specially designed PIM processing units tailored for GPM algorithms, while AceMiner employs traditional general purpose in-order cores—offering greater practical applicability. For comparison optimization, AceMiner's hardware comparison unit design shows more simplicity and efficiency with the help of *hybridCache* than NDMiner, as described in Sec. III-D. DIMMining's software comparison optimization approach places an additional burden on programmers and also introduces additional storage demands.

Fig. 8 shows the performance comparison under these 3 architecture with all results normalized to the execution time of NDMiner. On average, AceMiner achieves a 40.2% speedup over NDMiner and a 13.3% speedup over DIMMining. This performance enhancement is even more pronounced when mining complex patterns in large datasets. The performance improvements in AceMiner are largely attributed to exploiting the locality in set reuse and reducing data movement with the innovative *hybridCache*. In contrast, NDMiner and DIMMining are constrained by smaller caches unable to unlock

TABLE III
GPM ACCELERATING WITH PIM FRAMEWORKS.

	Comparison OPT	PIM logic	PIM cache
NDMiner	hardware	dedicated	small cache
DIMMining	software	dedicated	small cache
AceMiner	hardware	general	*hybridCache*

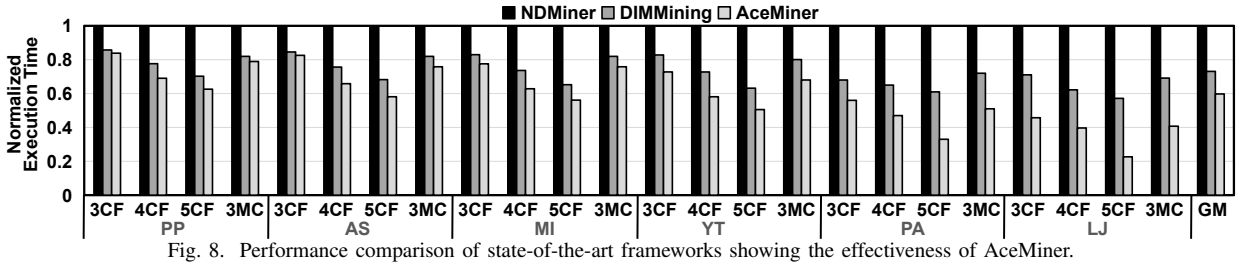

Fig. 8. Performance comparison of state-of-the-art frameworks showing the effectiveness of AceMiner.

such benefits. Additionally, the comparison unit in AceMiner further enhances performance. Unlike NDMiner, which distributes comparison units across all DRAM banks, AceMiner targets placement only within the *hybridCache* for "hot" neighbor sets. This targeted approach yields significant area savings versus NDMiner's distributed approach. And compared to DIMMining's pre-comparison optimization through software, AceMiner introduces minimal storage overhead while avoiding programming intricacies. In summary, AceMiner presents a practical alternative to specialized PIM frameworks through localized caching and judicious comparison unit placement, achieving state-of-the-art GPM acceleration.

B. Performance Analysis

To analyze the performance of AceMiner in detail, we evaluate the impact of integrating *hybridCache* and comparison units in AceMiner. We introduce AceMiner-base, excluding both *hybridCache* and comparison units, while keeping other configurations the same as AceMiner. We also implement AceMiner-*hybridCache*, a simplified version of AceMiner, which integrates only *hybridCache*. The execution time of each configuration is normalized to the AceMiner-base. Fig. 9 shows that, on average, AceMiner-*hybridCache* outperforms AceMiner-base by a factor of 10.5. Furthermore, AceMiner, which combines both *hybridCache* and comparison units, outperforms AceMiner-base by an average factor of 21.6, clearly highlighting the combined efficacy of these components.

The advanced performance of *hybridCache* primarily comes from the enhanced locality and reduced data movement. For locality improvement, the *hybridCache* can leverage the locality of the set reuse in GPM algorithms, especially for the complex pattern with deeper loops and larger dataset sizes. As depicted in Fig. 10(a), we measured the cache miss ratio for four patterns using the largest dataset, LJ. The cache miss ratio in AceMiner-base, observed from the small cache embedded in PIM, is extremely high, averaging 72.6%. In contrast, AceMiner-*hybridCache* significantly reduces the *hybridCache* miss ratio to an average of 15.8%. The improvement comes from the large size of the in-DRAM cache design with our access latency and replacement optimization. Regarding data movement reduction of inter and intra PIM, Fig. 10(b) illustrates the data movement in AceMiner-*hybridCache*, normalized to AceMiner-base, using the LJ dataset. The results indicate that AceMiner-*hybridCache* reduces data movement by 75.7% compared to AceMiner-base. This reduction is achieved through the caching of remote access data locally.

Based on the results of AceMiner in Fig. 9, it is deduced that with the help of a comparison unit, AceMiner can get 1.9× performance improvement over AceMiner-base. This improvement primarily comes from avoiding unnecessary loads through comparison units. The comparison unit shows effectiveness, especially for the complex pattern with more symmetry breaking constraints. This optimization could also significantly enhance performance for graphs with a larger memory footprint.

TABLE IV
DESIGN OVERHEAD COMPARISON.

	NDMiner	DIMMining	AceMiner
Area	0.64 mm^2	0.38 mm^2	0.11 mm^2
Power	51.59 mW	105.82 mW	10.85 mW

C. Energy Analysis

Fig. 11 shows the energy consumption comparison of AceMiner normalized to NDMiner using small, medium and large datasets. Results demonstrate that AceMiner reduces energy consumption by 22.4% on average compared to NDMiner. NDMiner introduces a vertex reordering technique to improve data locality. However, the reordering process itself uses additional energy. Furthermore, the distributed organization of comparison units in NDMiner induces more energy consumption than the approach used in AceMiner. A direct comparison of energy usage between AceMiner and DIMMining is not possible due to a lack of available open-source implementations. Nevertheless, DIMMining incorporates complex PIM logic design which likely increases energy demands. Additionally, DIMMining does not optimize for neighbor set reuse, which AceMiner leverages to be more efficient. We estimate that AceMiner reduces energy consumption to a greater extent than DIMMining as a result.

D. Overhead Analysis

We present the design overhead of AceMiner in Table IV. The area overhead of AceMiner primarily originates from the TAG cache and the comparison units. Each PIM module requires 0.092 mm^2 for the TAG cache and only 0.015 mm^2 for the comparison units. The total area overhead of AceMiner is considerably less than that of NDMiner and DIMMining. AceMiner contributes an additional overhead of 0.11% of a DRAM chip, which typically measures around 100 mm^2. Furthermore, the power overhead of AceMiner is 10.85 mW, which is considerably lower compared with the more complex circuit requirements of NDMiner and DIMMining.

VI. CONCLUSIONS

Irregular memory accesses in GPM applications make them particularly well-suited for acceleration using PIM. However,

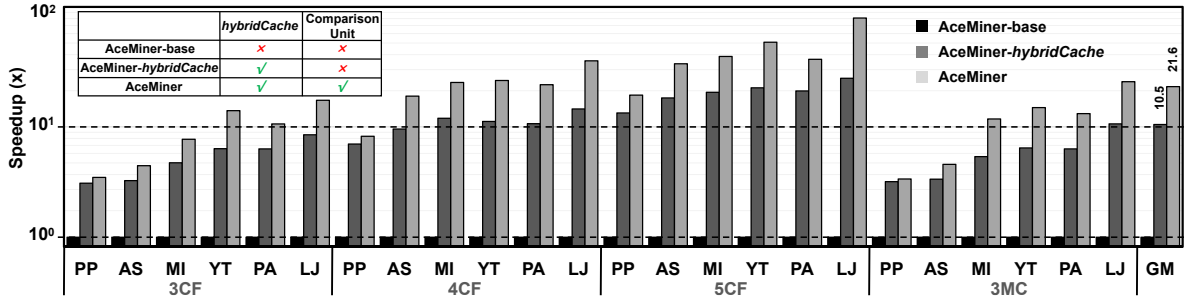

Fig. 9. Performance comparison of AceMiner configurations showing the effectiveness of proposed optimizations.

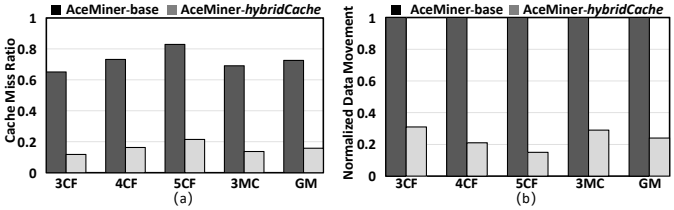

Fig. 10. Analysis of locality improvement and data movement reduction w/o *hybridCache* on the largest dataset LJ. (a) Cache miss ratio (b) Normalized data movement.

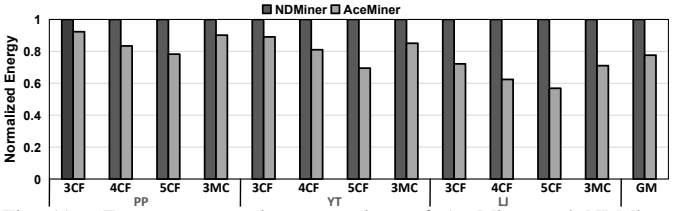

Fig. 11. Energy consumption comparison of AceMiner and NDMiner. Normalized to NDMiner.

the challenge arises when the large graph sizes of GPM applications encounter the limited SRAM cache capacity in current PIM architectures. This mismatch hinders the full utilization of data locality and minimization of data movement. In this work, we introduce AceMiner, a framework for accelerating GPM applications using PIM. Central to AceMiner is *hybridCache*, an in-DRAM cache system optimized for PIM. We have enhanced *hybridCache* by improving access latency and implementing an advanced replacement policy. Furthermore, we have integrated comparison units with *hybridCache* to manage the substantial overhead associated with symmetry breaking effectively. Our extensive evaluations show that AceMiner consistently surpasses state-of-the-art solutions while maintaining low overhead, demonstrating significant advancement in performance.

REFERENCES

[1] N. Alon *et al.*, "Biomolecular network motif counting and discovery by color coding," *Bioinformatics*, vol. 24, no. 13, pp. i241–i249, 2008.

[2] Z. Shao *et al.*, "Mining discriminative patterns from graph data with multiple labels and its application to quantitative structure–activity relationship (qsar) models," *JCIM*, pp. 2519–27, 2015.

[3] Y. I. Leon-Suematsu *et al.*, "Web spam detection by exploring densely connected subgraphs," in *IEEE ICWIIAT*, 2011.

[4] A. Topirceanu *et al.*, "Uncovering the fingerprint of online social networks using a network motif based approach," *Computer Communications*, vol. 73, no. JAN.1, pp. 167–175, 2016.

[5] M. Besta *et al.*, "Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems," in *MICRO*, 2021, pp. 282–297.

[6] N. Talati *et al.*, "Ndminer: accelerating graph pattern mining using near data processing," in *ISCA*, 2022, pp. 146–159.

[7] G. Dai *et al.*, "Dimmining: pruning-efficient and parallel graph mining on near-memory-computing," in *ISCA*, 2022, pp. 130–145.

[8] P. Yao *et al.*, "A locality-aware energy-efficient accelerator for graph mining applications," in *MICRO*, 2020, pp. 895–907.

[9] Y. Wu *et al.*, "Shogun: A task scheduling framework for graph mining accelerators," in *ISCA*, 2023, pp. 1–15.

[10] L. Yan *et al.*, "Copim: a concurrency-aware pim workload offloading architecture for graph applications," in *ISLPED*, 2021, pp. 1–6.

[11] Z. Li *et al.*, "Graphring: an hmc-ring based graph processing framework with optimized data movement," in *DAC*, 2022, pp. 1063–1068.

[12] D. Mawhirter *et al.*, "Automine: harmonizing high-level abstraction and high performance for graph mining," in *ACM SOSP*, 2019, pp. 509–523.

[13] Q. Chen *et al.*, "Fingers: exploiting fine-grained parallelism in graph mining accelerators," in *ASPLOS*, 2022, pp. 43–55.

[14] A. Boroumand *et al.*, "Conda: Efficient cache coherence support for near-data accelerators," in *ISCA*, 2019, pp. 629–642.

[15] B. Tian *et al.*, "Abndp: Co-optimizing data access and load balance in near-data processing," in *ASPLOS*, 2023, pp. 3–17.

[16] D. U. Lee *et al.*, "25.2 a 1.2 v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *ISSCC*, 2014, pp. 432–433.

[17] A. Farmahini-Farahani *et al.*, "Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules," in *HPCA*, 2015, pp. 283–295.

[18] M. Gao *et al.*, "Hrl: Efficient and flexible reconfigurable logic for near-data processing," in *HPCA*, 2016, pp. 126–137.

[19] D. Kim *et al.*, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 380–392, 2016.

[20] X. Xie *et al.*, "Spacea: Sparse matrix vector multiplication on processing-in-memory accelerator," in *HPCA*, 2021, pp. 570–583.

[21] K. Jamshidi *et al.*, "Peregrine: a pattern-aware graph mining system," in *Proceedings of European Conference on Computer Systems*, 2020, pp. 1–16.

[22] X.-H. Sun *et al.*, "The memory-bounded speedup model and its impacts in computing," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 64–79, 2023.

[23] X. Lu *et al.*, "Chrome: Concurrency-aware holistic cache management framework with online reinforcement learning," in *HPCA*, 2024, pp. 1154–1167.

[24] H. M. C. Consortium *et al.*, "HMC Specification 2.1," *Retrieved May*, 2019.

[25] D. Jevdjic *et al.*, "Unison cache: A scalable and effective die-stacked dram cache," in *MICRO*, 2014, pp. 25–37.

[26] L. Ke *et al.*, "Recnmp: Accelerating personalized recommendation with near-memory processing," in *ISCA*, 2020, pp. 790–803.

[27] C. Chou *et al.*, "Candy: Enabling coherent dram caches for multi-node systems," in *MICRO*, 2016, pp. 1–13.

[28] S. Jiang *et al.*, "Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[29] D. Lee *et al.*, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE TC*, vol. 50, no. 12, pp. 1352–1361, 2001.

[30] Y. Kim *et al.*, "Ramulator: A fast and extensible dram simulator," *IEEE CAL*, vol. 15, no. 1, pp. 45–49, 2015.

[31] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.