ProMiner: Enhancing Locality, Parallelism, and Offloading for Graph Mining on Processing-in-Memory Systems

Liang Yan, Xiaoyang Lu, *Member, IEEE*, Sheng Xu, *Associate Member, IEEE*, Xiaoming Chen, *Member, IEEE*, Xingqi Zou, Yinhe Han, *Senior Member, IEEE*, and Xian-He Sun, *Life Fellow, IEEE*

***Abstract*—Graph mining, critical for discovering specific patterns within complex structures, is becoming increasingly important in our data-driven world. Due to their memory-bound nature, graph mining applications encounter significant limitations with conventional processor-centric systems, like central processing units (CPUs) and graphics processing units (GPUs), stemming from the costly data movement between memory and processing units. Memory-centric computing systems, such as processing-in-memory (PIM) where computation occurs directly within or near memory modules, have the potential to accelerate graph mining. However, accelerating graph mining applications with PIM presents three primary challenges: 1) the difficulty in utilizing locality; 2) the challenge of exploring parallelism; and 3) the complexity of workload offloading between PIM and CPU. Addressing these intricate challenges, we introduce ProMiner, a novel framework that integrates three key techniques through cohesive software and hardware co-design. First, we propose a partitioning method tailored for graph mining to enhance data locality. Second, we design a coarse-fine parallelism optimization scheme to explore parallelism across different levels of memory. Third, we introduce a concurrency-aware mechanism for performance estimation, aimed at identifying the optimal computing engine for workload offloading to maximize performance. Our experimental results demonstrate that ProMiner significantly advances the state-of-the-art in graph mining, achieving 48.8% and 29.9% execution time reduction over NDMiner and DIM- Mining, respectively.**

***Index Terms*—Graph mining, parallelism, processing-in-memory (PIM), workload offloading.**

I. Introduction

GRAPH mining [1], [2] is a critical domain within the field of graph queries. Unlike conventional graph processing, which focuses on straightforward tasks like traversal or shortest-path calculations, graph mining aims to locate all embeddings that match specific patterns within a given graph. Due to its high algorithmic demands, irregular memory accesses, intricate data dependencies, and load imbalance [3], graph mining surpasses conventional graph processing in complexity. Hence, accelerating graph mining with conventional processor-centric systems, such as central processing units (CPUs) and graphics processing units (GPUs) presents significant challenges.

Processing-in-memory (PIM) minimizes the cost of data movement by embedding general-purpose or specialized processing elements (PE) in or close to the memory module. This data-centric architecture notably reduces latency and enhances throughput. Recently, there has been a surge in research focusing on accelerating graph processing with hardware and software co-designs in PIM architectures [4], [5], [6]. Besides graph processing, PIM also demonstrates potential in accelerating graph mining [7], [8]. However, fully and effectively exploiting PIM for accelerating graph mining still presents several challenges, including:

Difficulty in Utilizing Locality: Graph mining datasets are commonly stored using the compressed sparse row (CSR) format [3]. However, the nature of accessing vertices and edges in CSR often leads to irregular access patterns [7]. This complexity is further exacerbated in graph mining, which relies on a set-centric programming model [8]. Within this model, each embedding necessitates accessing all edges between its internal vertices and every newly extended vertex. Whenever a neighboring vertex is extended, the connectivity between that vertex and all of the embedding's vertices must be verified.

Challenge of Exploring Parallelism: PIM architectures offer the potential for various levels of parallelism optimization, including subarray, bank, rank, and channel-level parallelism [1]. However, graph mining algorithms often involve complex data dependencies and irregular computation patterns. These complexities make it challenging to fully leverage the potential parallelism provided by PIM in graph mining applications. In particular, for the utilisation of subarray level

Received 29 August 2024; revised 11 March 2025 and 28 April 2025; accepted 6 May 2025. Date of publication 14 May 2025; date of current version 24 November 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62488101, Grant 62495104, and Grant 62025404; and in part by the Youth Innovation Promotion Association CAS. This article was recommended by Associate Editor K. Olcoz. *(Corresponding author: Xiaoming Chen.)*

Liang Yan, Xiaoming Chen, Xingqi Zou, and Yinhe Han are with the Institute of Computing Technology, Chinese Academy of Sciences, and the University of Chinese Academy of Sciences, Beijing 100190, China (e-mail: yanliang19b@ict.ac.cn; chenxiaoming@ict.ac.cn; zouxingqi@ict.ac.cn; yinhes@ict.ac.cn).

Sheng Xu is with the School of Computer and Information, Anhui Normal University, Wuhu, 241000 Anhui, China (e-mail: xusheng2019@ahnu.edu.cn).

Xiaoyang Lu and Xian-He Sun are with the Department of Compute Science, Illinois Institute of Technology, Chicago, IL 60637 USA (e-mail: xlu40@iit.edu; sun@iit.edu).

Digital Object Identifier 10.1109/TCAD.2025.3570164

parallelism, none of the recent work has delved into this fine-grained parallelism study [7], [8].

Complexity in Workload Offloading: High-performance and energy-efficient computation can be achieved by properly partitioning and offloading the workloads and data between PIM and the host CPU [9]. However, offloading is far from straightforward, especially for graph mining applications. Unpredictable data dependencies, combined with irregular access patterns in graph mining and the differing internal processing frequencies between the PIM and CPU, pose significant challenges in determining the optimal offloading strategy.

In response to the aforementioned challenges, we introduce ProMiner, a novel co-designed software and hardware framework for graph mining on a real-world, general-purpose multi-PIM configuration. First, we introduce a graph mining-driven partitioning approach. This approach involves partitioning the entire graph and grouping tightly connected vertices and their associated edges into the same subdataset, thereby enhancing data locality. To minimize the communication overhead between subdatasets, we further design a data placement strategy to handle the problem of cut-vertices resulting from the partitioning. Second, we implement a three-step coarse-fine parallelism optimization scheme to explore parallelism across different memory levels. To enhance the inter-PIM parallelism, we design a task scheduler that allocates tasks from distinct subdatasets across multiple PIM units. By doing so, it capitalizes on the minimal interdependencies among the subdatasets, thereby enabling the simultaneous mining of different subdatasets on separate PIM units. In leveraging the inter-PE parallelism, we design the PE scheduler to switch tasks to the most suitable PE to enhance the parallelism and reduce memory stall time. Furthermore, we optimize the subarray parallelism by adopting a locality-friendly address mapping strategy. Third, we propose a concurrency-aware performance estimation mechanism to strategically guide our workload offloading decisions. By considering both data access locality and concurrency, this mechanism precisely evaluates and predicts the performance bottlenecks inherent in mining individual subdatasets.

We make the following contributions in this article.

1) We introduce ProMiner, a hardware and software co-designed framework that accelerates graph mining within a multi-PIM architecture.
2) ProMiner improves the locality in graph mining applications through graph mining-driven partitioning, explores multiple levels of coarse-fine parallelism, and determines the preferable computing engine using a concurrency-aware performance estimation mechanism.
3) Our extensive evaluation studies show that ProMiner achieves 48.8% and 29.9% execution time reduction over the state-of-the-art NDMiner [8] and DIMMining [7], respectively.

II. Background

A. Graph Mining

Graph mining aims to identify all unique embeddings E that are isomorphic to a specified input pattern within an input graph. The input graph G is composed of vertices and edges and is commonly stored in CSR format. An input pattern P is user-defined. An embedding E is said to be isomorphic to the pattern P if a one-to-one mapping exists between all the vertices and edges of the pattern and the embedding. Graph mining algorithms often employ a DFS approach to enumerate embeddings in the input graph G that match the user-defined pattern P [1]. By applying symmetry breaking [2], a technique designed to prevent the detection of the same embedding E in multiple iterations, these algorithms ensure completeness and uniqueness. All embeddings E that are isomorphic to the pattern P are identified singularly and precisely.

Algorithm 1 Pseudocode for Triangle Counting (TC)

```
1: procedure count_triangles (G)                    ◁ a Graph in CSR format
2: count = 0
3: for u in V do          ◁ V : Vertex set of G, {u}: single-vertex embedding
4:     u_neighbors = get_neighbors (u, G)   ◁ Neighbors of the vertex u
5:     for v in u_neighbors do   ◁ {u, v} : two-vertex embedding
6:         if u < v then ◁ Neighborhood filtration for symmery breaking
7:             v_neighbors = get_neighbors(v, G)  ◁ Neighbors of the vertex v
8:             common_neighbors = interset (u_neighbors, v_neighbors)
                                                          ◁ Set intersection
9:             for w in common_neighbors:    ◁ {u, v, w} : three-vertex
               embedding
10:                if v<w then ◁ Intersection filtration for symmetry breaking
11:                    count ++
12: return count
```

Algorithm 1 delineates the pseudo-code for counting triangles (3 clique finding) in a graph G. The algorithm employs a nested iteration structure. Starting with a primary vertex u (line 3), it fetches the set of its neighbors (line 4). The algorithm then iterates through these neighbors, using vertex v as the reference (line 5). To ensure uniqueness in the triangle and avoid counting the same triangle multiple times, the algorithm checks the condition (line 6). For each valid v, its neighboring vertices are fetched (line 7), and the intersection of the neighboring sets of u and v provides the common neighbors (line 8). The algorithm subsequently iterates over these common neighbors using vertex w (line 9) and checks the condition to ensure the uniqueness of the triangle (line 10). Every valid triangle discovered increments the counter (line 11). Finally, the accumulated count representing the number of unique triangles is returned (line 12).

B. Processing-in-Memory

PIM architectures minimize data movement by enabling computations near the data. 3-D memory technologies, such as hybrid memory cube (HMC) and high-bandwidth memory (HBM) [10], facilitate PIM by integrating memory dies with compute logic within the same package, thereby achieving high-memory bandwidth and reduced latency.

Given the continuous expansion of real-world graphs, a single PIM unit can encounter challenges in accommodating the entire dataset and its associated computational footprint. To tackle this, we adopt a mesh topology for interconnecting the PIM units, recognized as a memory-centric network [6]. Inspired by architectures, such as Tesseract [5] and GraphP [6], we consider a generalized PIM architecture, as depicted in Fig. 1(a). It consists of multiple PIM units

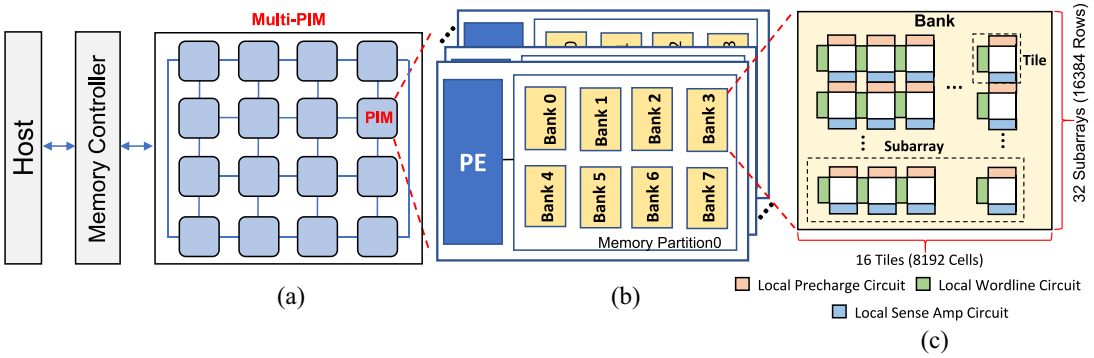


Fig. 1. Multi-PIM architecture overview. (a) PIM units connected in a 4×4 mesh topology with host side. (b) Single PIM with multiple PEs and banks. (c) Subarray in the bank with multiple tiles.

linked by high-bandwidth external connections. Each PIM encompasses several memory partitions and PEs. As shown in Fig. 1(b), these memory partitions consist of multiple banks. The connection between a PE and its respective memory partition is facilitated through through-silicon vias (TSVs), ensuring robust memory bandwidth. This bandwidth scales proportionally with capacity, underlining the scalability of the architecture. Similar to traditional DRAM configurations, these banks typically feature multiple subarrays, further segmented into tiles, as illustrated in Fig. 1(c). In this work, ProMiner remains adaptable, not imposing limitations on PIM design or the interconnection topology of PIM units.

PIM for Graph Mining: Graph mining algorithms are inherently memory-intensive. Mining large-scale real-world graphs can take several hours or even days on modern multicore processors [1]. In response to this challenge, there has been a growing trend in utilizing PIM for graph mining applications. For instance, NDMiner [8] embeds computing units within a buffer chip, aiming to minimize in-memory data transfers. It also introduces an graph remapping strategy in memory alongside a hardware-centric set operation reordering method. It is geared toward optimizing parallelism at the bank, rank, and channel levels within DRAM. However, NDMiner's parallelism optimizations are coarse-grained and do not address parallelism at the subarray level in DRAM. DIMMining [7] is a DIMM-based PIM solution, proposes an index precomparison mechanism to enhance symmetry breaking efficiency. It features a flexible bitmap combined with a CSR format, which enables greater parallelism for set operations from a data structure perspective. Nevertheless, substantial memory may still be required to store these structures. Moreover, both NDMiner and DIMMining overlook workload offloading; they merely transfer set operations in graph mining to PIM for computation, without considering the memory access behavior of these operations. ProMiner draws inspiration from these approaches, yet distinguishes itself as a comprehensive hardware-software co-designed framework, uniquely positioned to accelerate graph mining within a multi-PIM architecture.

C. Graph Partitioning

Graph partitioning plays a crucial role in the efficient distribution of graph datasets across multiple subdatasets, aiming to minimize interprocess communication overhead and ensure a balanced workload distribution. The initial overhead of graph partitioning constitutes an investment that significantly benefits subsequent graph mining tasks. Once partitioned, these subgraphs can be reused across multiple applications involving the same graph. This reusability effectively amortizes the partitioning overhead over multiple applications. Second, efficiently partitioned graphs optimize processing by reducing communication overhead and ensuring evenly distributed workloads. These enhancements not only offset the initial overhead but also significantly improve the overall efficiency and speed of subsequent graph mining operations. Thus, graph partitioning has become a common and efficient approach for long-term analysis of large graph datasets [11], [12].

There are two main strategies of graph partitioning: 1) vertex-cut partitioning and 2) edge-cut partitioning. Emerging research [13] underscores the efficacy of vertex-cut partitioning over traditional edge-cut methods. This preference stems from the observation that computational burdens in graph-centric applications are primarily driven by edges rather than vertices [14].

However, no matter the type of graph partitioning approach, it cannot be directly applied to graph mining applications due to the issue of incomplete neighbor information at the cut-vertices which cannot preserve the precision of graph mining processes. In this work, we first propose a comprehensive solution, graph mining-driven partitioning approach based on Distributed Neighbor Expansion (NE) [13], to effectively address this problem and fully leverage the significant locality benefits brought by graph partitioning.

D. Concurrent Memory Access Model

In modern processors, concurrency is widely adopted to alleviate the impact of memory access latency. Various techniques have been proposed to utilize data concurrency in modern processors and memory systems, such as out-of-order (OoO) execution, multiport cache, nonblocking cache, and pipelined cache. Therefore, a large amount of data concurrency exists in each layer of the memory hierarchy. In order to capture all types of concurrent memory accesses and quantify their combined impact on locality and concurrency, the concurrent memory access model, Concurrent-AMAT (C-AMAT) [15], has been introduced.

C-AMAT extends average memory access time (AMAT) [16] to quantitatively measure the combined

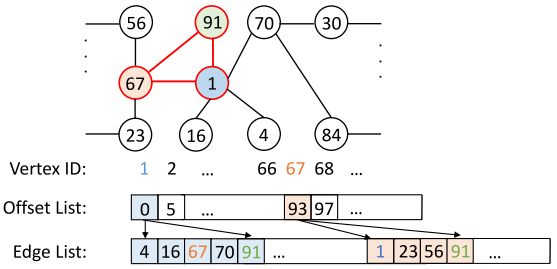


Fig. 2. Example of CSR traversing to find a triangle in graph.

impact of memory access locality and concurrency with the consideration of all data access overlapping [17]. C-AMAT is defined as the average real-time consumed per memory access, calculated as the memory active cycles divided by the number of memory accesses. In a specific memory hierarchy layer, the memory active cycles denote cycles with active memory accesses. They are counted in an overlapping mode, so if multiple memory accesses occur within a single cycle, only one cycle is counted.

C-AMAT is general and can be applied to each level of the memory hierarchy. Intel's performance monitoring units (PMUs) [18] provide performance counters to measure the number of active cycles and the number of data accesses, which can be used directly to calculate the C-AMAT value for a specific level. C-AMAT has already played an important role in guiding the optimization of memory systems [9], [19], [20]. In this study, C-AMAT is employed to identify performance bottlenecks in graph mining workloads, with these insights crucially guiding our offloading strategy decisions.

III. Motivation

A. Insufficient Locality Improvement With PIM

To overcome the challenges posed by poor locality in graph mining, we turn to the PIM architecture. However, due to the more irregular accesses to the CSR in graph mining compared with graph processing, the PIM architecture alone struggles to fully utilize the locality [3], [8].

Fig. 2 is an example of triangle finding. Typically, graph mining traverses in the order of the vertex index. Following the steps in Algorithm 1, we first find the neighbor set of vertex ① and subsequently the neighbor set of vertex ⑥⑦, where ⑥⑦ is a neighbor of ①. This sequence reflects the CSR accesses depicted in the figure. It can be seen that since the vertex indices of ① and ⑥⑦ are very different, this means that these two vertices and their associated neighbor sets are stored far apart in the offset and edge lists, resulting in poor locality. Additionally, whenever a neighboring vertex is extended, the connectivity between this vertex and all of the embedding's vertices needs verification [3]. Consequently, numerous random edge accesses arise, further complicating the locality issue.

Therefore, a more insightful method is required to enhance the locality in graph mining. As mentioned in Section II-C, the graph partitioning can cluster closely connected vertices and edges. This is highly advantageous for graph mining locality optimization since graph partitioning can drastically reduce the irregular accesses to the CSR.

B. Parallelism Challenges in Graph Mining

The reasons for the lack of parallelism in graph mining can be explored at both software and hardware levels. For software, most graph mining algorithms follow the DFS traversal order. The intrinsic dependencies between neighboring search tree levels can lead to prolonged memory stalls, which, in turn, reduce parallelism. As illustrated in Algorithm 1, for each newly expanded vertex v in line 5, the PE is required to fetch its neighbor set (v_neighbors) from memory to compute the result of commom_neighbors in line 8. Only after this computation can the PE proceed to the next level in line 10. In essence, each level must be explored in a DFS order. If the neighbor set is not present in the cache, the PE is left idle and must wait until the required data becomes available. For hardware, while graph mining applications can enhance parallelism across various memory levels—such as channels, bank groups, and banks—using memory interleaving techniques, attaining improvement in parallelism at a finer granularity like the subarray level remains challenging without specific hardware modifications.

C. Workload Offloading in Graph Mining

In PIM architectures, effective workload offloading between the CPU and PIM is crucial for optimal overall performance. While PIM offers extremely high-bandwidth and alleviates memory-bound problems, the processing performance within memory is significantly lower than that of the CPU. On the other hand, for graph mining algorithms, existing studies [7] indicate that due to the extensive set operations involved in graph mining, the application is not only constrained by memory but also demands more compute capability compared to traditional graph processing. Therefore, it is crucial to consider the performance disparities between the CPU and PIM to maximize overall system performance benefits.

Most prior studies focused on optimization specific on the PIM side, simply assigning set operations to PIM for execution [8]. Furthermore, recent studies [1], [7] have shown that graph mining has a significant parallel potential with appropriate parallelism optimizations while there exists a large amount of data concurrency. However, the impact of data concurrency on offloading is often overlooked by traditional offloading strategies.

IV. Design of ProMiner

We present ProMiner, a novel software-hardware co-designed framework for graph mining. ProMiner is tailored to enhance data locality, exploit multiple levels of coarse-to-fine parallelism, and intelligently offload workloads, all of which collectively optimize the efficiency of multi-PIM architectures for graph mining.

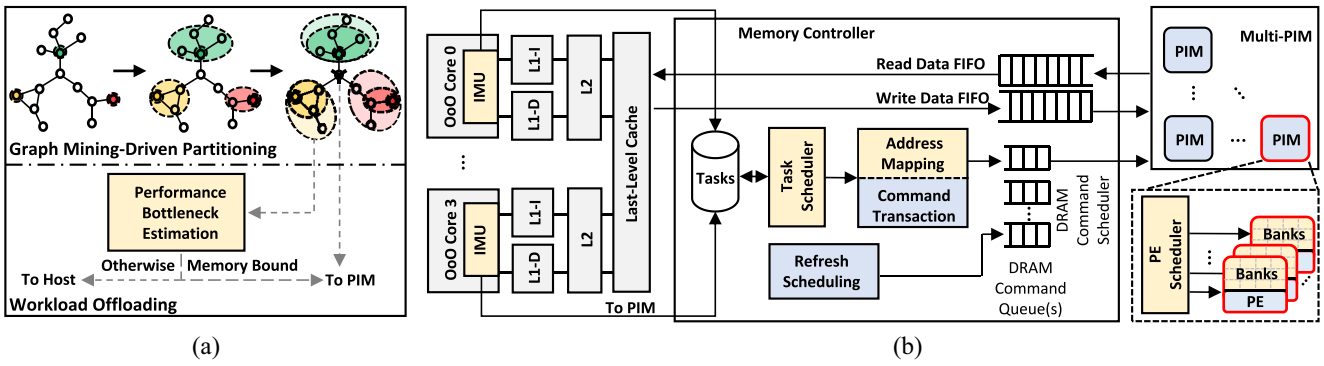


Fig. 3. Overview of ProMiner. (a) Software domain: graph mining-driven partitioning and workload offloading with performance bottleneck estimation. (b) Hardware side: architecture of ProMiner with host and multi-PIM.

A. Overview

Fig. 3(a) illustrates the effects of ProMiner in the software domain. We introduce a graph mining-driven graph partitioning approach to segment the entire graph into subdatasets. This approach aims to minimize communication overhead between these subdatasets and subsequently enhance data locality (detailed in Section IV-D). Additionally, we develop a concurrency-aware performance bottleneck estimation mechanism, which predicts the memory access patterns of each subdataset to facilitate more efficient workload offloading (detailed in Section IV-F).

The overall architecture of ProMiner is depicted in Fig. 3(b). The architecture of ProMiner consists of two main components: 1) the host and 2) the multi-PIM. On the host side, there are four OoO CPU cores. To enable seamless offloading of workload instructions to the PIM or CPU, we embed an IMU within each CPU core. Each core is equipped with its own L1 and L2 caches, while all cores share a last-level cache (LLC). A task scheduler, integrated into the memory controller, is designed to leverage inter-PIM parallelism [5]. This scheduler allocates the mining tasks within a subdataset to the appropriate PIM. The PIM units are configured in a 4×4 mesh topology. Each PIM consists of stacked DRAM banks, PEs, and a corresponding PE scheduler. PE schedulers, designed to maximize inter-PE parallelism, can strategically schedule tasks to the most appropriate PEs. Additionally, modifications have been made to the subarray controller circuit. This enables concurrent access to multiple subarrays within a singular bank, further enhancing the benefits of subarray parallelism. The details of the coarse-to-fine parallelism optimization across all levels are discussed in Section IV-E.

B. ProMiner Workflow

Upon the arrival of a graph mining application, ProMiner embarks on a detailed processing journey.

Graph Mining-Driven Graph Partitioning: ProMiner begins by partitioning the graph using a tailored approach. Conventional methods, such as those presented in [13], may overlook intricate patterns associated with cut-vertices. To address this, ProMiner proposes the cut-vertex neighbor package (CVNP) that encapsulates the N-dimensional neighbor set of the cut-vertex, ensuring that vertices in related subdatasets maintain an encompassing view of their neighboring vertices. Moreover, by employing a Manhattan distance-based data placement strategy, ProMiner optimally allocates the vertices in CVNP to the most suitable PIM, effectively minimizing inter-PIM communication overheads. Notably, for each dataset, our proposed partition method can be reused for any pattern mining by simply changing the dimension of the neighbor set of cut-vertex contained in the CVNP. This reusability effectively amortizes the partitioning overhead across various pattern mining applications for each dataset.

Performance Bottleneck Estimation and Workload Offloading: Once the graph partitioning finishes, two primary vertex categories emerge within each subdataset. These categories are determined by the graph partitioning process: cut-vertices and subdataset vertices. Due to their extensive inter-PIM communication requirements, workloads related to cut-vertices are directly offloaded to the multi-PIM. For the workloads of subdataset vertices, we observe that the C-AMAT values of a few sampled vertices closely approximate those of the entire subdataset vertices (detailed in Section IV-F). Leveraging this observation, ProMiner employs the C-AMAT model to estimate and pinpoint performance bottlenecks, subsequently guiding offloading decisions. The offloading determination is based on a comparison between the LLC C-AMAT value for sampled subdataset vertices and the DRAM access time. If the C-AMAT value exceeds the DRAM access time, the workload is directed to the multi-PIM; if not, it is offloaded to the CPU. This decision-making process is enhanced by the IMU seamlessly integrated within each CPU core.

Coarse-Fine Parallelism Optimization: For workloads offloaded to the multi-PIM, we decompose the execution of mining from a vertex into multiple tasks. We define a task as the work to extend a new vertex from the current partial embedding. ProMiner explores parallelism enhancements across different levels: inter-PIM, inter-PE, and subarray. For inter-PIM parallelism, the task scheduler adopts a round-robin method, systematically directing tasks of a subdataset to an available PIM. The inter-PE parallelism optimization is spearheaded by the PE scheduler, allocating tasks to PEs based on the specific set length for each task. To exploit parallelism across subarrays, ProMiner refines the subarray controller circuit, enabling concurrent accesses to multiple

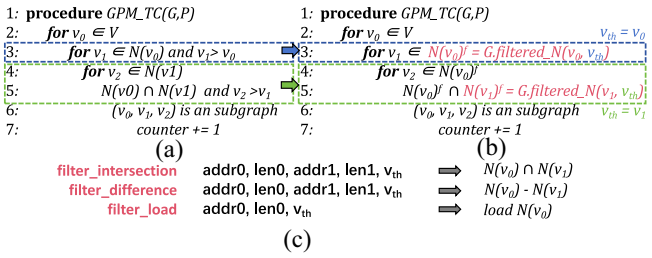


Fig. 4. Programming interface. (a) Original TC Code. (b) ProMiner TC Code. (c) Host ISA instructions to support PIM.

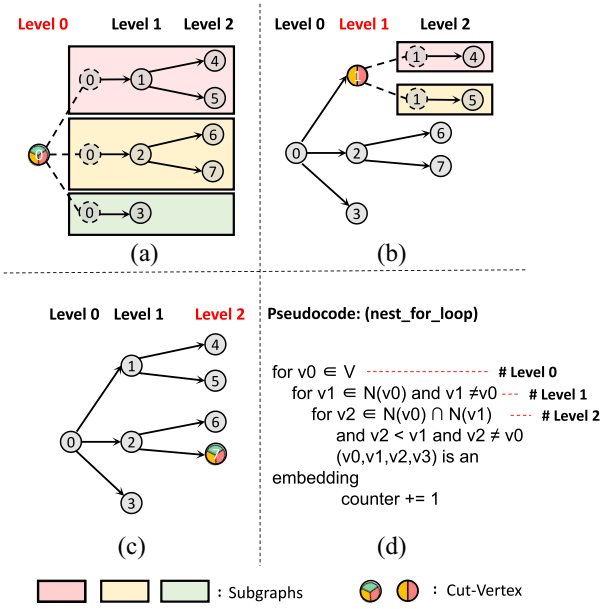


Fig. 5. Cut-vertex at different searching levels. (a) Cut-Vertex at Root Level. (b) Cut-Vertex at Intermediate Level. (c) Cut-Vertex at Last Level. (d) Pseudocode of Triangle Counting.

subarrays within a single bank. Furthermore, a comparison unit is strategically positioned after each subarray row buffer, aiming to enhance the efficiency of symmetry breaking.

C. Programming Interface

In order to facilitate effective communication of graph mining operations to memory, ProMiner introduces a specialized compiler that transforms original graph mining code into PIM-friendly code via the host CPU. After our C-AMAT-based workload offloading finished, the workload suitable for PIM will be transferred to PIM-friendly codes. As depicted in Fig. 4(a) and (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 ProMiner [21]. ProMiner's compiler uses a CUDA-like annotation system for task partitioning. Users simply annotate code sections, and the system automatically identifies which parts run on the CPU and PIM. The CPU code follows standard compilation, while PIM code leverages custom host ISA extensions tailored for PIM.

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 design (details to be provided in Section IV-E), is determined at runtime by the host CPU and then communicated to the PIM units. These 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.

D. Graph Mining-Driven Partitioning

Graph mining-driven partitioning within ProMiner introduces two pivotal innovations. The first involves maintaining the CVNP for each cut-vertex to ensure the integrity and completeness of each vertex's neighbor set across subdatasets. The second innovation is the adoption of a Manhattan distance-based data allocation strategy, optimizing data placement. These enhancements not only improve the precision of graph mining applications but also reduce inter-process communication overhead, significantly enhancing the overall efficiency.

Cut-Vertex Neighbor Package: Traditional graph partitioning techniques segment extensive graph datasets into smaller subdatasets at cut-vertices. Graph mining algorithms depend on the neighbor set of each vertex for identifying patterns. In this setup, cut-vertices are replicated across subdatasets, but this replication can lead to each copy having a limited view of its neighbor set, confined to its subdataset. This limitation can introduce inaccuracies in pattern recognition.

To overcome this challenge, ProMiner integrates the complete N-dimensional neighbor set into the CVNP for every cut-vertex. As depicted in Fig. 5, determining N is crucial for ensuring pattern recognition accuracy. For instance, in the case of a 3-clique pattern, the pseudo code shown in Fig. 5(d) establishes three loop levels, each corresponding to the number of vertices in the pattern. At loop level 0, a cut-vertex requires a 2-D neighbor set for accurate pattern detection. Following this logic, cut-vertices at loop levels 1 and 2 necessitate 1-D and 0-D neighbor sets, respectively. Consequently, for a 3-clique pattern, a minimum of a 2-D neighbor set becomes indispensable. This requirement leads to a general formula for N, given as $N = \alpha - 1$, where α is the number of vertices in the pattern being analyzed. Considering the CVNP of each cut-vertex ensures that the cut-vertex at any level has the necessary neighbor set depth to maintain mining accuracy.

While generating CVNPs for cut-vertices in graph mining is crucial, their effective storage is equally important. Two predominant storage strategies are evident. The first strategy entails distributing CVNPs across all PIM units that require them. This strategy mirrors the replica method observed in traditional graph processing, as referenced in [6]. However, implementing this approach in graph mining poses challenges due to the substantial size of CVNPs, leading to significant storage overhead [22]. Alternatively, storing a single CVNP within a designated PIM, with other PIM units accessing it as needed, reduces memory capacity demands but increases inter-PIM communication. To minimize the inter-PIM communication overhead, CVNPs can be strategically assigned to the

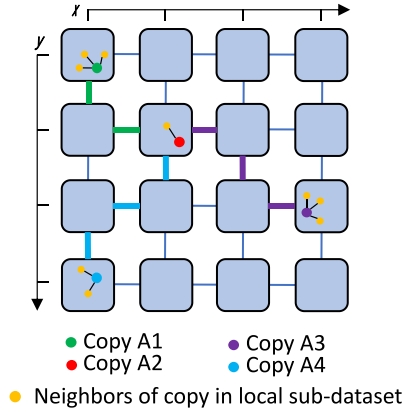


Fig. 6. Example of using Manhattan distance to determine CVNP location.

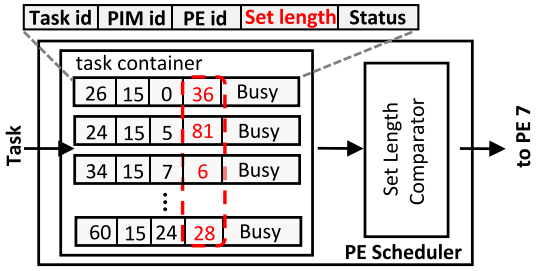


Fig. 7. Example of using PE scheduler to determine which PE the incoming task to allocate.

most suitable PIM units. To facilitate this, we introduce a data placement strategy based on Manhattan distance calculations for the allocation of CVNPs.

Manhattan Distance-Based Data Placement for CVNP: To preserve the accuracy of graph mining while addressing the challenge of memory capacity associated with replicating CVNPs among multiple PIM units, ProMiner opts to store a single CVNP within the PIM hosting one of the pertinent subdatasets and proposes a Manhattan distance–based data placement strategy for mitigating the communication overhead. The Manhattan distance–based data placement strategy involves two critical factors: the overhead of a single communication and the number of communications requested. Under our architectural design, the single communication overhead is quantified using the Manhattan distance metric [23]. The number of communications requested, on the other hand, is primarily influenced by the number of neighboring vertices of the cut-vertex copy present in the subdataset.

By applying this Manhattan distance-based approach, ProMiner can strategically choose a PIM to house the CVNP in a way that minimizes overall communication overhead with other PIM units. This method balances the need for efficient data access with the constraints of memory capacity and inter-PIM communication, ensuring that ProMiner operates effectively within the multi-PIM environment.

Fig. 6 illustrates an example of determining the optimal CVNP location. Suppose a vertex A is a cut-vertex, which is partitioned into four copies, and each copy is located in one of the four subgraphs, i.e., copy A1-A4. And these four subgraphs along with the four copies are assigned to the PIM location shown in the figure. The CVNP location can be possibly in the four PIM corresponding to the four cut-vertex copies. Due to the topology logic of the PIM is fixed, we can easily obtain the single communication overhead between these subgraphs, characterized by the distance between PIM units. However, as mentioned earlier, the communication overhead of CVNPs does not depend only on the overhead of a single communication, but also on the number of communications. The number of communications is related to the number of neighboring nodes of the copy in the subgraph in which it is located. The yellow vertex next to each copy in Fig. 6 represents its 1-D neighbors, and for simplicity of illustration, we only consider 1-D neighbors here. Now we assume that the CVNP is located at A2, then its single communication overhead with A1, A3, and A4 are 2, 3, and 3, respectively. The number of communication overheads are 3, 3, and 2, respectively. Based on this, we conclude that the total communication overhead for the CVNP, when located at A2, amounts to 21. And so on we get the communication overheads for the CVNPs located at A1, A3, and A4, respectively. Finally we arrive at the optimal CVNP location in the PIM.

E. Coarse-Fine Parallelism Optimization

In ProMiner, we introduce a coarse-fine parallelism optimization that leverages various memory levels to exploit the abundant parallelism inherent in graph mining. Inter-PIM parallelism allows for mining different subdatasets simultaneously, rather than strictly following a DFS processing order. Inter-PE parallelism, leveraging intelligent scheduling, allows efficient parallel processing of multiple tasks between PEs. Subarray-level parallelism goes a step further by parallelizing individual memory accesses, enabling concurrent accesses within a single bank.

Inter-PIM Parallelism: Data dependencies primarily constrain the inter-PIM parallelism among different PIM units [24]. Following our partitioning, the vertices within a subdataset exhibit high interconnectivity, while maintaining minimal connections with vertices in other subdatasets. Consequently, we design a task scheduler that adopts a round-robin method to distribute the tasks across the 4×4 multi-PIM configuration, where each PIM is designated to execute tasks from a singular subdataset. By utilizing the task scheduler, data dependencies between PIM units are substantially reduced, allowing for concurrent mining of different subdatasets across PIM units.

Inter-PE Parallelism: PE units are primarily utilized for the set intersection and subtraction (I/S) operations in graph mining. Notably, these operations exhibit higher complexity compared to the value computations typical of traditional graph processing. Furthermore, due to the skewed distribution of datasets, the set lengths of neighbors span a wide range.

To enhance inter-PE parallelism, ProMiner features a PE scheduler located adjacent to the PEs within a PIM. This PE scheduler comprises two primary components: the task container and the set length comparator, as depicted in Fig. 7. Each item in the task container encapsulates details, such as the task id, PIM id, PE id, set length, and status. The set length is derived from the sum of the two neighboring set lengths associated with each I/S operation. Task statuses are

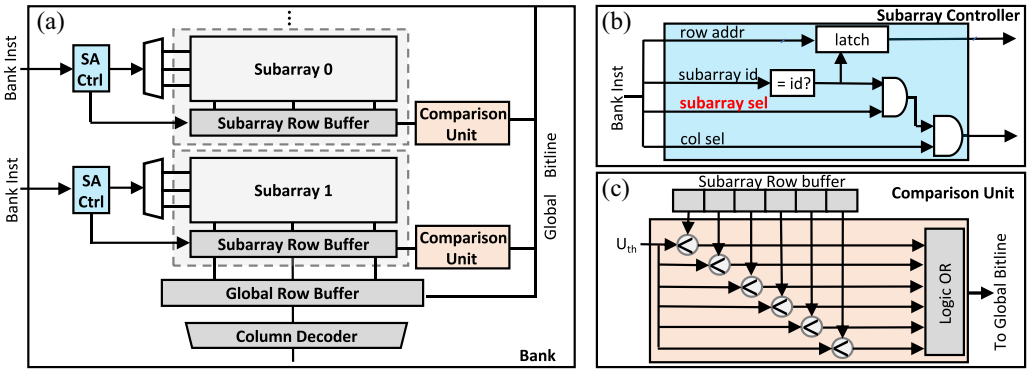


Fig. 8. Hardware design for the subarray level parallelism in ProMiner. (a) A bank with multiple subarrays. (b) Subarray controller design. (c) Comparison unit design.

categorized as `ready`, `busy`, or `completed`. For instance, in Fig. 7, a new task with a `ready` status arrives while all current tasks in the container are marked as `busy`. Using the set length comparator, it is determined that task 34, with a set length of 6, is likely to complete first. Once task 34 concludes its operation and its status updates to `completed`, the incoming task is immediately slated for execution on PE 7. This approach effectively reduces inter-PE downtime and enhances the parallelism of set operations.

Subarray Parallelism: The first two parallelism optimizations primarily concentrate on selecting the appropriate PIM or PE for task execution. However, once a task is allocated to a specific PIM and PE, enhancing parallelism at a finer granularity in hardware level becomes vital for optimizing overall performance. ProMiner emphasizes subarray parallelism optimization, introducing two key techniques: locality-friendly mapping and customized subarray design.

After implementing the graph mining-driven partitioning, vertices within a subdataset display notable interconnectivity. As a result, tasks associated with this subdataset are allocated to an appropriate PIM through the task scheduler. Due to the inherent interconnectedness of a subdataset, executing tasks within it benefits from superior data locality. In a standard PIM architecture, multiple banks exist within a PIM, each containing several subarrays, as shown in Fig. 8(a). One consideration of our subarray parallelism design is how to store data in the subarray through a dedicated mapping strategy. The conventional address mapping strategy disperses the vertices of a subdataset among various banks and their respective subarrays. Such a distribution strategy might be beneficial for enhancing parallelism in standard graph mining applications that do not employ graph partitioning. However, in the context of ProMiner, a conventional interleaving approach risks undermining the locality advantages gained through graph partitioning. Our key idea for data storage in subarrays is first storing the data in different subarrays of the same bank, and then moving to other banks when the bank is full, which fully leverages the locality benefits offered by graph mining-driven partitioning. ProMiner introduces a refined, locality-friendly address mapping strategy. In Fig. 9, both the details of the conventional address mapping and locality-friendly address mapping are illustrated.

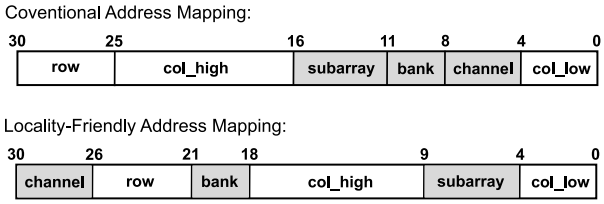


Fig. 9. Conventional and locality-friendly address mapping.

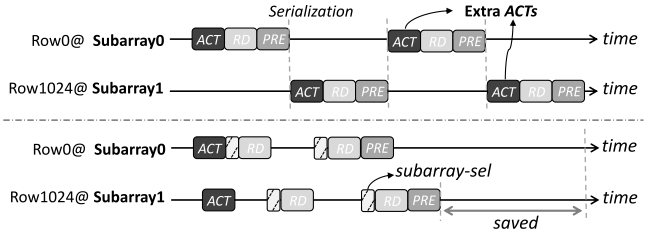


Fig. 10. Execution of four successive read accesses to two different subarrays in same bank using conventional subarray configuration (up) and ProMiner's customized subarray configuration (bottom).

In Fig. 9, both mapping strategies map data at the burst length granularity, which corresponds to the width of the column low bits, and referred to as a data block. In the conventional mapping strategy, once the column low bits are full, the channel bit flips, meaning that subsequent data is mapped to a different channel. Thus, the conventional mapping strategy arranges the data blocks across different channels, banks, and subarrays, which could compromise the locality of the subdatasets generated by the graph mining-driven partitioning, and lead to excessive data movement. Our proposed locality-friendly mapping strategy will map the data block first to the same bank but to different subarrays, maintaining data locality without affecting parallelism at the bank and channel levels. To further enhance parallelism, we also implement lightweight circuit modification that increases parallelism between subarrays within the same bank.

In traditional DRAM architectures, upon word-line activation, data is immediately transmitted to the global row buffer to circumvent potential short circuits. This immediate data relay monopolizes the global bitlines, constraining activation to a single subarray. Motivated by the insights from [25], we

decouple subarray activation from data transmission, which permits the concurrent activation of multiple subarrays. We achieve that by temporarily storing the data in the local row buffer. The memory controller then designates a single activated subarray to command the global bitlines and transfer the data to global row buffer during the next column command.

To realize the subarray-level parallelism, alterations are incorporated into the subarray controller circuit, controlling the data transition from the local to the global row buffer. As Fig. 8(b) shows, the subarray controller is designed to control the data transmission from the local to the global row buffer. A specific row connects to the local row buffer upon activation by the row address (row addr) and subarray id. It connects to the global row buffer only when designated by the subarray selection (subarray sel) signal. In this way, multiple subarrays in a bank can be activated simultaneously and temporarily stored in the local row buffer. Only the designated subarray is connected to the global bit-lines at any given time, ensuring safe and correct operation.

Illustrated in Fig. 10, consider four requests accessing two distinct rows within the same bank but in separate subarrays. Traditional PIM configurations, without subarray optimization, necessitate four activations. In contrast, ProMiner requires a mere two. Given that activation time constitutes a significant portion of DRAM access time, this activation concurrency markedly bolsters memory access performance. The proposed partitioning ensures that the vertices accessed are predominantly employed in ensuing memory accesses, thus maintaining a commendable row buffer hit rate.

Furthermore, a comparison unit is integrated just after the subarray row buffer to enhance the parallelism of the symmetry breaking process in graph mining. The logic blueprint of the comparison unit is presented in Fig. 8(c). This unit integrates a comparator suite to compare the vertex index retrieved from DRAM banks with the threshold vertex index extracted from the memory request. Upon meeting the symmetry breaking criteria, the comparison unit emits a signal, precluding additional loads. By positioning the comparison unit post the subarray row buffer, we exploit heightened symmetry breaking parallelism at the subarray level.

F. Performance Bottleneck Estimation and Workload Offloading

ProMiner employs a concurrency-aware mechanism for estimating performance bottlenecks. This mechanism evaluates the performance of subdataset vertices, guiding the decision-making process for workload offloading.

Offloading Granularity: In this work, we select the loop body within graph mining applications as the primary code block for workload offloading. ProMiner provides flexibility in loop identification. Users have the option to either manually annotate loops in the source code or rely on compiler-driven automatic identification. Our choice of granularity is based on three primary considerations. First, in graph mining, loop bodies, especially those involved in mining intricate patterns, constitute a significant portion of the code and can be pinpointed with ease. Second, the irregular memory access patterns prevalent in graph mining predominantly arise within the loop body. This characteristic positions the loop as a prime candidate for optimization in graph mining applications. Third, applying the granularity of the loop body helps mitigate the substantial overhead tied to pinpointing the appropriate PIM. This overhead escalates when utilizing more refined granularity, such as at the instruction level.

Performance Bottleneck Estimation: To determine the optimal compute engine (CPU or multi-PIM) for mining subdataset vertices, one might instinctively turn to preprocessing. However, the preprocessing approach can introduce significant overheads. Responding to this challenge, we conduct an online analysis of the memory access behavior associated with mining these subdataset vertices and design a programmer-transparent mechanism for performance estimation. The performance estimation mechanism, based on C-AMAT, is specifically tailored to predict potential performance bottlenecks. Consequently, this mechanism enables an informed decision on the most suitable compute engine before executing the complete mining operations on subdataset vertices.

We employ the C-AMAT model to analyze the memory access behavior of subdataset vertices. C-AMAT is a comprehensive performance model that takes into account both data access locality and concurrency. It represents the average access time required for a specific memory level. We observe a consistent distribution of LLC C-AMAT values when do 3-clique finding (3CF) and 4-clique finding (4CF) from varying numbers of sampled subdataset vertices. Fig. 11 showcases the LLC C-AMAT values for mining from different portions of sampled subdataset vertices: 10%, 20%, 50%, and the entirety of the subdataset vertices. We observe that the C-AMAT value of mining from a few sampled subdataset vertices closely mirrors that of the entire subdataset. This observation lends strong support to the design of a performance estimation mechanism. Leveraging the C-AMAT value from the sampling, the mechanism can predict performance bottlenecks, aiding in the determination of the optimal offloading engine for the remaining subdataset vertices.

In this study, we select the first 10% of the vertices from each subdataset as a representative sample for analysis. We operate the mining from the sampled vertices using the CPU, and evaluate the LLC C-AMAT value during the mining process. After obtaining the C-AMAT value from the sampled vertices, we compare it with the DRAM access time to pinpoint potential performance bottlenecks. If the LLC C-AMAT value surpasses the DRAM access time, we classify these sampled subdataset vertices as memory-bound. A higher C-AMAT value indicates that the cache hierarchy fails to offer performance benefits, rendering data retrieval through it less efficient than direct DRAM access. If the sampled subdataset vertices are identified as memory-bound, the preferred decision is to offload the entirety of the subdataset vertices to multi-PIM for execution.

Instruction Management Unit: After partitioning, the mining workload is divided into two parts: 1) mining from subdataset vertices and 2) from the cut-vertices. For the subdataset vertices, we rely on performance bottleneck estimation to determine the execution engine. Conversely, when it comes to

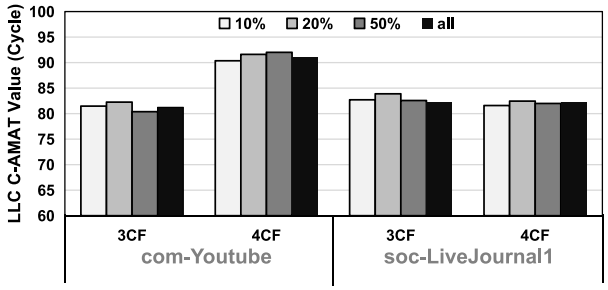


Fig. 11. LLC C-AMAT values across mining varying proportions of vertices for two datasets.

TABLE I
SIMULATION SYSTEM CONFIGURATIONS

Component	Configuration
On-chip Processor	4 cores, OoO, 2GHz frequency
	Private L1 Cache: Separated 32KB I/D Cache per core, 8-ways, 2-cycle hit latency, 64B line size
	Private L2 Cache: 256KB/Core, 32-ways, 12-cycle hit latency, 64B line size
	Shared L3 Cache: shared 2MB/core, 32-ways, 35-cycle hit latency, 64B line size
Memory Model	16 PIM units organized in a 4x4 mesh (Fig.1a)
	4GB, 32 memory partitions per PIM, 256 banks
	32 single-issue processing elements, 500MHz, each per PIM
	L1 I/D Caches: 32KB private, 4-ways, 64B line size
	Bandwidth: TSV 10GB/s, 320GB per cube; 120GB/s perlink

the cut-vertices, due to their complex memory access behavior, we opt to execute this workload directly on the multi-PIM.

ProMiner integrates an IMU in each host core to manage the instruction data path. Instructions not requiring in-memory execution are loaded directly into the L1 cache for processing on the host CPU. Conversely, instructions designated for execution in PIM bypass the cache hierarchy, being directly offloaded to PIM as memory requests. The IMU is coupled to the processor's fetch stage. PIM instructions can be identified during the fetch stage and directly routed to the PIM, preventing them from occupying CPU pipeline resources [26].

V. EVALUATION METHODOLOGY

We integrate ZSim [27] with Ramulator [28] to simulate CPU-PIM systems. We modify ZSim to generate PIM-specific traces. To accurately evaluate the performance and efficacy of our proposed design, we further refined Ramulator. By adjusting DRAM timing constraints, we implement subarray parallelism, allowing us to evaluate the performance impact of simultaneous accesses to the same bank. Table I shows the detailed system configuration. In ProMiner, we utilize the simple in-order cores as PE like HMC to reduce power consumption and area overhead [6], [29]. These in-order cores are single-issue but incorporate SIMD capabilities to enhance parallelism for set operations in graph mining applications.

We select four mining patterns: 1) 3-clique finding (3CF); 2) 4-clique finding (4CF); 3) 5-clique finding (5CF); and 4) 3-motif counting (3MC). As shown in Fig. 12, the first three patterns are clique findings and represent fully connected dense patterns. The last pattern, motif counting, counts all possible patterns with a specified number of vertices. These

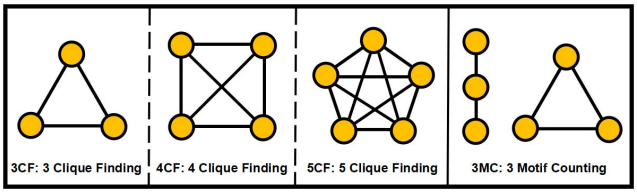


Fig. 12. Input graph patterns used for evaluation.

TABLE II
DATASETS USED IN EXPERIMENTS

Graph	#Vertex	#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

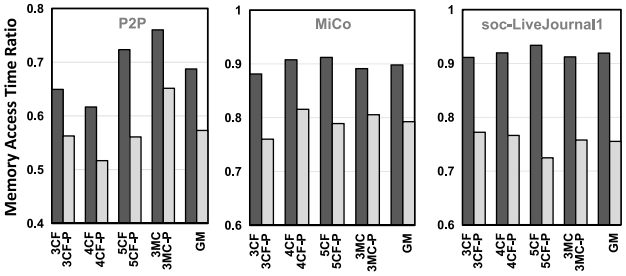


Fig. 13. Proportion of memory access time to total running time.

patterns encompass both dense and sparse patterns [2]. To ensure reasonable simulation times, we limit our mining to patterns with up to five vertices, consistent with prior works [2], [7], [8]. It is worth noting that although our simulations focus on these four patterns, ProMiner is versatile and not limited to any specific mining pattern. It can accommodate any arbitrary user-defined pattern.

We have chosen six real-world graph datasets for performance evaluation, as detailed in Table II. These datasets are often employed in prior research [2], [7], [8]. They vary in size, ranging from small (e.g., P2P) to large (e.g., soc-LiveJournal), and exhibit diverse connectivity patterns. This assortment of datasets effectively underscores the advantages of ProMiner across a spectrum of mining patterns.

VI. EXPERIMENT RESULTS

A. Effectiveness of Graph Mining-Driven Partitioning

To understand the influence of graph mining-driven graph partitioning on enhancing performance, we analyze the memory access time associated with fetching the neighbor sets during graph mining. This evaluation compares the memory access time for graph mining execution with and without utilizing graph mining-driven graph partitioning, as represented by cases, such as CF-P and CF in Fig. 13, providing insights into the effectiveness of this partitioning method. Memory access times are obtained from mining three distinct datasets,

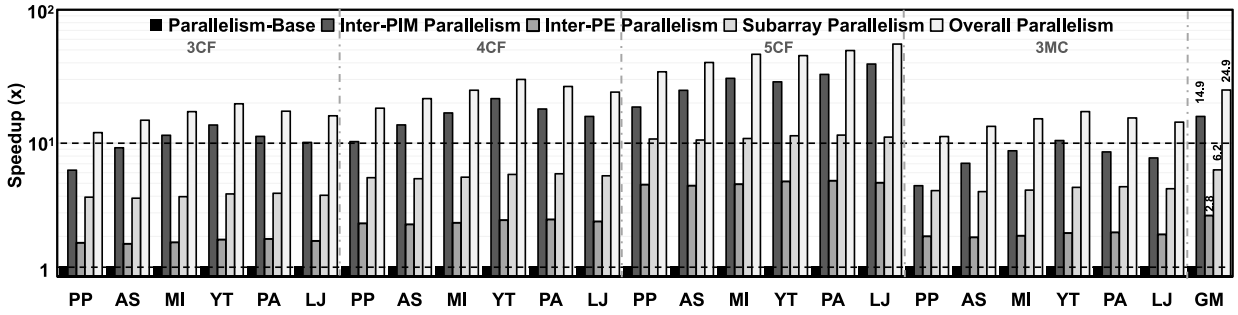


Fig. 14. Comparison of speedups for three different granularity parallel optimizations.

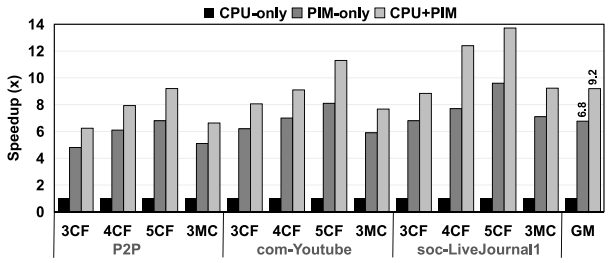


Fig. 15. Speedup comparison across offloading strategies, all based on graph mining-driven partitioning.

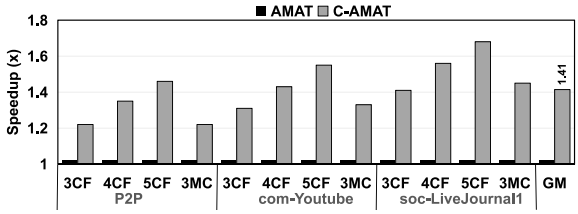


Fig. 16. Speedup comparison across AMAT-based and C-AMAT-based offloading strategies.

covering a range from small to large scales, with all mining conducted on the multi-PIM side.

As demonstrated in Fig. 13, employing graph mining-driven partitioning reduces the time required to access memory for fetching neighbor sets. On average, mining with partitioning spends 70.6% of its memory access time on neighbor sets, which is considerably lower than the 83.5% required without leveraging the graph mining-driven partitioning technique. These results underscore the effectiveness of graph mining-driven partitioning.

B. Comparison of Different Parallelism Optimizations

To quantify the advantages of various parallelism optimizations, we examine the performance gains from each—inter-PIM, inter-PE, and subarray. This analysis follows the application of graph mining-driven partitioning and the offloading of all tasks to multi-PIM. Parallelism-Base serves as the baseline for our parallelism optimization experiments, in which we execute the entire graph mining workload on the PIM side without employing any graph partitioning. We have three observations from Fig. 14. First, on average inter-PIM parallelism provides a 15.9× speedup. By utilizing graph mining-driven graph partitioning, vertices cluster with higher connectivity, which reduces data dependencies among PIM units, thereby boosting inter-PIM parallelism. Second, inter-PE parallelism offers an average improvement of 2.9× speedup over the baseline. This increase is primarily due to task prioritization based on the length of set operations. Such an approach markedly diminishes PE stall time, enhancing parallelism among PEs within the same PIM. Third, the subarray parallelism optimization demonstrates a speedup of 6.3× on average over baseline. This notable improvement underscores the optimization introduced for subarrays, enabling it to effectively support concurrent accesses within DRAM subarrays. Moreover, the integration of a comparison unit alongside each subarray row buffer significantly reduces the symmetry breaking time. As mining patterns increase in complexity, the advantages of subarray parallelism become more pronounced. For example, the speedup of subarray parallelism with dataset LJ for 3CF and 5CF are 4.06× and 11.09×, respectively. This is largely due to efficiencies in row activation and symmetry breaking. It is worth noting that ProMiner's three different granularity parallel optimizations can be enabled simultaneously, and the overall parallelism achieves a speedup by the geometric mean of 24.9× than baseline.

C. Effectiveness of Workload Offloading

We evaluate the efficiency of workload offloading by comparing the speedup of the entire graph mining application across different offloading strategies. To show the efficiency of the offloading strategy separately, the results of this part of the experiment are all based on graph mining-driven partitioning, which means that we determine which engine the subdatasets execute on. Fig. 15 depicts the speedup across three offloading strategies: CPU-only (all subdatasets are executed on CPU), PIM-only (all subdatasets are offloaded to multi-PIM), and CPU-PIM (decide where to execute the subdatasets based on the C-AMAT value). In the CPU-PIM strategy, graph datasets are first segmented into several subdatasets. These subdatasets are then offloaded either to CPU or to PIM, guided by the performance bottleneck estimation introduced in Section IV-F. Generally, the CPU-PIM strategy yields a speedup of 9.2× when compared to the CPU-only approach. In contrast to the PIM-only method, CPU-PIM achieves an average performance boost of 6.8×. Significantly, as input sizes grow and mining patterns become more intricate, CPU-PIM surpasses both CPU-only and PIM-only strategies. For instance, in cases, such as soc-LiveJournal1 and 5CF,

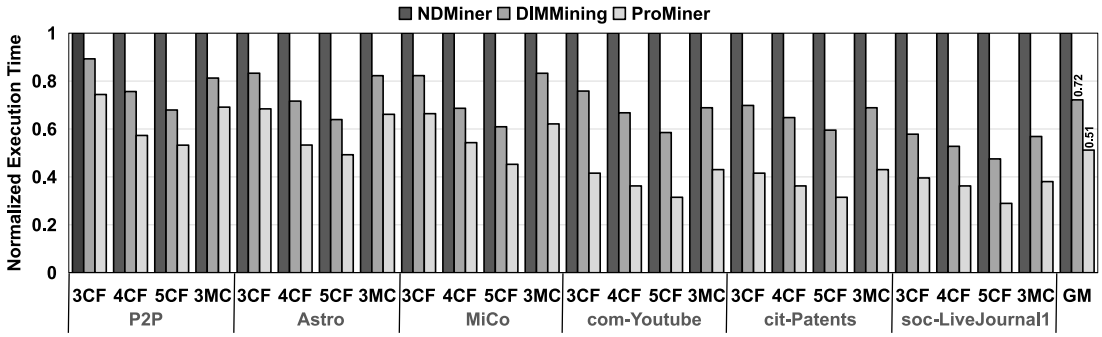


Fig. 17. Speedup comparison of the state-of-the-art schemes.

the speedups are 9.6× and 13.7×, respectively. The C-AMAT model can pinpoint memory-bound workloads that profoundly impact memory efficiency, especially when data access concurrency is critical. Therefore, utilizing the C-AMAT model to estimate the performance of workloads and to guide offloading decisions can offer substantial performance improvements.

Fig. 16 shows that the C-AMAT-based workload offloading strategy surpasses an AMAT-based strategy by 1.41× on average. This difference arises because C-AMAT considers both locality and concurrency, whereas AMAT primarily focuses on locality. As we mentioned in Section IV-E, there is plenty of potential for parallelism in graph mining applications. Therefore, the C-AMAT-based offloading strategy offers more comprehensive guidance for graph mining workload offloading than the AMAT-based strategy does.

D. Performance Comparison With Other Frameworks

As shown in Fig. 17, we compare the performance of ProMiner and the state-of-the-art NDMiner [8] and DIMMining [7]. All results are normalized to the NDMiner execution time. On average, ProMiner achieves 48.8% and 29.9% execution time reduction over NDMiner and DIMMining, respectively. To understand the reasons behind these improvements, we analyze the key architectural differences between ProMiner and the existing designs. First, regarding parallelism granularity, DIMMining employs processing element arrays with systolic array structures to enable parallel computation, while NDMiner utilizes networked computational units to accelerate set operations in graph mining. However, both approaches exhibit limitations in parallelism granularity: their memory access mechanisms exploit parallelism only at the bank level, without leveraging the inherent parallelism potential at the subarray level. Prior studies demonstrate that coordinated memory accesses across 32 subarrays within a single bank can significantly enhance overall parallel efficiency [25], a capability that ProMiner fully exploits through subarray-level parallel optimization. Second, regarding task offloading, existing works uniformly classify set operations as compute-intensive tasks for PIM execution.In contrast, ProMiner introduces a fine-grained task classification mechanism, guided by the C-AMAT model, to accurately distinguish between computation-bound and memory-bound operations. This enables more efficient resource utilization in CPU-PIM heterogeneous systems. Third, regarding PE scheduling, current architectures lack effective solutions to address load imbalance among processing units. ProMiner tackles this challenge with an innovative PE scheduler that dynamically allocates tasks based on set-length comparisons, minimizing PE idle time and improving utilization. Finally, regarding symmetry breaking, DIMMining does not introduce a hardware-based symmetry breaking unit, and its software symmetry breaking method incurs additional time overhead.

TABLE III
DESIGN OVERHEAD COMPARISON

	NDMiner	DIMMining	ProMiner
Area (mm^2)	0.64	0.38	0.21
Power (mW)	51.59	105.82	21.26

E. Overhead Analysis

We present the additional area and power of ProMiner in Table III using Synopsys Design Compiler with 32 nm process. The primary reason for ProMiner's reduced hardware design cost lies in the choice of processing units. Both NDMiner and DIMMining introduce complex dedicated PIM processing units for set operations, leading to considerable additional area and power overheads. Although dedicated computing units can accelerate the speed of set operations, our analysis reveals that the primary bottleneck in graph mining when moved to PIM is memory access rather than computation. Therefore, ProMiner adopts a mature HMC-like general-purpose processor core, enhanced with finer-grained subarray-level memory access parallelism and a more precise set operation workload offloading strategy. This approach significantly reduces design complexity while preserving architectural generality and scalability. The additional overhead of ProMiner primarily originates from the comparison units and entries of task and PE schedulers. Each PIM module requires only $0.031mm^2$ for the comparison units and $0.207mm^2$ for entries. ProMiner contributes an additional overhead of 0.21%, relative to the total area of a typical DRAM chip, which measures around $100mm^2$. Furthermore, the additional power overhead for ProMiner is 21.26mW, considerably lower than that of NDMiner and DIMMining, which require more complex circuits.

VII. Related Work

Besides using PIM to accelerate graph mining applications, there are also some domain-specific accelerators. Unlike traditional graph processing accelerators, which mainly aim to improve irregular memory accesses through memory system optimization [30], [31], graph mining accelerators require more comprehensive design considerations. Recent works have designed specialized architectures for graph mining. FlexMiner [32] improves the performance and generality of graph mining by proposing a pattern-aware GPM accelerator. IntersectX [33] optimizes GPM execution on a CPU by extending the ISA and architecture support. TrieJax [34] improves memory locality by introducing a custom caching mechanism and leverage inherent concurrency of graph mining. Fingers [1] exploit fine-grained parallelism to overcome the issues of hardware underutilization, inefficient resource provision, and limited single-thread performance under imbalanced loads. GRAMER [3] impoves locality and parallelism by introducing a specialized memory hierarchy, the valuable data is permanently resident in a high-priority memory while others are maintained in a cache-like memory under a lightweight replacement policy. Among these accelerators, Fingers, like ProMiner, introduces finer parallelism optimization. However, Fingers primarily employs software methods to uncover fine-grained independent tasks at each tree level and across different branches. ProMiner, on the other hand, introduces both software and hardware methods to enhance parallelism, particularly subarray parallelism, an aspect not addressed by Fingers.

VIII. Conclusion

Irregular memory accesses in graph mining applications make them particularly well-suited for acceleration using PIM. However, fully and effectively exploiting PIM for accelerating graph mining still presents several challenges, including 1) difficulty in utilizing locality; 2) challenge of exploring parallelism; and 3) complexity in workload offloading. In this study, we introduce ProMiner, a multi-PIM framework with hardware-software co-design to accelerate graph mining. We propose a graph mining-driven graph partitioning approach to address the long-standing challenge of poor data locality. We escalate the level of parallelism from coarse-grained to fine-grained, aligning with distinct hardware levels, to exploit parallelism across inter-PIM, inter-PE, and at the subarray level. We develop an intelligent workload offloading strategy that considers both data locality and concurrency, directing workloads to the optimal engine for better acceleration. Experimental results show that ProMiner outperforms state-of-the-art schemes.

References

[1] Q. Chen, B. Tian, and M. Gao, "Fingers: Exploiting fine-grained parallelism in graph mining accelerators," in *Proc. 27th ACM Int. Conf. Archit. Support Programming Lang. Oper. Syst.*, 2022, pp. 43–55.

[2] D. Mawhirter and B. Wu, "Automine: Harmonizing high-level abstraction and high performance for graph mining," in *Proc. 27th ACM Symp. Oper. Syst. Principles*, 2019, pp. 509–523.

[3] P. Yao et al., "A locality-aware energy-efficient accelerator for graph mining applications," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2020, pp. 895–907.

[4] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling instruction-level PIM offloading in graph computing frameworks," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2017, pp. 457–468.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM Sigarch Comput. Archit. News*, vol. 43, no. 3, pp. 105–117, 2015.

[6] M. Zhang et al., "GraphP: Reducing communication for PIM-based graph processing with efficient data partition," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 544–557.

[7] G. Dai et al., "DiMMining: pruning-efficient and parallel graph mining on near-memory-computing," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 130–145.

[8] N. Talati et al., "NDMiner: Accelerating graph pattern mining using near data processing," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, 2022, pp. 146–159.

[9] L. Yan et al., "CoPIM: A concurrency-aware PIM workload offloading architecture for graph applications," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, 2021, pp. 1–6.

[10] J. H. Kim et al., "Aquabolt-XL HBM2-PIM, LPDDR5-PIM with in-memory processing, and AXDIMM with acceleration buffer," *IEEE Micro*, vol. 42, no. 3, pp. 20–30, May/Jun. 2022.

[11] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph processing in a distributed dataflow framework," in *Proc. 11th USENIX Symp. Oper. syst. Design Implement.*, 2014, pp. 599–613.

[12] S. Salihoglu and J. Widom, "Optimizing graph algorithms on pregel-like systems," *Proc. VLDB Endow.*, vol. 7, no. 7, pp. 577–588, 2014.

[13] M. Hanai, T. Suzumura, W. J. Tan, E. Liu, G. Theodoropoulos, and W. Cai, "Distributed edge partitioning for trillion-edge graphs," 2019, *arXiv:1908.05855*.

[14] D. Brickley, M. Burgess, and N. Noy, "Google dataset search: Building a search engine for datasets in an open Web ecosystem," in *Proc. World Wide Web Conf.*, 2019, pp. 1365–1375.

[15] X.-H. Sun and D. Wang, "Concurrent average memory access time," *Computer*, vol. 47, no. 5, pp. 74–80, May 2014.

[16] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam, The Netherlands: Elsevier, 2019.

[17] X.-H. Sun and X. Lu, "The memory-bounded speedup model and its impacts in computing," *J. Comput. Sci. Technol.*, vol. 38, no. 1, pp. 64–79, 2023.

[18] X.-H. Sun, N. Zhang, B. Toonen, and B. Allcock, "Performance modeling and evaluation of a production disaggregated memory system," in *Proc. Int. Symp. Memory Syst.*, 2020, pp. 223–232.

[19] X. Lu, R. Wang, and X.-H. Sun, "CARE: A concurrency-aware enhanced lightweight cache management framework," in *Proc. 29th IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, 2023, pp. 1208–1220.

[20] X. Lu, R. Wang, and X.-H. Sun, "Premier: A concurrency-aware pseudo-partitioning framework for shared last-level cache," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, 2021, pp. 391–394.

[21] H. Ahmed et al., "A compiler for automatic selection of suitable processing-in-memory instructions," in *Proc. Design, Autom. Test Europe Conf. Exhibition (DATE)*, 2019, pp. 564–569.

[22] H. Jin et al., "Accelerating graph convolutional networks through a PIM-accelerated approach," *IEEE Trans. Comput.*, vol. 72, no. 9, pp. 2628–2640, Sep. 2023.

[23] S. Xu, H. Xue, L. Luo, L. Yan, and X. Zou, "DrPIM: An adaptive and less-blocking data replication framework for processing-in-memory architecture," in *Proc. Great Lakes Symp. VLSI*, 2023, pp. 385–389.

[24] Y. Zhuo et al., "GraphQ: Scalable PIM-based graph processing," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 712–725.

[25] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (SALP) in DRAM," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 368–379, 2012.

[26] P. C. Santos, B. E. Forlin, and L. Carro, "Providing plug N'play for processing-in-memory accelerators," in *Proc. 26th Asia South Pacific Design Autom. Conf.*, 2021, pp. 651–656.

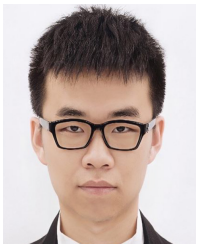
[27] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 475–486, 2013.

[28] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Comput. Archit. Lett.*, vol. 15, no. 1, pp. 45–49, Jan.–Jun. 2016.

[29] J. D. Leidel and Y. Chen, "HMC-Sim-2.0: A simulation platform for exploring custom memory cube operations," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2016, pp. 621–630.
[30] A. Basak et al., "Analysis and optimization of the memory hierarchy for graph processing workloads," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2019, pp. 373–386.
[31] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–13.
[32] X. Chen, T. Huang, S. Xu, T. Bourgeat, C. Chung, and A. Arvind, "FlexMiner: A pattern-aware accelerator for graph pattern mining," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2021, pp. 581–594.
[33] G. Rao, J. Chen, J. Yik, and X. Qian, "IntersectX: An efficient accelerator for graph mining," 2020, *arXiv:2012.10848*.
[34] O. Kalinsky, B. Kimelfeld, and Y. Etsion, "The TrieJax architecture: Accelerating graph operations through relational joins," in *Proc. 25th Int. Conf. Archit. Support Programming Lang. Oper. Syst.*, 2020, pp. 1217–1231.

Liang Yan is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

His current research interest is focused processing-in-memory architectures and graph computing.



Xiaoyang Lu (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, Illinois Institute of Technology (Illinois Tech), Chicago, IL, USA.

He is currently an Research Assistant Professor with Illinois Tech. His research focuses on computer architecture, memory performance modeling, memory performance optimizations, and ML-assisted computer architectures.

Sheng Xu (Associate Member, IEEE) received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2019.

He is currently an Associate Professor with the Anhui Normal University, Wuhu, China. His main research interests include Processing-in-Memory, heterogeneous systems, and the memory system.

Xiaoming Chen (Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2009 and 2014, respectively.

He is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interest is focused on design automation for integrated circuits and PIM architectures.

Xingqi Zou received the Ph.D. degree in microelectronics and solid-state electronics from the University of Chinese Academy of Sciences, Beijing, China, in 2019.

He is an Senior Engineer in Research Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests are Chiplet designe and VLSI design, with an emphasis on processing in memory and AI Chip.

Yinhe Han (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 2003 and 2006, respectively.

He is currently a Professor with ICT, CAS. His main research interests are microprocessor design, integrated circuit design, and computer architecture.

Xian-He Sun (Life Fellow, IEEE) received the Ph.D. degree in computer science from Michigan State University, Michigan, USA.

He is a University Distinguished Professor and the Ron Hochsprung Endowed Chair of the Department of Computer Science with the Illinois Institute of Technology (Illinois Tech), Chicago, IL, USA.

His research interests include highperformance computing, memory and I/O systems, and performance evaluation and optimization.